02-04-2025 жылғы №47 қызметтерді мемлекеттік сатып алу туралы шартқа сәйкес «Балаларға арналған мемлекеттік тілді үйрететін компьютерлік ойын әзірлеу» қызметтері бойынша 3-кезеңі
ЕСЕБІ

Тапсырыс беруші:
Қазақстан Республикасы
Ғылым және жоғары білім министрлігі
Тіл саясаты комитеті

Өнім беруші: «United Kingdom» ЖШС

Астана, 2025 жыл

Мазмұны

Қызмет көрсету кестесі 3
Тестілеу барысында ескерту-ұсыныстарды есепке алу 6
Қорытынды түзетулер енгізу 10
Өнімділікті оңтайландыру 15
Ойыншыларға (пайдаланушыларға) және әкімшілік панельді пайдалануға арналған нұсқаулық әзірлеу 16
Ойынның үш деңгейінің демо-нұсқасын дайындау 23
Деректерді жергілікті және бұлтты сақтау шешімдерін қолдау 32
Ойындар мен қосымшаларды орналастыру платформаларында аккаунттар ашу 37
Steam талаптарына сәйкес ойын платформасын әзірлеу 48
Дайын ойынның таныстырылымын өткізу 51
Steppe rider ойынын платформаларда жариялау (бета-нұсқа) 53
Қорытынды тексерулер 61
Жарнамалық-ақпараттық науқан 64
Ойынды басқару жүйесіне, әкімшілік панеліне қол жеткізуді ұсыну 71
Тапсырыс берушіге барлық бастапқы кодтарды тапсыру 74
Зияткерлік меншік құқықтарын тапсыру 75
Ойынның мобильді нұсқасын ұсыну 76
Релиз туралы анонс жасау 79
Көрсетілген қызмет бойынша толық (шығармашылық және қаржылық) есеп беру 80
2026 жылдың желтоқсан айына дейін тегін техникалық қолдау көрсету 82
Әзірлеу барысындағы қателерді түзету 83
Функционалға өзгеріс енгізбестен ұсақ жетілдірулер жасау 85
Қосымша-1 (Ойынның бастапқы және орындалатын кодтары) 87
Қосымша-2 (Ойын жарияланған платформалардың логиндері мен құпия сөздері) 174
Қосымша-3 (Серверлік бөлігінің конфигурациясы) 177
Қаржылық есеп 179

1. ТЕСТІЛЕУ БАРЫСЫНДА ЕСКЕРТУ-ҰСЫНЫСТАРДЫ ЕСЕПКЕ АЛУ

Steppe Rider жобасының әзірлеу кезеңінде ойынның ДК *(дербес компьютер)* және мобильді нұсқаларын қамтитын толық тестілеу сериясы жүргізілді. Тестілеудің негізгі мақсаты - барлық ойын модульдерінің дұрыс жұмысын, жүйенің тұрақтылығын, интерфейстің ыңғайлылығын және жүзеге асырылған функционалдың техникалық тапсырма талаптарына сәйкестігін тексеру.

Тестілеу бірнеше кезеңнен тұрды:

- модульдік тестілеу (unit-тестирование) - жеке компоненттердің, раннер логикасының, викторина, сөздік және прогресс жүйесінің жұмысын тексеру;
- интеграциялық тестілеу - ойын модульдері, деректер базасы және интерфейс арасындағы өзара әрекеттестікті тексеру;
- жүйелік тестілеу - әртүрлі құрылғыларда өнімнің тұрақтылығын, өнімділігін және визуалды тұтастығын бағалау;
- бета-тестілеу - UX талдау және кері байланыс алу үшін нақты пайдаланушыларды тарта отырып жүргізілетін сынақтар.

Тестілеу шеңберінде келесі аспектілер қарастырылды:

- негізгі ойын режимдерінің (раннер және викторина) жұмысы;
- ұпайлардың, «асықтардың» және жетістіктердің дұрыс есептелуі;
- ойыншы профилінің және көшбасшылар тақтасының жұмыс істеуі;
- пайдаланушы сөздігінің көрсетілуі және жаңартылуы;
- интерфейстің әртүрлі экран өлшемдеріне бейімделуі;
- локализация сапасы және қазақша таңбалардың дұрыс көрсетілуі;
- FPS тұрақтылығы және ресурстарды жүктеуді оңтайландыру.

Тестілеу методологиясы

Тестілеу интерациялық тәсіл арқылы жүргізілді, ол қателерді бірнеше рет тексеру, тіркеу және қайта валидациялау циклдерін (Regression Testing) қамтыды. Тестілеу функционалдық және функционалдық емес талаптарға негізделген чек-листер мен тест-кейстерді пайдалана отырып қолмен өткізілді.

Жұмыс барысында келесі құралдар мен әдістер қолданылды:

- **TestRail / Google Sheets** - есеп жүргізу және чек-листерді толтыру;
- **Unity Profiler** - өнімділікті талдау және графиканы оңтайландыру;
- **ADB (Android Debug Bridge)** - логтар, FPS және жедел жад көрсеткіштерін бақылау;
- **Screen Recording және Crash Logs** - визуалды және функционалдық ақауларды тіркеу;
- **User Feedback Forms** - бета-тестерлерден пікірлер жинау.

Ойынның нақты қолдану жағдайларына сәйкес тұрақтылығы мен оңтайландырылуын анықтау мақсатында өнімділік бағасы әртүрлі санаттағы құрылғыларда (бюджеттік смартфондардан бастап ортадан жоғары деңгейлі ДК-ге дейін) жүргізілді.

Жүргізілген сынақтардың нәтижесінде барлық критикалық және орта деңгейдегі қателер түзетілді, ал ойын регрессиондық тестілеуден сәтті өтті. Қазіргі нұсқа тұрақтылық, өнімділік және пайдаланушылық тәжірибе талаптарына толық сәйкес келеді және релизге әрі кейінгі масштабтауға дайын.

Тестілеудің жалпы нәтижелері:

Көрсеткіш	**Мәні**
Барлығы анықталған қателер	***50***
Түзетілген	***50***
Критикалық қателер	14
Орта деңгейлі қателер	13
Төмен деңгейлі қателер	13
Қайта тестілеу (Regression)	Толық жүргізілді, барлық тексерулер сәтті өтті

Критикалық қателер (барлығы түзетілді)

Сипаттамасы және нәтижелері:

- **Порталдардың қате көрсетілуі:** портал ойын басталған сәтте бірден пайда болатын - түзетілді, белсендіру логикасы қайта қаралды.
- **Жеңіл деңгейде порталардың қайталануы:** псевдокездейсоқ таңдау алгоритмі енгізілді.
- **«Е» әрпін тану қатесі - сөздерді жинаудың бұғатталуы:** кирилл таңбаларын талдау түзетілді.
- **«Ғаламжан» скинын таңдағанда ойынның жабылуы:** анимациялық файл мен текстура арасындағы конфликт жойылды.
- **Қойындылар арасында тез ауысқанда қара экран шығуы:** UI күйін өңдеу оңтайландырылды.
- **«Сатып алу» батырмасы жұмыс істемейтін:** экономика модулімен байланыс түзетілді.
- **Ойыншы аты енгізілмесе де кейіпкердің қозғала беруі:** профиль күйін тексеру қосылды.
- **Тірілуден кейін анимация қателері:** кейіпкер контроллері түзетілді.
- **Сөздер жиналғаннан кейін жоғалмай қалуы:** сөздік жаңарту логикасы түзетілді.
- **Викторинадағы «Артқа» батырмасы ойынды қайта іске қосуы:** UI өзгертілді.
- **Тілді ауыстырғаннан кейін қиындық деңгейлерінің қақтығысы:** локализация модулі қайта жасалды.
- **Кейіпкерлердің ойынға кіргенде қатып қалуы:** анимацияның қайталанатын оқиғалары жойылды.
- **Сөздік жаңармауы:** деректердің автоматты жаңартылуы енгізілді.
- **Қайта іске қосқанда жаңа сөздердің шықпауы:** сөздік жүктеу логикасы түзетілді.

Орта деңгейлі қателер (барлығы түзетілді)

- Мәзірде көмекші құралдар мен ақпараттық хабарламалардың болмауы - **tooltips қосылды**.
- Батырмалардағы мәтіндердің дұрыс аударылмауы немесе қате көрінуі - **қос тілді қолдау іске асырылды**.

- Мәтіндердің дұрыс орналаспай, қиылып кетуі - **UI-контейнерлер жаңартылды**.
- Скиндер сақталмауы - **аватар кэші түзетілді**.
- Кейіпкерлерде idle-анимацияның болмауы - **анимация циклдері қосылды**.
- Қалған әріптермен сөз генерациясының логикасы қатесі - **алгоритм қайта жасалды**.
- Интерфейс визуалындағы ұсақ ақаулар (шекара қалыңдығы, көлеңкенің жоқтығы) - **түзетілді**.
- Порталдар мен объектілердің орналасуы - **оңтайландырылды**.
- Викторина жаңартылды: **сөздер лимиті төмендетілді және жауап таңдау жеңілдетілді**.
- Лидерлер тақтасында қателер - **түзетілді**.
- Жиналған сөздер санын көрсету үшін **визуалды индикаторлар қосылды**.

Төмен деңгейлі қателер (барлығы түзетілді)

- Бұталар мен кіші объектілердің жерге «жайылып» кетуі - **түзетілді**.
- UI-шекаралардың жеткіліксіз қалыңдығы - **түзетілді**.
- Кейбір қазақша таңбалардың дұрыс көрсетілмеуі - **түзетілді**.
- Кей экрандарда көмекші мәтіннің аймаққа сыймауы - **түзетілді**.
- Кей элементтердің кейіпкерді визуалды түрде жауып қалуы - **түзетілді**.
- Секіру анимациясы кейде камера шолуын жауып қалуы - **түзетілді**.
- Скин анимациясының дұрыс көрсетілмеуі - **түзетілді**.

Локализация және интерфейсті тестілеу

- Екітілді локализация толық жүзеге асырылды (қазақша / орысша).
- Барлық мәтіндер, батырмалар, қалқыма кеңестер және сипаттамалар тексерілді.
- Тілді ауыстыру қайта бастаусыз жұмыс істейді.
- Интерфейс 1280×720-дан 4K-ға дейінгі экран рұқсаттарына бейімделген.
- UI элементтерінің қолжетімділігі тексерілді (контраст, қаріп өлшемі, оқылымдылық).

2. ҚОРЫТЫНДЫ ТҮЗЕТУЛЕР ЕНГІЗУ

Толық тестілеу циклі мен пайдаланушылар және тапсырыс берушіден алынған кері байланысты талдау нәтижелері бойынша Steppe Rider ойынына тұрақтылықты арттыру, өнімділікті жақсарту және интерфейс қолайлылығын көтеруге бағытталған финалдық түзетулер енгізілді.

Негізгі түзетулер қамтыды:

- **Өнімділікті оңтайландыру** - деңгейлердің жүктелу уақыты қысқартылды, текстуралар, анимациялар және фондық процестер әртүрлі сипаттамалары бар құрылғыларда тұрақты жұмыс істеу үшін оңтайландырылды.
- **Интерфейсті (UI/UX) түзету** - батырмалардың өлшемдері мен орналасуы қайта өңделді, қалқып шығатын кеңестер, прогресс бойынша визуалды индикаторлар және белсенді элементтердің подсветкасы қосылды.

- **Ойын модульдерінің логикасын түзету** - сөздер мен порталдарды генерациялау алгоритмдері жаңартылды, викторинаның жұмысы және пайдаланушы сөздігімен деректерді синхрондау жақсартылды.

- **Көптілді қолдауды жетілдіру** - аудармалар нақтыланды, қазақ тілінің арнайы таңбаларының (ә, ө, ү, ң, қ, ғ) көрсетілуі түзетілді.

- **Анимациялар мен визуалды эффектілерді жақсарту** - тегіс өткелдер, көлеңкелер және визуалды қабылдауды арттыратын ұсақ элементтер қосылды.

- **Дыбыстық сүйемелдеуді жаңарту** - дыбыс деңгейінің балансы реттелді, қысқа сигналдар мен әрекеттерді растау дыбыстары қосылды.

- **Деректерді сақтау және жаңартуды оңтайландыру** - ойынның қайта ашылуын қажет етпей, сөздік пен жетістіктерді автоматты жаңарту іске асырылды.

- **Лидерлер тақтасы мен жетістіктер панелін жетілдіру** - нұсқау қосылды, оқылымдылығы жақсартылды, визуалды иконкалар жаңартылды.

- **Релиз нұсқасының финалдық тестілеуі** - құрастырудың техникалық тапсырмаға және платформалардың (Steam, Android, IOS) талаптарына сәйкестігі тексерілді.

Енгізілген өзгерістер нәтижесінде пайдаланушылық тәжірибенің жалпы сапасы жақсарды, қосымшаның тұрақтылығы артты және өнімнің қоғамдық релизге әрі демонстрацияға дайындығы күшейді.

3. ӨНІМДІЛІКТІ ОҢТАЙЛАНДЫРУ

Әзірлеудің соңғы кезеңінде дербес компьютерлерде де, мобильді құрылғыларда да тұрақты және бірқалыпты жұмысты қамтамасыз ету үшін Steppe Rider ойынының өнімділігін оңтайландыру бойынша жұмыстар кешені жүргізілді.

Оңтайландыру барысында келесі іс-шаралар орындалды:

- **Графикалық ресурстарды оңтайландыру** - визуалды сапаны жоғалтпай қолданылатын текстуралар мен спрайттардың көлемі азайтылды. Құрылғының жұмысына байланысты динамикалық ажыратымдылықты басқару енгізілді.
- **Жүктеу уақытын қысқарту -** көріністерді инициализациялау және ассеттерді кэштеу механизмдері қайта қаралды, соның арқасында деңгейдің орташа жүктеу уақыты 35-40% - ға қысқарды.
- **Кадр жиілігін тұрақтандыру (FPS)** - анимацияларды, жарық эффектілерін және фондық бөлшектерді теңестіру; Android жүйесінде 58-60 FPS және компьютерде 60+ FPS тұрақты кадр жиілігі қамтамасыз етілген.
- **Логика мен есептеулерді оңтайландыру** - ойын циклдарының құрылымы қайта қаралды, әр кадрдағы «қоңыраулар» саны азайды, Unity физикалық қозғалтқышының жұмысы оңтайландырылды.
- **Жедел жадқа жүктемені азайту** - пайдаланылмаған объектілерді жадтан түсіру жүйесі іске асырылды, бір уақытта белсенді анимациялар мен дыбыстардың саны азайды.
- **Дыбыстық ішкі жүйені оңтайландыру** - аудио файлдарды қысу қолданылды, эффектілерді ойнату кезінде кідірістер жойылды, фондық музыка мен дауыстың қабаттасуы оңтайландырылды.
- **Ұзақ ойын сеанстарында тұрақтылықты арттыру** - 60 минутқа дейін созылатын стресс-тесттер өткізілді, бұл жадтың «ағып кетпеуін» және өнімділікті арттырды.
- **Әр түрлі қуаттылықтағы құрылғыларға бейімделу** - сапаның автоматты алдын ала орнатулары (төмен, орташа, жоғары) енгізілді, бұл көптеген заманауи смартфондар мен ноутбуктерде тұрақты жұмыс істеуге мүмкіндік берді.

Жұмыстың нәтижесі жалпы өнімділікті айтарлықтай арттырды, мобильді құрылғылардың қуат тұтынуын төмендетті, сондай-ақ бірқалыпты және ыңғайлы геймплейге қол жеткізілді.

4. ОЙЫНШЫЛАРҒА (ПАЙДАЛАНУШЫЛАРҒА) ЖӘНЕ ӘКІМШІЛІК ПАНЕЛЬДІ ПАЙДАЛАНУҒА АРНАЛҒАН НҰСҚАУЛЫҚ ӘЗІРЛЕУ

Пайдаланушы тәжірибесін жақсарту және жаңа ойыншылардың бейімделуін жеңілдету үшін Steppe Rider ойынына алғаш кірген кезде автоматты түрде іске қосылатын оқу экраны модулі (tutorial slides) енгізілді.

Бұл интерфейс элементі **onboarding** жүйесінің бөлігі болып табылады және ойыншыға алғашқы секундтардан бастап басқарудың негізгі принциптерін, ойын мақсаттарын және негізгі механиканы түсінуге бағытталған.

Нұсқаулық экрандарын енгізу мақсаттары:

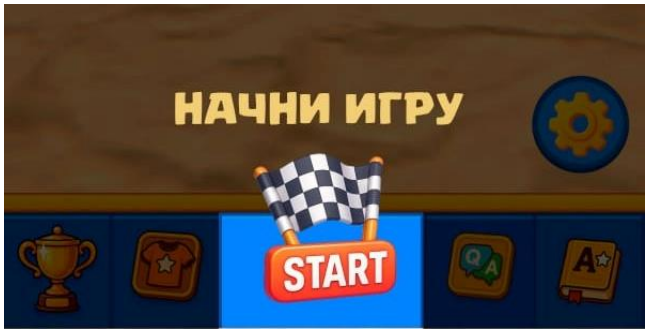
- Жаңа пайдаланушыларды жеке нұсқаулыққа жүгінбестен ережелер мен интерфейспен таныстыру.
- Алғашқы ойын сеанстарындағы қателерді азайту.
- Түсінікті және жағымды алғашқы әсер арқылы ойыншылардың қатысуы мен сақталуын арттыру.
- Геймплейге және логикалық оқу дәйектілігіне біркелкі кіруді қамтамасыз ету.

Нұсқаулық экрандарының құрылымы мен мазмұны

Нұсқаулық визуалды слайдтар сериясынан тұрады, олардың әрқайсысы қысқа мәтіндік кеңестермен негізгі ойын әрекеттерін көрсетеді. Барлық иллюстрациялар жобаның визуалды стилінде - жылы түстермен, үлкен қаріптермен жасалған.

Бастау экраны ("Ойынды бастаңыз")

- Басты кейіпкер мен оның серігі барысты таныстырады.
- Геймплей басталатын "**Start**" батырмасының мақсатын түсіндіреді.
- Эмоционалды тонды орнатады және ойыншыны қазақ даласы әлемімен таныстырады.



"Далада жүгіріп, қажетті әріптерді жина" экраны

- Раннердің негізгі механикасын көрсетеді: кейіпкердің дала бойымен қозғалысы және әріптер жинау.

- Ұсынылған нұсқалардың ішінен қазақ тілінде сөз құрастыру үшін дұрыс әріптерді қалай таңдау керектігін көрсетеді

- Басқару батырмаларын (жебелер, свайп қимылдары) жарықпен белгілейді.

«Тек дұрыс әріптерді таңда» экраны

- Дұрыс таңдау маңызды екенін түсіндіреді: қате болған жағдайда сөз есепке алынбайды.

- Әр әріп сөзді аяқтауға және дауыстық оқуға жақындататынын көрсетеді.

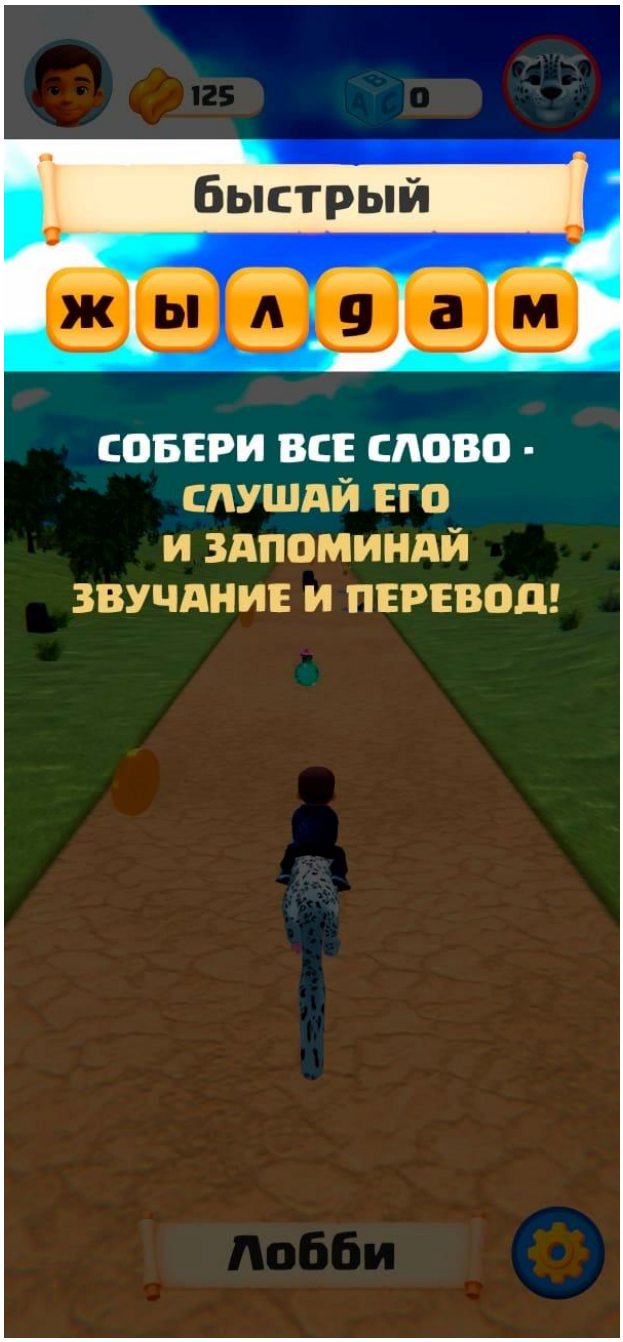
«Кедергілерден аулақ бол» экраны

- Деңгейдің динамикалық элементтерімен таныстырады - кедергілер, жануарлар, тастар, өсімдіктер.
- Олармен соқтығысу - барыстың денсаулығын азайтатынын түсіндіреді.
- Жоғарғы панельде денсаулық индикаторы визуалды көрсетіледі.

«Барлық әріптерді жина - тында және есте сақта» экраны

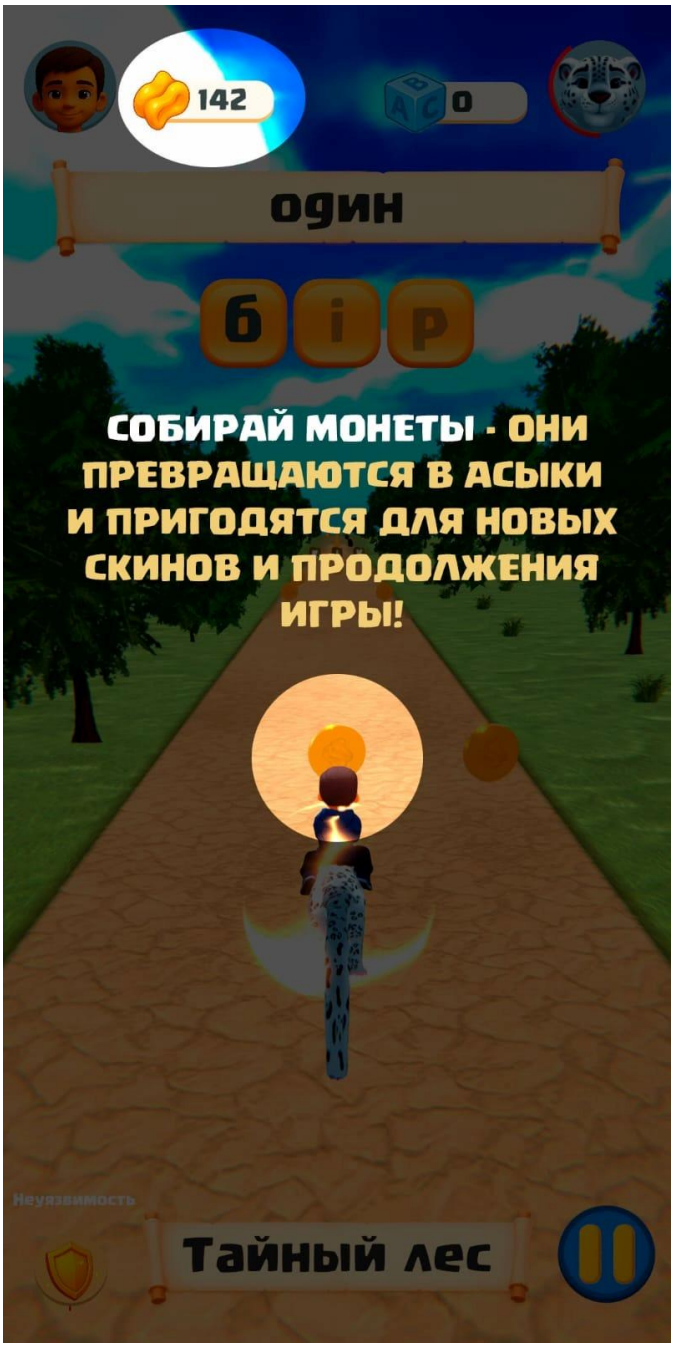
- Барлық әріптерді сәтті жинағаннан кейін ойыншы сөздің қазақ тіліндегі дұрыс айтылуын естиді.

- Мәтіндік блок визуалды және аудиалды қабылдауды біріктіре отырып, аудармасын және айтылуын есте сақтауға болатынын еске салады.



«Асықтарды (тиындарды) жина» экраны

- Ойын ішіндегі валюта - «асықтардың» мәнін түсіндіреді.
- Асықтарды - скиндер сатып алу, ойынды жалғастыру және жаңа деңгейлерді ашу үшін пайдалануға болатынын көрсетеді.
- Валюта есептегішінің визуалды элементін көрсетеді.



«Үнемі ойна - жаңа тақырыптарды аш» экраны

- Ойыншыны ойынды қайта оралуға ынталандырады.
- Белсенді ойнаған жағдайда жаңа локациялар мен тақырыптардың жылдам ашылатынын хабарлайды.
- Қол жетімді деңгейлердің әртүрлілігін көрсетеді (орман, дала, шөл, таулар).

5. ОЙЫННЫҢ ҮШ ДЕҢГЕЙІНІҢ ДЕМО-НҰСҚАСЫН ДАЙЫНДАУ

Steppe Rider жобасын әзірлеудің соңғы кезеңінде ойынның негізгі механикасы мен визуалды тұжырымдамасын көрсететін үш ойын деңгейін қамтитын демонстрациялық нұсқасы дайындалып, іске асырылды.

Демо-нұсқа бастапқы тестілеуге, тұтынушыға көрсетуге, презентациялауға және оқу құралы ретінде әрі қарай пайдалануға арналған.

Бұл кезеңнің негізгі міндеті - негізгі ойын жүйелерінің дайындығын, қосымшаның тұрақтылығын көрсету және пайдаланушы үшін біртұтас, аяқталған тәжірибе шеңберінде негізгі ойын мазмұнын енгізу.

Демо-нұсқаны әзірлеу мақсаттары

- Ішкі тестілеу және сыртқы көрсету үшін соңғы өнімге мүмкіндігінше жақын жұмыс прототипін беру.
- Ойын механикасының тұрақтылығын, қозғалтқыштың өнімділігін және әртүрлі құрылғыларға (Android, IOS, Windows) оңтайландыруды тексеру.
- Интерфейстің ыңғайлылығы мен оқу ережелерінің түсінігін бағалау.
- Платформаның әртүрлі қиындық деңгейлері мен тақырыптарына бейімделуін көрсету.
- Тесттен өтушілерден және мақсатты аудиториядан кері байланыс алу.

Демо-нұсқа тек көрнекі стиль мен геймплейді көрсетіп қана қоймай, ойынның білім беру әлеуетін және қазақ тілінің лексикасын интерактивті түрде меңгеру әдістемесін нақты көрсетуі тиіс.

Демо-нұсқаның құрамы мен құрылымы

Демонстрациялық құрастыру ойыншыға бейімделуге және тұтынушыға геймплей эволюциясы мен қиындықтың өсуін көруге мүмкіндік беретін біртіндеп күрделену принципі бойынша құрылған үш ойын деңгейін қамтиды.

1 деңгей сипаттамасы - "Жеңіл"

Steppe rider ойынының демо-нұсқасының бірінші деңгейі ойыншыны интерфейспен, басқарумен және негізгі геймплей механикасымен алғашқы таныстыруға арналған. Ол қарапайым және түсінікті оқыту сценарийі форматында орындалған, ол пайдаланушыда жүйемен өзара қарым-қатынастың базалық дағдыларын қалыптастырады және қазақ тіліндегі алғашқы сөздік минимумын бекітеді.

Деңгейдің жалпы жұмыс принципі

Экранда ойыншы көреді:

- орысша сөз - аударма (мысалы, "Кошка"),
- әріптер жиынтығынан қазақ тілінде дұрыс сөз жасау қажет ("Мысық"),
- кейіпкердің қозғалыс бағытын басқаруға арналған көрнекі батырмалар (солға/оңға),
- кейіпкер әріптер мен асықтарды жинап, үнемі алға жылжитын интерактивті жол.

Ойын процесі ассоциативті оқытуға негізделген: ойыншы сөздің орыс тіліндегі аудармасын көреді және есте сақтау мен көрнекі белгілерге назар аудара отырып, оның баламасын қазақ тілінде жинауы керек. Сәтті аяқталғаннан кейін сөзді диктор айтады және визуалды түрде белгіленеді, бұл дұрыс айтылу мен емлені есте сақтауға көмектеседі.

Интерфейс пен геймплейдің негізгі элементтері

1. **Сөз-тапсырма:**
   Экранның жоғарғы жағында ойыншыға кеңес ретінде көрсетілетін орысша сөз орналастырылған. Ол жеңіл фонда визуалды ерекшеленген, оқуға ыңғайлы.

2. **Әріп-блоктар:**
   Төменде қазақ әліпбиінің әріптерімен жарқын, интерактивті блоктар орналасқан. Ойыншы оларды дұрыс ретімен таңдай отырып, жолдарды пайдаланып жинайды.

3. **Кейіпкер мен серіктес:**

Ойыншы басты кейіпкерді басқарады, ол Барыстың үстінде отыр - ойынды символдайтын кейіпкер. Барыс ойыншының таңдауларына жауап береді, жылдамдайды немесе кедергілерден аулақ болады, бұл динамика мен қызығушылық қосады.

4. **Ойын ортасы:**

Локация жылы құм түстерінде жасалған, Ұлы шөлді бейнелейді. Деңгей бойында ойыншы әртүрлі қоршаған орта объектілерін көреді - тастар, өсімдіктер, жануарлар.

5. **Тиындар және кедергілер:**

Жолда ішкі ойын валютасын жинауға болатын тиындар бар. Сондай-ақ, кедергілер кездеседі (жыландар, тастар), денсаулықты жоғалтпау үшін олардан аулақ болу керек.

6. **Индикаторлар мен кеңестер:**

Жоғарғы панельде жиналған асықтар саны және Барыстың денсаулығы көрсетіледі. Экранның төменгі жағында локация атауы («Ұлы шөл») және белсенді эффектілер («Алтын ×2») көрсетіледі.

Оқу әсері

- Ойыншы геймплейдің негізгі принциптерін меңгереді: қозғалыс, әріптерді жинау, кедергілерден аулақ болу.

- Орысша сөз бен қазақша аудармасы арасындағы байланыс бекітіледі.

- Визуалды және аудио сүйемелдеу дұрыс айтылуды және орфографияны есте сақтауға көмектеседі.

- Қиындық минималды, бұл қазақ тілін алдын ала білмейтіндер үшін де ыңғайлы бастауға мүмкіндік береді.

Деңгейді іске асыру ерекшеліктері

- Қажетті әріптердің интерактивті жарықтандырулары (дұрыс жауапты визуалды ерекшелеу) іске асырылды.

- Сөз сәтті жиналғаннан кейін қазақ тілінде дауыстап ойнатылады, бұл тыңдау арқылы қабылдауды ынталандырады.

- Жетістік анимациясы қосылды (фанфарлар, жарқыл, дыбыстық сигнал).
- Деңгей арнайы экранмен аяқталады: жиналған сөздер саны, асықтар және нәтижелер.

Деңгейдің педагогикалық мәні

Бірінші деңгей интерактивті сабақ ретінде қызмет етеді:

- Ойыншы сыртқы нұсқаусыз - әрекет, бақылау және дыбыс арқылы үйренеді.
- Негізгі сөздік қалыптасады (жануарлар, заттар, түстер, әрекеттер тб).
- Орысша және қазақша сөздердің эквиваленттері арасындағы тұрақты ассоциация құрылады.
- Геймификация арқасында оқуды жалғастыру мотивациясы артады.

Қорытынды

Бірінші деңгей Steppe Rider-дың оқыту концепциясының негізі болып табылады. Ол ойын элементтері мен білім беру құрамын біріктіріп, тілді үйрену процесіне жұмсақ кіріспе береді.

Қарапайым тапсырмалар, көрнекі кеңестер және жарқын визуализация арқасында ойыншы механиканы тез түсінеді және алғашқы оң нәтижелерге қол жеткізеді, бұл келесі, күрделі деңгейлерге өтуге мотивация қалыптастырады.

2-деңгей сипаттамасы - «Орташа»

Steppe Rider демо-нұсқасының екінші деңгейі базалық оқытудан ойыншының жүйемен күрделірек өзара әрекет ету формасына өтуін көрсетеді. Осы кезеңде ойыншы сөзде қандай әріптің жетіспейтінін өзі анықтауы керек.

Бұл деңгей практикалық жаттығу ретінде қызмет етеді: алынған білімді бекітеді, зейінділікті дамытады және қазақ сөздерін контексте белсенді меңгеруге ықпал етеді.

Деңгейдің жалпы принципі

Экранда көрсетіледі: орысша сөз, мысалы, «короткий», және қазақ тілінде жартылай жасырылған сөз («қысқа»), оның бір әрпі түсіп қалған.

Ойыншы локация бойымен жолда жүгіретін кейіпкерді басқарады. Жол бойында әріп нұсқалары пайда болады, олардың ішінде тек біреуі дұрыс. Сөзді жинауды жалғастыру үшін ойыншы дұрыс әріп орналасқан жолды таңдауы керек.

Қате таңдау - айыппұл: кейіпкер ұпай жоғалтады, ал дұрыс жауап ойынның алға жылжуына және ішкі валютаны (асықтарды) алуға мүмкіндік береді.

Ойын интерфейсінің негізгі элементтері

1. **Орыс тіліндегі сөз-кеңес:**
   Экранның жоғарғы бөлігінде орналасқан, қай сөзді құрастыру керегін түсінуге көмектеседі.
   Мысалы: «короткий» → қазақша сөз: «қысқа».

2. **Қазақ тіліндегі жартылай жасырылған сөз:**
   Жарықпен белгіленген әріптер бұрыннан белгілі символдарды көрсетеді, ал жетіспейтін орын underscore (астынғы сызық) («_») белгіленеді.
   Бұл сөзді “қалпына келтіру” эффектісін тудырып, ойыншының визуалды және тілдік есте сақтауын белсендіреді.

3. **Әріп жолдарын таңдау:**
   Жолда үш әріп нұсқасы пайда болады (мысалы, М, Қ, Ә). Ойыншы сөзге сәйкес әріпті таңдауы керек.
   Дұрыс таңдау ойыншының ұпай мен денсаулықты жоғалтпай қозғалысты жалғастыруына мүмкіндік береді.

4. **Сыйлықтар мен айыппұлдар жүйесі**

Қате жіберілсе жиналған тиындарды жоғалтады.

Экранда жетістік немесе қате визуалды эффектілер көрсетіледі (жарқыл, дыбыс, түсті жарықтандыру, жинау анимациясы).

Деңгейдің оқу ерекшеліктері

- Лексикалық есте сақтауды дамытады - ойыншы бұрын үйренген сөздерді еске алады.

- Қазақ әліпбиінің әріптерін дұрыс жазуды және танып-білуін жаттықтырады.

- Жаңа дыбыстар мен символдарды енгізеді, мысалы «қ», «ә», «ң», бұл қазақ тілінің ерекшеліктеріне бейімделуге маңызды.

- Марапаттар мен айыппұлдар жүйесі арқылы өзін-өзі бақылау және зейінділік дағдыларын қалыптастырады.

Ойыншы тек сөздерді тануды емес, олардың құрылымын қалпына келтіруді үйренеді, бұл есте сақтауды күшейтеді және ұзақ мерзімді меңгеруге ықпал етеді.

Өзара әрекет ету механикасы және басқару

- Басқару свайп арқылы (мобильді құрылғыларда) немесе пернетақтадағы жебелермен (ДК) жүзеге асады.

- Әріпті таңдау дұрыс жолға қозғалу арқылы орындалады.

- Кедергіге соғылғанда (мысалы, тас немесе жануар) ойыншы денсаулығының бір бөлігін жоғалтады.

- Сөз дұрыс жиналғаннан кейін жүйе автоматты түрде келесі тапсырмаға өтеді.

Визуалды және дыбыстық эффектілер

- Дұрыс таңдау кезінде - әріптің жұмсақ жарықпен ерекшеленуі және жағымды дыбыстық сигнал.

- Қате таңдау кезінде - ескерту дыбысы.

Деңгейдің педагогикалық мәні

Бұл деңгей ойын динамикасын тілдік дағдыларды белсенді жаттығумен біріктіреді. Негізгі назар төмендегілерге аударылады:

- аналитикалық ойлауды дамыту (ойыншы сөзге логикалық сәйкес әріпті анықтауы керек),
- шешім қабылдауда тәуелсіздік,
- қайталауға ынталандыру - нәтиже жақсарту және қателіктерден аулақ болу,

Бұл тәсіл оқу процесінің тиімділігін арттырады: ойын және әрекет арқылы, механикалық жаттау емес.

Деңгейді іске асыру нәтижелері

- Дұрыс әріпті таңдау және жауаптарды тексеру логикасы тұрақты жүзеге асырылды.
- Визуалды және техникалық тұрғыдан деңгей жоба көркемдік концепциясына сәйкес жасалды.
- Өнімділік оңтайландырылды: анимация бірқалыпты, FPS тұрақты.
- Ойыншылар оқыту мен ойын-сауық арасындағы балансқа оң баға берді.

Қорытынды

Екінші деңгей «Орташа» оқыту кезеңінің логикалық жалғасы болып табылады және ақпаратты пассивті қабылдаудан материалмен белсенді өзара әрекетке өтуін көрсетеді. Ойыншыда қазақ тілінде жазу және оқу бойынша алғашқы тұрақты дағдылар қалыптасады, зейін мен визуалды есте сақтау дамиды, сондай-ақ пайдаланушыны үшінші деңгейге дайындайды, онда сөздерді кеңестерсіз толық жинау қажет болады.

3-деңгей сипаттамасы - «Қиын»

Steppe Rider үшінші деңгейі ең күрделі және талаптары алғашқы деңгейлерден ерекше келеді. Осы деңгейде ойыншы қазақ сөздерін тек орысша аудармаға сүйене отырып, толықтай өзі құрауы керек, ешқандай кеңестер немесе жартылай ашылған әріптер жоқ.

Бұл деңгей механиканы толық меңгеруді және ойыншының тек есте сақтау, зейін және қазақ тілін білу арқылы сөздерді қалпына келтіре алу қабілетін көрсетеді.

Деңгейдің жалпы принципі

Экранда тек орысша сөз көрсетіледі, мысалы: «у тебя». Оның астында бос ұяшықтар тізбегі орналасқан - қазақша сәйкес сөздегі әріптер санына тең («сенде»). Барлық ұяшықтар жабық, кеңестер жоқ.

Ойыншының мақсаты - дұрыс әріптерді таңдай отырып, сөзді толықтай өзі құрастыру. Әр әріп үш нұсқадан біреуін таңдау арқылы алдыңғы жолдарда беріледі.

Ойын интерфейсінің негізгі элементтері

1. Орысша сөз-кеңес
Жоғарғы бөлігінде орысша сөз тіркесі көрсетіледі, ол болашақ сөздің мағынасын түсіндіреді. Бұл дұрыс жауапты құрастырудағы жалғыз кеңес болып табылады.

2. Қазақша сөздің бос ұяшықтары
Кеңестің астында әр әріпке сәйкес келетін бос квадраттар орналасқан. Ойыншы сөздің ұзындығын көреді, бірақ оның құрылымын білмейді.

3. Жолдағы әріптер
Кейіпкер алдында үш әріп нұсқасы пайда болады: мысалы, Й, Ұ, С
Олардың ішінде тек біреуі сөзді жалғастыруға жарамды.

4. Кедергілер және бонустар

- Жолда тастар, жануарлар және денсаулықты азайтатын басқа объектілер кездесуі мүмкін.
- «Алтын ×2» сияқты эффектілер пайда болып, ойынды ұқыпты ойнауға ынталандырады.

Өзара әрекет ету механикасы және ережелер

- Дұрыс таңдау әріпті жолға қосады, дыбыстық сигнал және жұмсақ жарықпен сүйемелденеді.

- Сөз толық құрылғаннан кейін қазақша сөздің дыбыстамасы ойнатылады, бұл бекітуді жеңілдетеді.

Деңгейдің оқу бағыттылығы

Үшінші деңгей келесі аспектілерге назар аударады:

- Сөздерді өз бетінше қалпына келтіру
- Сөздік қорының бекітілуі
- Қазақ сөздерінің құрылымын түсіну
- Зейін мен концентрацияны дамыту
- Орфографиялық есте сақтау

Педагогикалық әсері

Деңгей соңғы тексеру функциясын атқарады:

- Сөздерді танудың тұрақты механизмі дамиды
- Қазақ тілінде оқу және сөз құрастыру дағдылары бекітіледі
- Күрделі мақсатқа жету арқылы мотивация артады

Бұл кезең материалдың қаншалықты мықтап меңгерілгенін және ойыншының жаңа тақырыптарға өтуге дайын екенін көрсетеді.

6. ДЕРЕКТЕРДІ ЖЕРГІЛІКТІ ЖӘНЕ БҰЛТТЫ САҚТАУ ШЕШІМДЕРІН ҚОЛДАУ

Steppe Rider жобасын әзірлеу барысында екі негізгі бағытты қамтитын кешенді деректер сақтау жүйесі іске асырылды:

1. **жергілікті деректер сақтау,** ойыншыға толық автономдық қамтамасыз етеді;

2. **серверлік сақтау,** техникалық қолдау, ресурстарды сақтау және жобаны болашақта масштабтау үшін арналған.

Бұл архитектура ойынның тұрақтылығын, деректердің қауіпсіздігін және болашақтағы жаңартулар мен модернизацияда икемділікті қамтамасыз етеді.

Жергілікті деректер сақтау (автономды офлайн режимі)

Steppe Rider ойыны толық автономды өнім ретінде жобаланған, интернетке қосылуды қажет етпейді. Бұл білім беру мекемелері, мектептер, балалар орталықтары үшін және тұрақты желіге қол жеткізе алмайтын пайдаланушылар үшін маңызды.

Жергілікті сақтау жұмыс принциптері

Ойын барлық пайдаланушы ақпаратын құрылғыда тікелей сақтайды. Бұл келесі мүмкіндіктерді береді:

- сыртқы серверлерге тәуелді болмай, ойын процесін үздіксіз жүргізу;
- сақталған деректерге тез қол жеткізу;
- ойынның кез келген жағдайда толық қолжетімділігі;
- желідегі кешігулер мен қателердің болмауы;
- трафикті үнемдеу және өнімділіктің тұрақтылығы.

Жергілікті сақталатын деректер

Пайдаланушы құрылғысында сақталады:

- деңгейлер бойынша прогресс (жеңіл, орташа, қиын);
- жиналған сөздік қоры (сөздер, әріптер, оқу статистикасы);
- ойын валютасы (асықтар) және XP;
- ашылған скиндер мен таңдалған кейіпкер элементтері;
- жетістіктер, рекордтар, сессия нәтижелері;
- тіл, дыбыс, басқару, қиындық параметрлері;
- соңғы ойын сессиясының жағдайы.

Техникалық іске асыру

Жергілікті сақтау үшін оңтайландырылған жүйе қолданылады:

- деректерді JSON файлдарында немесе PlayerPrefs арқылы сақтау;
- жүктеуді жылдамдату үшін жергілікті кэштеу;
- қосымша ішінде деректерді резервтеу;
- негізгі параметрлерді базалық шифрлау;
- әр маңызды әрекеттен кейін автосақтау механизмі.

Осының арқасында ойын жылдам, тұрақты жұмыс істейді және сыртқы факторларға тәуелді емес.

Жергілікті сақтау артықшылықтары

- Толық автономия - ойын интернетсіз іске қосылады және прогресті сақтайды.
- Тұрақтылық - сервердің жоқтығынан деректер жоғалмайды.
- Жоғары жылдамдық - дереу жүктеу және сақтау.

- Қолжетімділік - мектептер мен білім беру бағдарламаларында пайдалануға болады.

- Қауіпсіздік - деректер желіге жіберілмейді, тек пайдаланушыда қалады.

Бұлтты сақтау шешімі (жобаның техникалық инфрақұрылымы)

Steppe Rider ойыны ойын барысында интернет қолданбаса да, тұрақты әзірлеу, материалдарды сақтау және жобаны болашақта масштабтау үшін бөлек серверлік инфрақұрылым құрылды.

Бұл ішкі сервис, ойыншылар үшін емес, әзірлеу және әкімшілік командасы үшін арналған.

Арнайы серверді алу

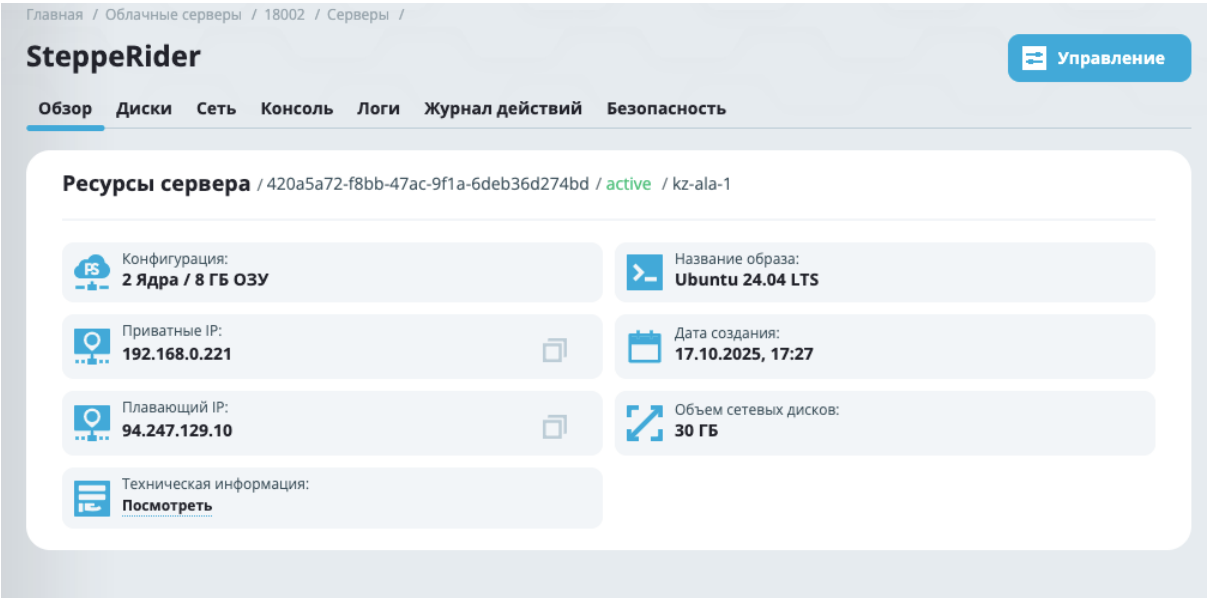
Жоба қажеттіліктері үшін арнайы сервер сатып алынып, іске қосылды, сипаттамалары:

- Процессор: 2 ядролық

- Жедел жады: 8 ГБ ОЗУ

- Операциялық жүйе: Ubuntu 24.04 LTS

Сервер қазіргі заманғы деректер орталығында орналастырылған, жоғары қолжетімділік пен деректерді қорғауды қамтамасыз етеді.



Сервердің тағайындалуы

Сервер жобаның техникалық қолдауына арналған және келесі мақсаттарда қолданылады:

Ресурстарды сақтау:

- ойынның бастапқы кодтары мен жұмыс нұсқаларын сақтау;
- сөздік базалары мен мәтіндік модульдерді сақтау;
- аудио дауыстаулар, суреттер, 3D модельдер және ассеттер;
- оқу материалдары мен құжаттамалар.

Командалық әзірлеуді қамтамасыз ету:

- әзірлеушілер үшін жобаның нұсқаларын сақтау;
- файлдар мен құжаттармен орталықтандырылған жұмыс;
- резервтік көшірмелерді жүргізу.

Жобаны болашақта кеңейту:

- әкімшілік панельді орналастыру (енгізілген жағдайда);
- ойын жаңартуларын және жаңа тақырыптық модульдерді дайындау;
- аналитика, логтау, рейтинг жүйелерін интеграциялау мүмкіндігі;
- масштабтау кезінде веб-сервистерді қолдау.

Маңызды: сервер ойын процесінде қолданылмайды және пайдаланушылардың жұмысына әсер етпейді - ойын кезінде 100% офлайн болып қалады.

Серверлік инфрақұрылымды қолданудың артықшылықтары

- Сенімділік - жобаның барлық ресурстары үшін біртұтас қорғалған сақтау орны.

- Икемділік - функционалды кеңейту мүмкіндігі.

- Қауіпсіздік - деректерді қорғау және резервтік көшірмелер.

- Әзірлеудің қолайлылығы - команданың орталықтандырылған жұмысы.

- Масштабтау дайындығы - сервер болашақта онлайн модульдер үшін қолданылуы мүмкін, егер олар қосылса.

Қорытынды

Steppe Rider жобасында заманауи, сенімді және тұрақты деректер сақтау архитектурасы іске асырылды:

- **Жергілікті сақтау -** ойынның толық автономиясын және прогресті интернетсіз сақтауды қамтамасыз етеді.

- **Серверлік шешім -** команданың қауіпсіз жұмысын, ресурстарды сақтауды және жобаны болашақта дамытуға негізді қамтамасыз етеді.

Бұл тәсіл бір уақытта білім беру процесінің талаптарын (офлайн қолжетімділік) қанағаттандырады және заманауи цифрлық өнімдер деңгейінде кәсіби әзірлеу мен қолдауды қамтамасыз етеді.

7. ОЙЫНДАР МЕН ҚОСЫМШАЛАРДЫ ОРНАЛАСТЫРУ ПЛАТФОРМАЛАРЫНДА АККАУНТТАР АШУ

Steppe Rider оқу ойынын әртүрлі операциялық жүйелерде және ойын платформаларында тарату үшін негізгі цифрлық дүкендерде ресми әзірлеуші аккаунттары жасалып, тіркеліп, толық бапталды: Google Play, App Store және Steam.

Бұл кезең жобаны жарыққа шығару алдындағы техникалық дайындықтың негізгі бөлігі болып табылады, өйткені әзірлеуші аккаунттары болмаса, өнімді орналастыру, жаңартуларды басқару, аналитика жүргізу, есептер жасау және ойынның толық өмірлік циклінде қолдау көрсету мүмкін емес.

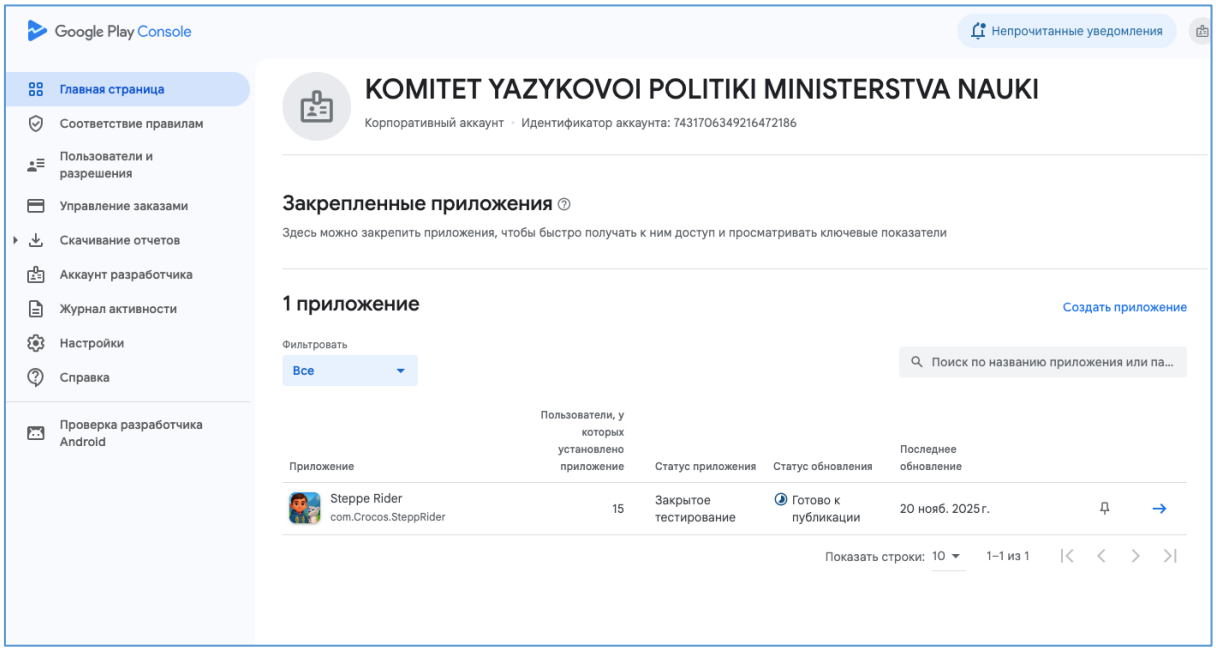
Google Play Developer Console аккаунтын жасау және баптау

Android - ең кең таралған платформалардың бірі болғандықтан, ойынды Google Play-ге орналастыру өнімді кең аудиторияға қолжетімді ету үшін маңызды қадам болып табылады.

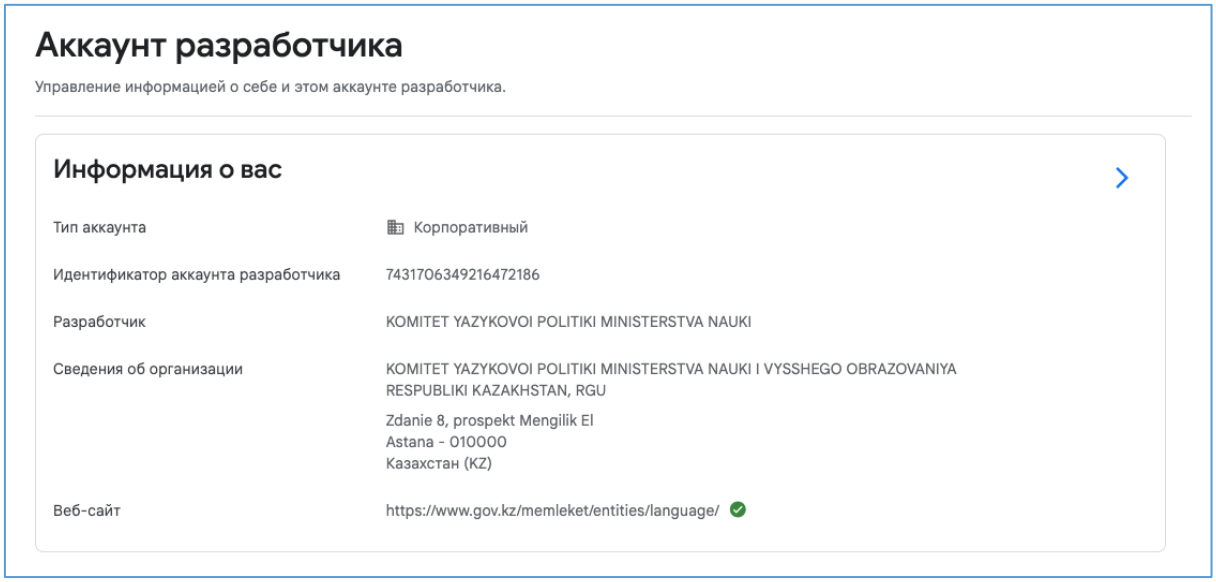
Аккаунтты тіркеу

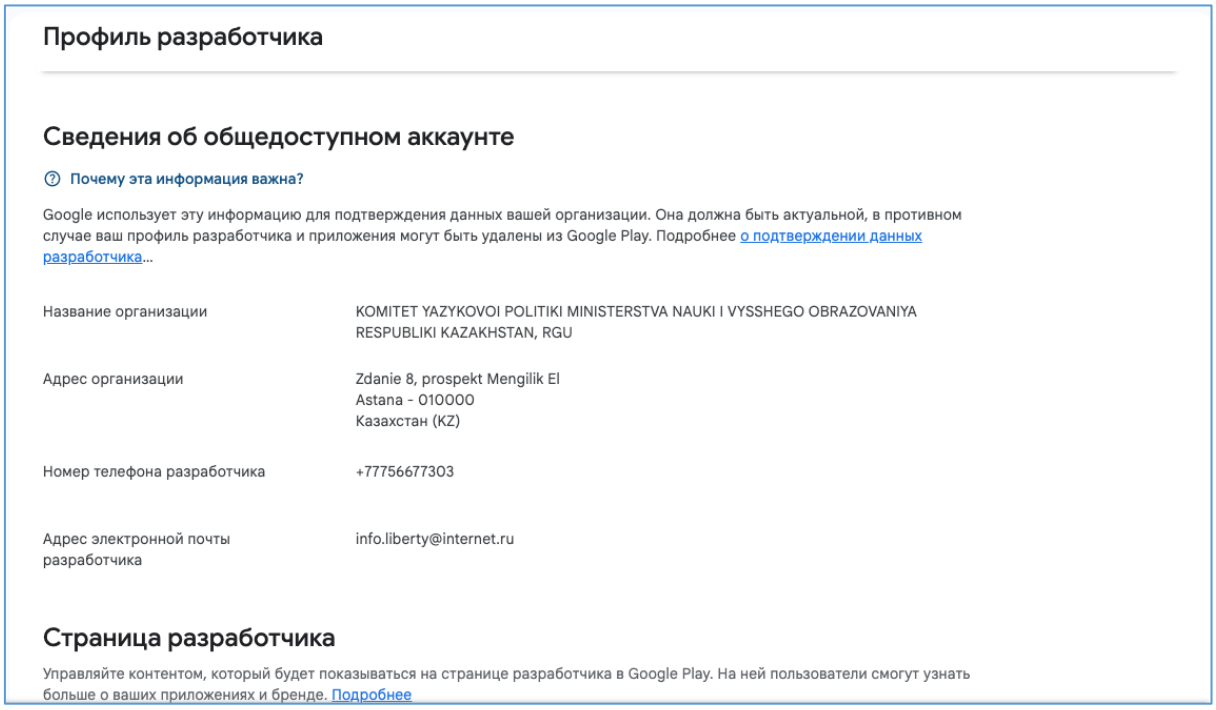
Дайындық аясында келесі шаралар жүргізілді:

- Steppe Rider жобасына арналған тек осы мақсатқа арналған Google аккаунты жасалды;



- Әзірлеуші статусын активация жасау үшін қажет бір реттік тіркеу жарнасы төленді;
- Құқықтық деректер толтырылды: ұйым атауы, тіркеу мекенжайы, байланыс ақпараттары;





Профиль разработчика

Сведения об общедоступном аккаунте

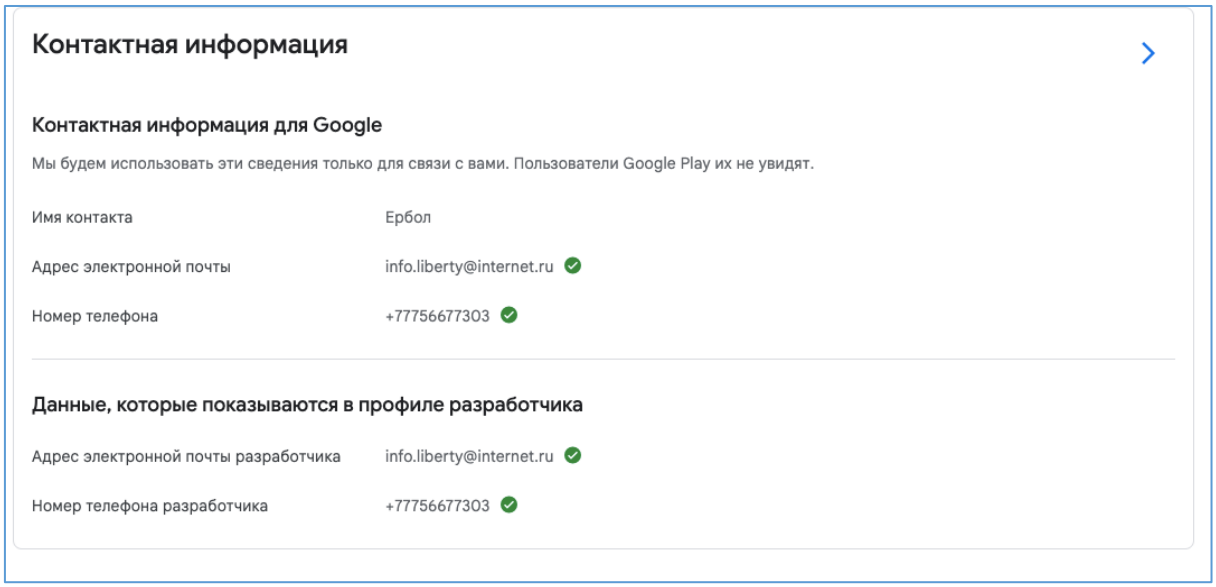
Почему эта информация важна?

Google использует эту информацию для подтверждения данных вашей организации. Она должна быть актуальной, в противном случае ваш профиль разработчика и приложения могут быть удалены из Google Play. Подробнее о подтверждении данных разработчика...

Название организации	KOMITET YAZYKOVOI POLITIKI MINISTERSTVA NAUKI I VYSSHEGO OBRAZOVANIYA RESPUBLIKI KAZAKHSTAN, RGU
Адрес организации	Zdanie 8, prospekt Mengilik El Astana - 010000 Казахстан (KZ)
Номер телефона разработчика	+77756677303
Адрес электронной почты разработчика	info.liberty@internet.ru

Страница разработчика

Управляйте контентом, который будет показываться на странице разработчика в Google Play. На ней пользователи смогут узнать больше о ваших приложениях и бренде. Подробнее



Контактная информация

Контактная информация для Google

Мы будем использовать эти сведения только для связи с вами. Пользователи Google Play их не увидят.

Имя контакта	Ербол
Адрес электронной почты	info.liberty@internet.ru
Номер телефона	+77756677303

Данные, которые показываются в профиле разработчика

Адрес электронной почты разработчика	info.liberty@internet.ru
Номер телефона разработчика	+77756677303

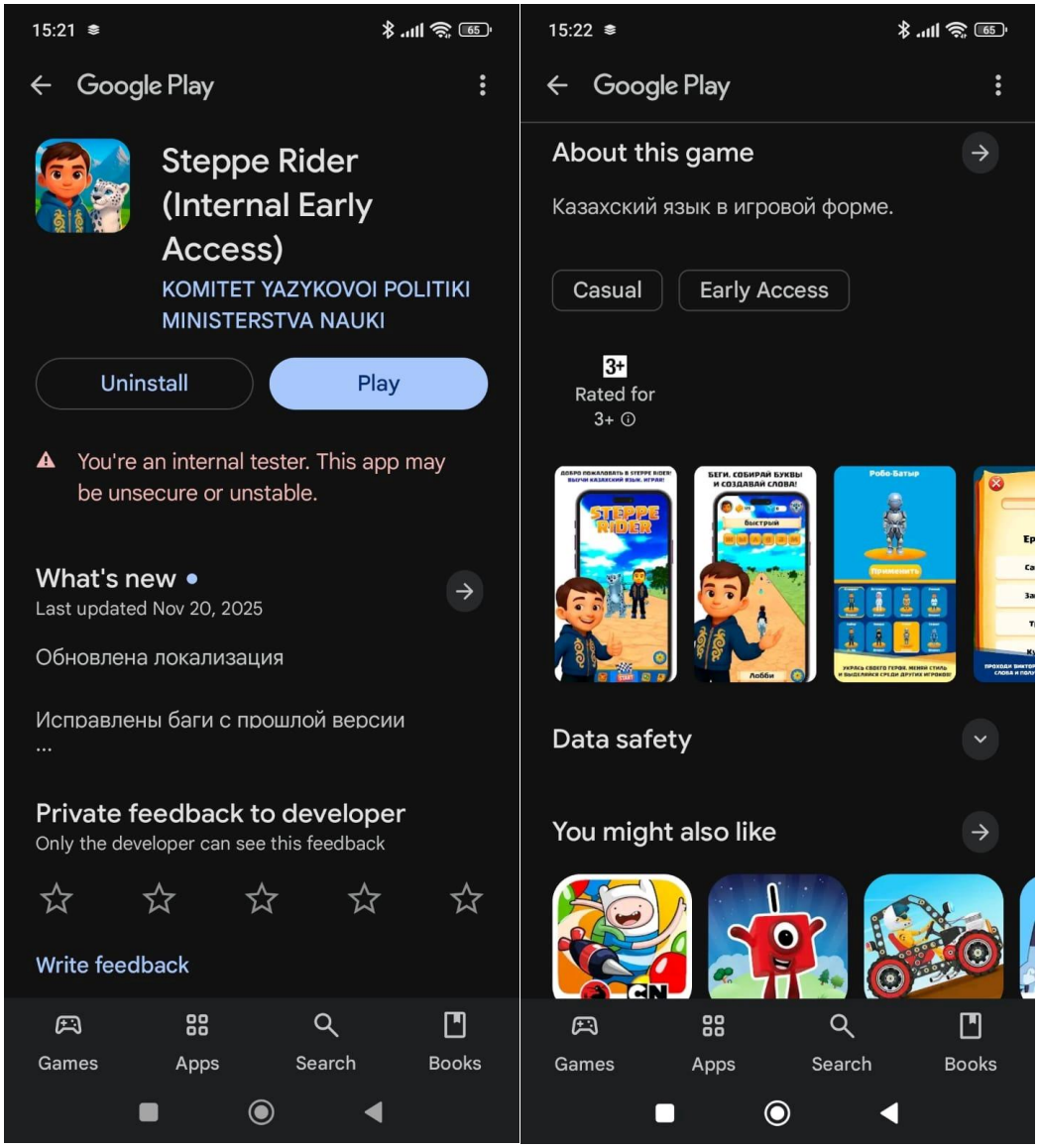
- аккаунт қауіпсіздігін қорғау құралдары (2FA, резервті кодтары) белсендірілді;
- пайдаланушыларға көрсетілетін әзірлеуші профилі бапталды.

Жариялау ортасын баптау

Аккаунт белсендірілгеннен кейін келесі техникалық дайындық жұмыстары жүргізілді:

- Google Play Console қолданбасына арналған Steppe Rider қосымшасының картасы құрылды;

- дайындалған ресурстар жүктелді: иконка, логотип, мұқаба, баннер және қосымша сипаттамасы;



- бөлімдердің құрылымы қалыптастырылды (Сипаттама, Құпиялылық саясаты, Байланыс ақпараты);

- қауіпсіздік параметрлері бапталды - Data Safety Form, рұқсаттар, талаптар;

- жабық тестілеу арналары іске қосылды (Internal testing / Closed testing);

- Android Vitals есептері бапталды (сыни қателер, қатып қалулар, FPS);

Аккаунттың маңызы

Бұл аккаунт мүмкіндік береді:

- ойынның финалдық APK/AAB жинағын жүктеуге;
- құрылымдық тексерістерден қайта өтпей-ақ жаңартулар мен түзетулер шығаруға;
- ойынды Google Play Market-ке жариялауға;
- жабық, ашық, альфа/бета тестілеуді жүргізуге;
- орнатулар, қателер, пікірлер статистикасын талдауға;
- ойының әртүрлі құрылғылардағы өнімділігін бақылауға.

Осылайша, жасалған аккаунт Steppe Rider ойынының Android-нұсқасын релизге толықтай дайындайды.

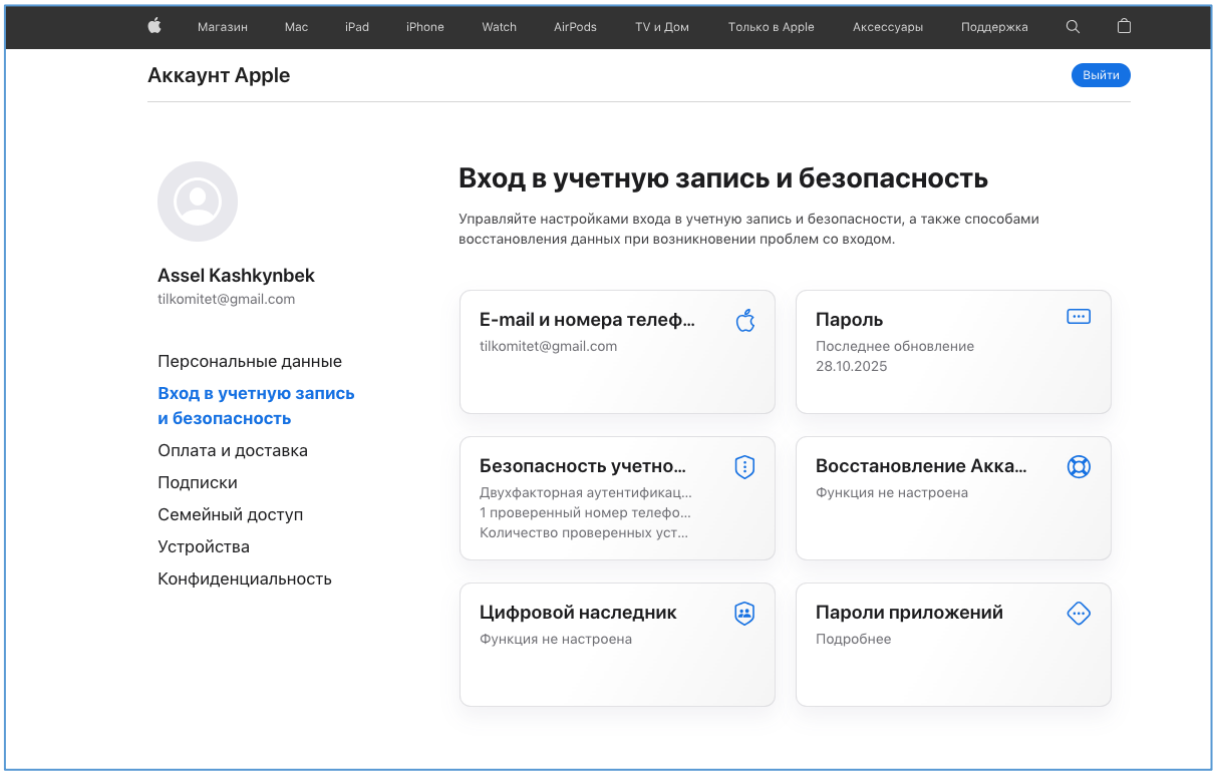
Apple Developer Account - App Store-ға жариялауға дайындық

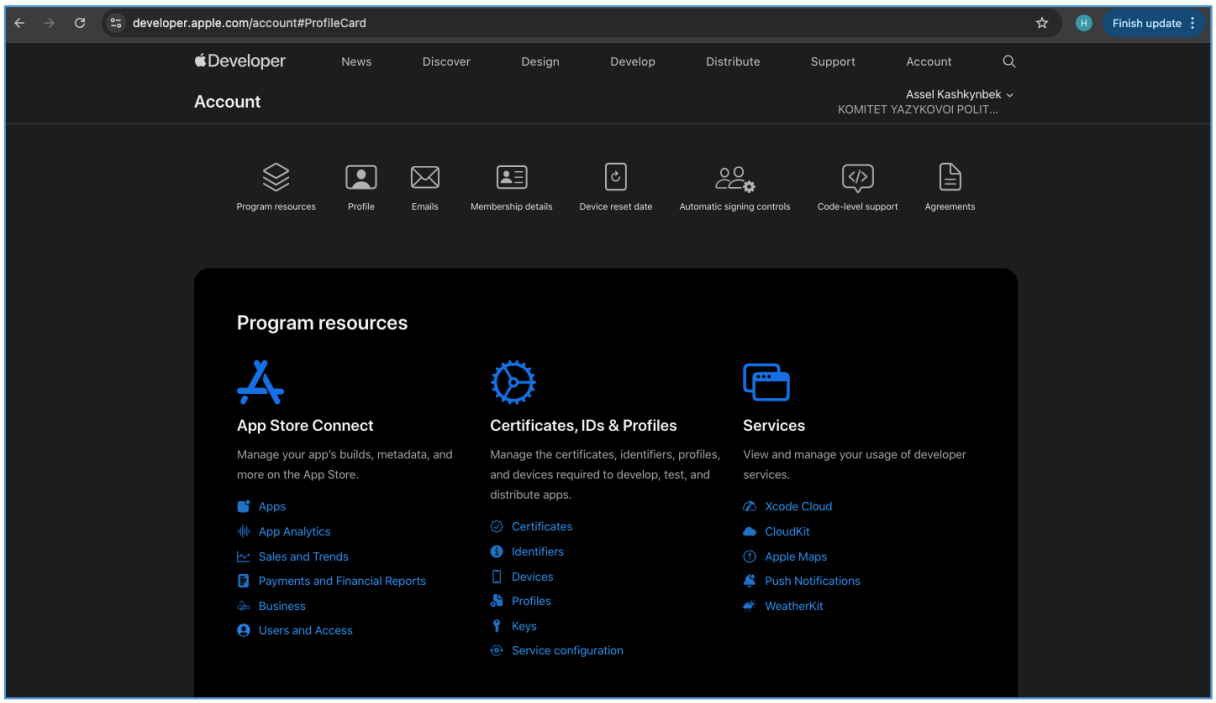
Apple App Store-ға ойын орналастыру экожүйенің қатаң талаптарына байланысты қосымша дайындықты қажет етеді.

Apple Developer Program тіркеуі

Төмендегі әрекеттер орындалды:

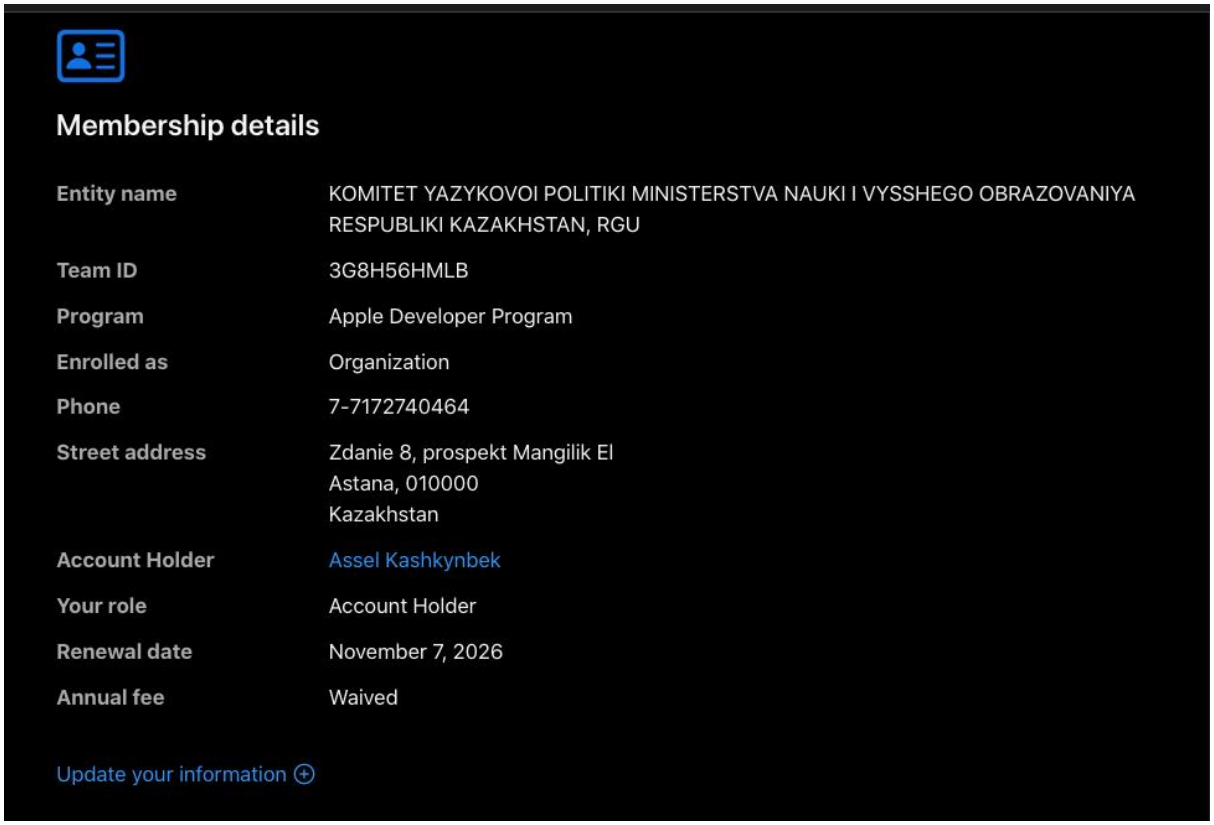
- жоба үшін жеке Apple ID құрылды;





- Apple Developer Program жазылымы рәсімделді, құны - жылына $99 USD;

- Құқықтық құжаттар толтырылды : ұйым/жеке тұлға туралы мәліметтер, мекенжай, байланыс деректері;



- төлем әдістері мен банктік ақпарат қосылды;
- екі факторлы аутентификация (2FA) іске қосылды;
- қол қою сертификаттарын беру расталды.

App Store Connect баптауы

App Store Connect - бұл Apple қосымшаларын басқару панелі.

Орындалған жұмыстар:

- Steppe Rider ойынының iOS нұсқасына арналған қосымша карточкасы жасалды;
- скриншоттар, иконкалар, сипаттамалар, санаттар дайындалды;
- қажетті сертификаттар жүктелді:
    - Distribution Certificate
    - Provisioning Profiles
    - Bundle Identifiers

- ішкі және сыртқы тестілеу үшін TestFlight бапталды;
- келесі бөлімдер толтырылды: Privacy Policy, Data Handling, Content Rights;
- Apple талап ететін жас санаты мен қауіпсіздік талаптары орындалды.

Аккаунттың маңызы

Apple Developer Account мүмкіндік береді:

- Steppe Rider ойынының iOS жинақтарын (IPA) жүктеуге және жариялауға;
- TestFlight арқылы тестілеу жүргізуге;
- жаңартуларды шектеусіз шығаруға;
- құрылғылар мен өнімділік бойынша аналитика алуға;
- App Store Review Guidelines талаптарына сай болуға.

Аккаунт Steppe Rider–дің iOS нұсқасын жариялауға толық дайын.

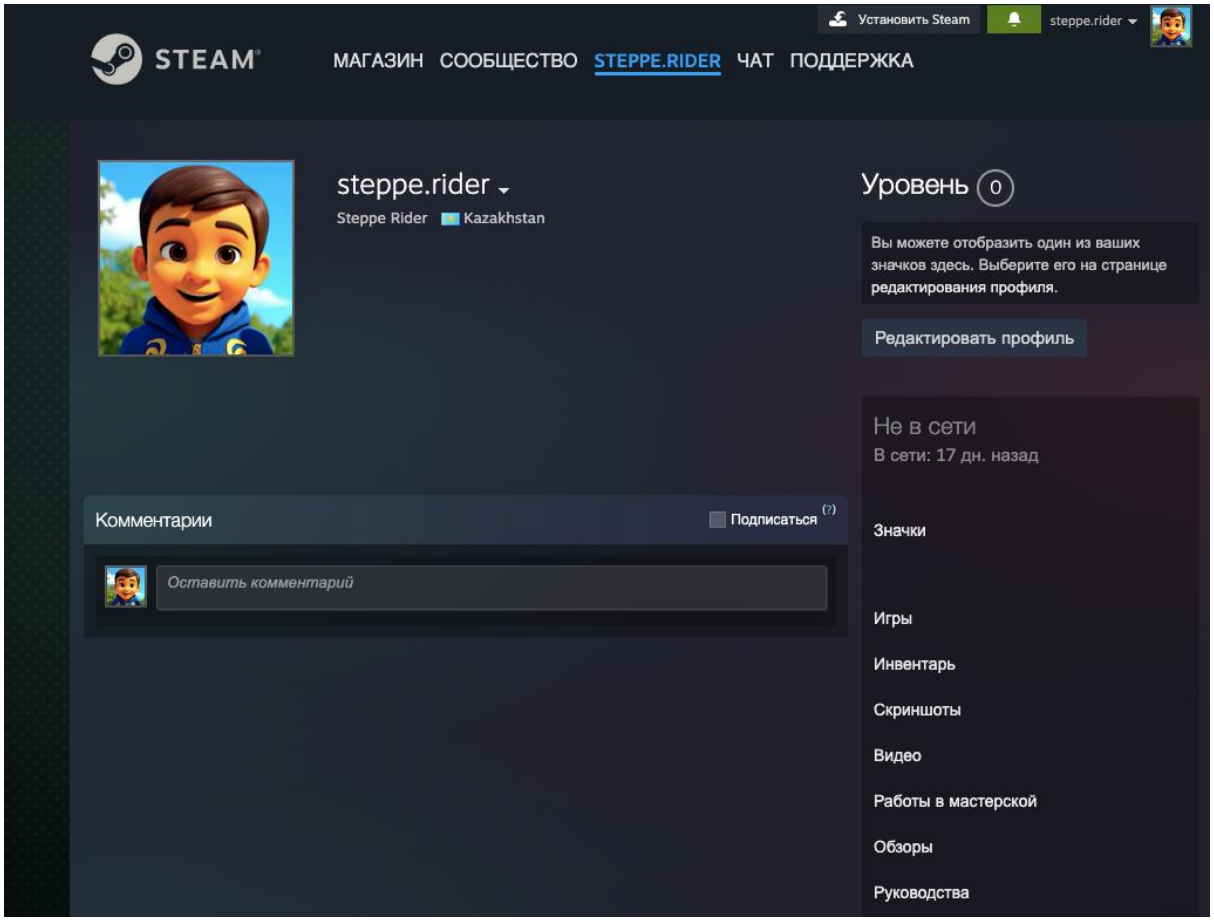
Steamworks (Steam) аккаунтын тіркеу және баптау

Steam - әлемдегі ең ірі PC-ойындар платформасы.
Steppe Rider ойынын компьютерге шығару үшін толыққанды Steamworks әзірлеуші аккаунты тіркелді.

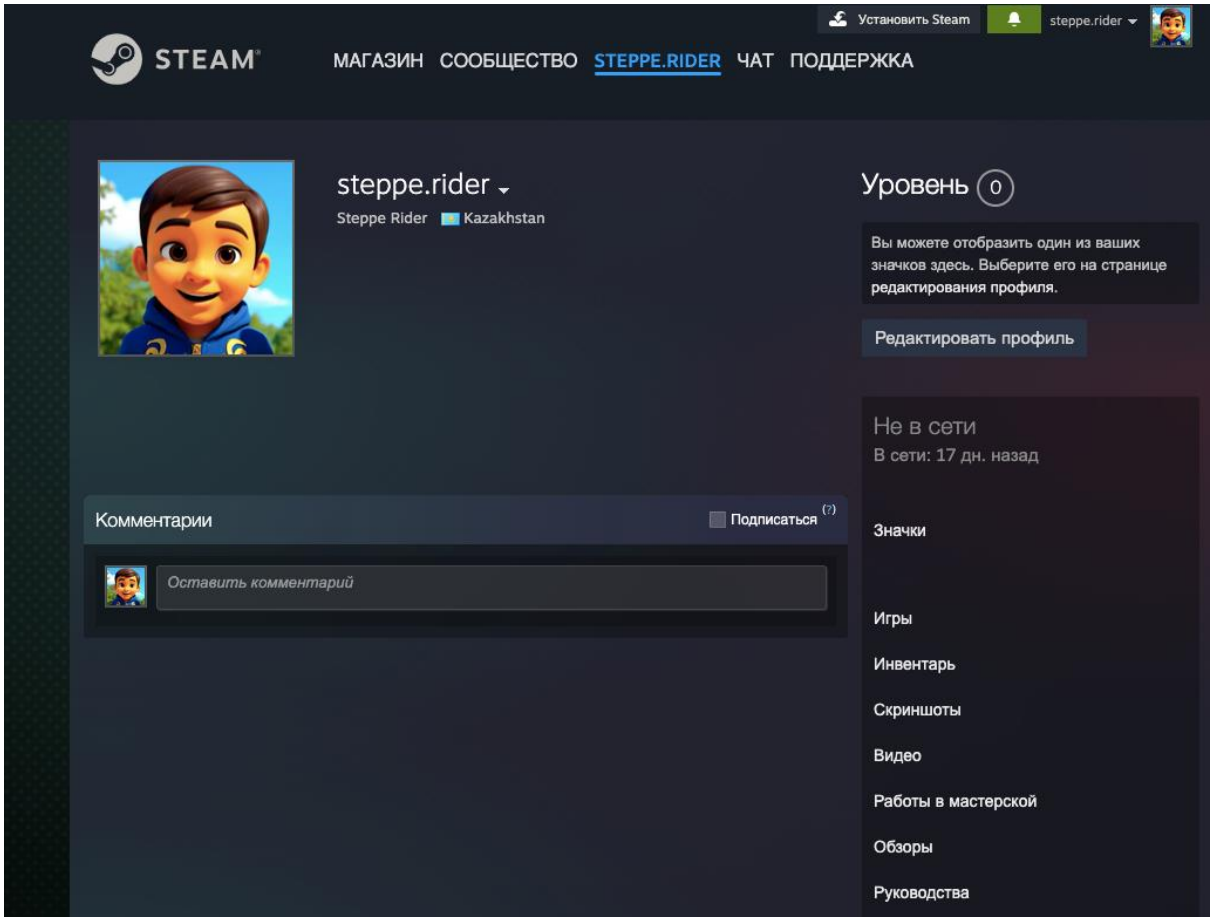
Тіркеу және заңдық баптау

Орындалған әрекеттер:

- корпоративтік Steam аккаунты құрылды;



- ойын үшін міндетті тіркеу төлемі ретінде $100 USD ақысы төленді;
- заңды деректер мен банк реквизиттері толтырылды;
- салықтық тексерістерден өтілді (W-8BEN / W-8BEN-E);
- Steamworks басқару панелі іске қосылды.



Steppe Rider жобасын Steamworks жүйесіне дайындау

Аккаунт белсендірілгеннен кейін келесі жұмыстар орындалды:

- Steamworks ішінде Steppe Rider жобасы құрылды;
- негізгі материалдар қосылды: атауы, сипаттамасы, жанры, жас санаты;
- келесі материалдарға арналған бөлімдер дайындалды:
    - трейлерлер,
    - скриншоттар,
    - баннерлер,
    - ассеттер,
    - жетістіктер (Achievements),
    - орналасуы;

- құрастырылымдарды сақтау жүйесі (Depot & Build System) бапталды;
- аналитика мен тестілеуге арналған Steam құралдары іске қосылды.

Аккаунттың маңызы

Steamworks келесілерді орындауға мүмкіндік береді:

- Steppe Rider жобасын Steam Store дүкенінде жариялау;
- жаңартуларды шексіз жүктеу және басқару;
- ойыншылар жөніндегі аналитиканы, ойын уақыты мен пайдаланушылардың кетуін бақылау;
- жетістіктерді, статистиканы, бұлттық сақтауларды (қажет болған жағдайда) қосу;
- ойынды мектептер мен білім беру ұйымдарына ДК нұсқасында тарату.

Аккаунт сәтті жасалды және толық жұмысқа дайын.

Қорытынды

Steppe Rider жобасын жариялауға дайындау барысында барлық негізгі платформаларда ресми әзірлеуші инфрақұрылымы толық ұйымдастырылды:

Google Play Developer Account
— тіркелген, төленген, бапталған, Android нұсқасын жүктеуге толық дайын.

Apple Developer Program
— ресімделген, белсендірілген, App Store Connect толық дайындалған.

Steamworks Developer Account
— тіркелген, Steppe Rider жобасы бапталған, PC нұсқасын жариялауға дайын.

Осы шаралар Steppe Rider жобасының барлық негізгі платформаларда релизге толық дайын болуын, әрі қарай қолдауды, жаңартулар шығаруды және аудиторияны кеңейтуді қамтамасыз етеді.

8. STEAM ТАЛАПТАРЫНА СӘЙКЕС ОЙЫН ПЛАТФОРМАСЫН ӘЗІРЛЕУ

Steppe Rider ойынын әлемдегі ең ірі сандық тарату платформаларының бірі - Steam-де сәтті орналастыру және тарату үшін барлық техникалық талаптар мен Steamworks ұсыныстарын ескере отырып, платформаны әзірлеу және баптау жұмыстары жүргізілді. Steamworks - бұл Steam платформасы арқылы ойындар мен қосымшаларды олардың жүйесімен интеграциялау үшін берілетін құралдар мен API жиынтығы.

Бұл процесс тек ойынды дайындаумен шектелмей, Steam-де толық жұмыс істеу үшін барлық қажетті сервистерді баптауды қамтиды. Бұл ойын шығаруға, статистиканы бақылауға, жаңартуларды енгізуге, жетістіктерді, бұлттық сақтау (Cloud Save) және басқа маңызды функцияларды интеграциялауға мүмкіндік береді.

Steamworks Developer аккаунтын тіркеу

Steppe Rider ойынын Steam платформасында орналастыру үшін Steamworks-те әзірлеуші аккаунты құрылды. Бұл аккаунт қажет, себебі:

- Steam Store-де ойынды орналастыру;
- Ойын билдтерін жүктеу және оларды жаңарту;
- Steam құралдарын пайдалану (Achievements, Trading Cards, Cloud Save және т.б.).

Тіркеу процесі келесі қадамдарды қамтыды:

- Steamworks Developer Portal-де ресми аккаунт құру;
- Әзірлеуші туралы ақпарат беру: заңды атауы, байланыс деректері, төлем алу үшін банктік ақпарат;
- Ойынды платформаға орналастыру үшін $100 USD бір реттік төлемді төлеу.

Steppe Rider жобасын Steamworks-те дайындау

Аккаунт сәтті тіркеліп, белсендірілгеннен кейін Steppe Rider жобасы дайындалды, оның негізгі кезеңдері:

Steam Store-де ойынның бетін құру
Ойынды сәтті жариялау үшін Steam Store Page құрылды:

- Жобаның негізгі деректерін толтыру (атауы, сипаттамасы, жанры);
- Ойын иконкасын, баннерін және скриншоттарын қосу;
- Ойын функционалының сипаттамасы, жас рейтингі, жүйелік талаптар (PC нұсқасы үшін);
- Пайдаланушылардың назарын аударатын видеоматериалдарды (трейлерлер) дайындау.

Қауіпсіздік және алаяқтықтан қорғау жүйесі
Читтер мен модификацияларды қолдануды болдырмау үшін Anticheat модулі интеграцияланды, бұл Steam-де жариялау үшін міндетті.

Steam талаптарына сәйкестікті тексеру
Ойынды Steam-ге финалдық нұсқасын жүктемес бұрын, Steam Technical Requirements талаптарына сәйкестігі мұқият тексерілді:

- Steam интерфейсінің жұмысын тексеру, Steam Overlay (чаттар және басқа функциялар үшін қалқымалы терезе) қолдауын қоса;
- SteamOS талаптарына сәйкестігін тексеру (Linux үшін операциялық жүйе);
- Ойын локализациясы мен әртүрлі аймақтарға сәйкес контентін тексеру;
- Steam’s Content Delivery System жүйесімен интеграциясы, жаңартулар мен контентті дұрыс тарату үшін.

Steam платформасында ойын жариялау және шығару
Барлық баптаулар аяқталғаннан кейін және тестілеу талаптарға сәйкестігін растаған соң, Steppe Rider ойыны Early Access-де Steam платформасында орналастырылды:

- Ойын Steam Store-де жүктеп алу мүмкіндігімен жарияланды;
- Жаңартулар, патчтар және жақсартулар тұрақты түрде шығатын етіп бапталды;

- Steam-дегі ойын беті ашық жарияланды, бұл жобаның назарын аударуға және пайдаланушы пікірлерін жинауға мүмкіндік берді.

Steam-де орналастырудың артықшылықтары

- **Кең аудитория** - бүкіл әлем бойынша миллиондаған ойыншыларға, оның ішінде білім беру ойындарына қызығушылық танытатындарға қолжетімділік;

- **Маркетинг және насихаттау механизмдері** - промо-акциялар, жеңілдіктер, сондай-ақ Steam Curators арқылы жобаны насихаттау мүмкіндігі;

- **Steam экожүйесімен интеграция** - Trading Cards, Cloud Save, жетістіктер, синхронизация және басқа функцияларды қолдау;

- **Кері байланыс және қолдау** - пайдаланушы пікірлерін жинау және ойын сапасын жақсарту үшін жылдам әрекет ету мүмкіндігі.

Қорытынды

Steppe Rider ойын платформасын Steam талаптарына сәйкес әзірлеу және баптау PC үшін ең ірі ойын платформасында ойынды сәтті орналастыруға мүмкіндік берді. Бұл қамтамасыз етеді:

- Әлем бойынша пайдаланушыларға ойын қолжетімділігі;

- Жаңарту және қолдау жүйесінің тиімділігін іске асыру;

- Achievements, Steam Cloud, Trading Cards сияқты қосымша функциялармен интеграция, бұл ойыншылардың қызығушылығы мен мотивациясын айтарлықтай арттырады.

Steam ойын платформасы ойынды тарату және нарықта ұзақ мерзімді табысқа жету стратегиясының ажырамас бөлігі болып табылады.

9. ДАЙЫН ОЙЫННЫҢ ТАНЫСТЫРЫЛЫМЫН ӨТКІЗУ

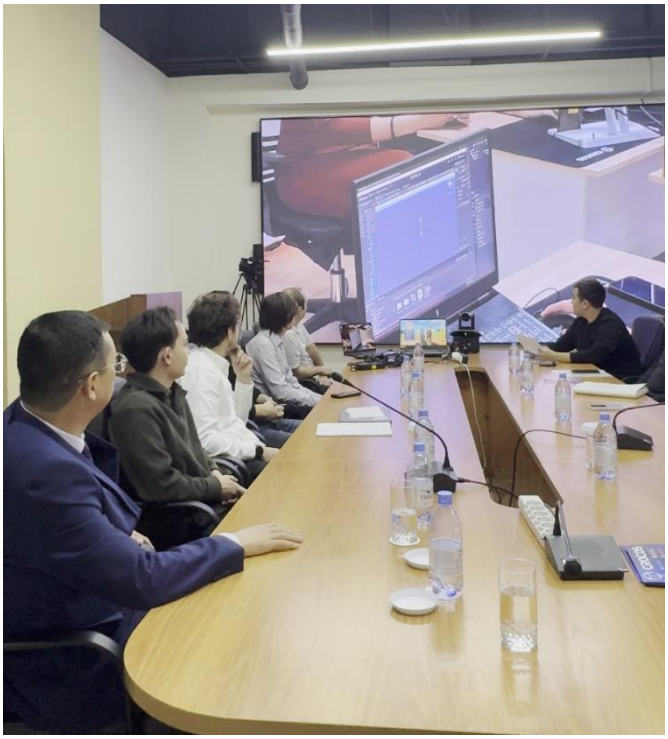
2025 жылдың 01 желтоқсанында Тапсырыс берушіге дайын ойынның таныстырылымы ұйымдастырылды. Шараға Тіл саясаты комитеті төрағасы мен қызметкерлері, жобаны жүзеге асыруға қатысқан бағдарламашылар, әдістемелік топтың мүшелері, дизайнерлер, тестілеушілер және басқа да қызығушылық танытқандар шақырылды.

Шара барысында жоба жетекшілері Steppe Rider ойынының мақсаттары, оның құрылымы және әзірлеу кезеңдеріне тоқталды. Ойынның ДК және мобильді нұсқалары демонстрациялық түрде көрсетіліп, функционалдық мүмкіндіктері мен тіл үйрету механикасы жан-жақты таныстырылды.

Сондай-ақ, команданың жобаны жасау барысын көрсететін бейнематериал ұсынылып, әдістемелік негіздер мен білім беру мазмұны туралы толық ақпарат берілді. Таныстырылымда Астана қаласындағы №104 мектепте жүргізілген бета-тестілеу нәтижелері де жарияланды. Оқушылардың пікірлері ойынның түсінікті, қызықты және тілді меңгеруге тиімді екенін көрсетті.

Тапсырыс берушілер дайын өнімді жоғары бағалап, Steppe Rider жобасының мемлекеттік тіл саясатын қолдауда маңызы зор екенін атап өтті.

STEPPE RIDER
STEPPE RIDER
Балаларға қазақ тілін
үйретуге арналған ойын
Білімге толы қызықты



10. STEPPE RIDER ОЙЫНЫН ПЛАТФОРМАЛАРДА ЖАРИЯЛАУ (БЕТА-НҰСҚА)

Steppe Rider ойынының әзірлеу кезеңінің соңында бета-нұсқасы сәтті дайындалып, негізгі сандық платформаларда орналастырылды. Бұл нақты пайдаланушылардан кері байланыс жинап, тестілеу процесін бастауға мүмкіндік берді.

Жариялау платформалары
Кең аудиторияны қамту және конструктивті пікірлер алу үшін ойын келесі платформаларда жарияланды:

- Android құрылғылары үшін **Google Play Store**;
- iOS құрылғылары үшін **Apple App Store**;
- ДК нұсқасы үшін **Steam**.

Бета-тестілеудің міндеттері
Бета-тестілеудің негізгі мақсаттары:

- Ойынның әртүрлі құрылғылардағы тұрақтылығы мен өнімділігін тексеру;
- Негізгі ойын функционалды жұмысын тексеру: әріптерді жинау, сөз құру, кейіпкерді басқару;
- Интерфейс пен пайдаланушы тәжірибесінің ыңғайлылығын бағалау;
- Графика, дыбыстық дизайн, оқу процесі бойынша кері байланыс жинау.

Тестілеуге қатысқан ойыншылар жабық тест топтарына шақырылып, оларға кірістірілген есеп беру жүйесі және кері байланыс формалары арқылы тұрақты байланыс жүргізілді. Алғашқы кезеңде тестілеу команданың және әзірлеуші компанияның ішінде жүргізілді.

Бета-нұсқаның артықшылықтары
Ойынның бета-нұсқасын осы платформаларда жариялау осыларға мүмкіндік берді:

- Ойынның нақты жағдайда жұмысын толық бағалау;

- Механика мен интерфейстегі потенциалды қателер мен багтарды анықтау және түзету;

- Әртүрлі сипаттамалары бар құрылғыларда ойын өнімділігін бағалау (орта деңгейден премиум сегментке дейін);

- Ойын мәтінінің локализациясы мен бейімделу мәселелері туралы ақпарат жинау.

Ойынның мектептегі бета-тестілеуі

Steppe Rider ойынын релизге дайындау аясында мектепте бета-тест жүргізілді. Бұл нақты жағдайда тестілеуді өткізуге және мақсатты аудиториядан маңызды кері байланыс жинауға мүмкіндік берді.

Тестілеуді дайындау және ұйымдастыру

Тестілеу үшін оқушылар сыныбы таңдалды, онда ойын компьютерлерге орнатылды. Астана қаласындағы №104 орта мектептің 8-сынып оқушылары осы тест аясында ойынның алғашқы пайдаланушылары болды. Мектеп аудиторияның негізгі бөлігі болып табылатындықтан таңдалды, себебі Steppe Rider ойыны қазақ тілін ойын арқылы үйретуге бағытталған.

Ойынды орнату және жабдықтау дайындығы
Сәтті тестілеу үшін келесі қадамдар жасалды:

- Сыныптағы компьютерлерге Steppe Rider ойын нұсқасы орнатылды;

- Жабдық ойынның минималды жүйелік талаптарына сай дайындалды, бұл әртүрлі конфигурациядағы компьютерлерде ойынның тұрақтылығын қамтамасыз ету үшін қажет болды.

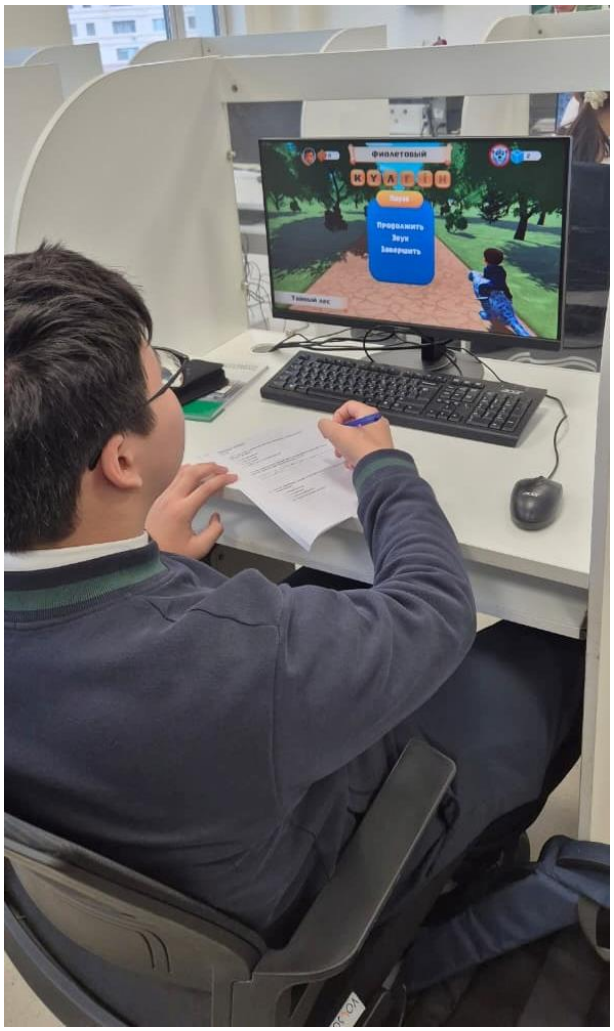
Тестілеу процесі
Әр оқушыға ойынды 30–40 минут ойнауға мүмкіндік берілді. Ойын барысында мектеп оқушылары:

- Ойынның негізгі механикаларымен танысты - әріптерді жинау, сөз құру, кедергілерден аулақ болу.

- Ойынның деңгейлерін (жеңіл, орташа және қиын) өтіп, интерфейстің ыңғайлылығы, тапсырмалардың күрделілігі және оқыту әсерін бағалады.

- Ойын нәтижелері туралы кері байланыс алды - қалқымалы терезелер арқылы (жиналған сөздер саны, асықтар, XP).

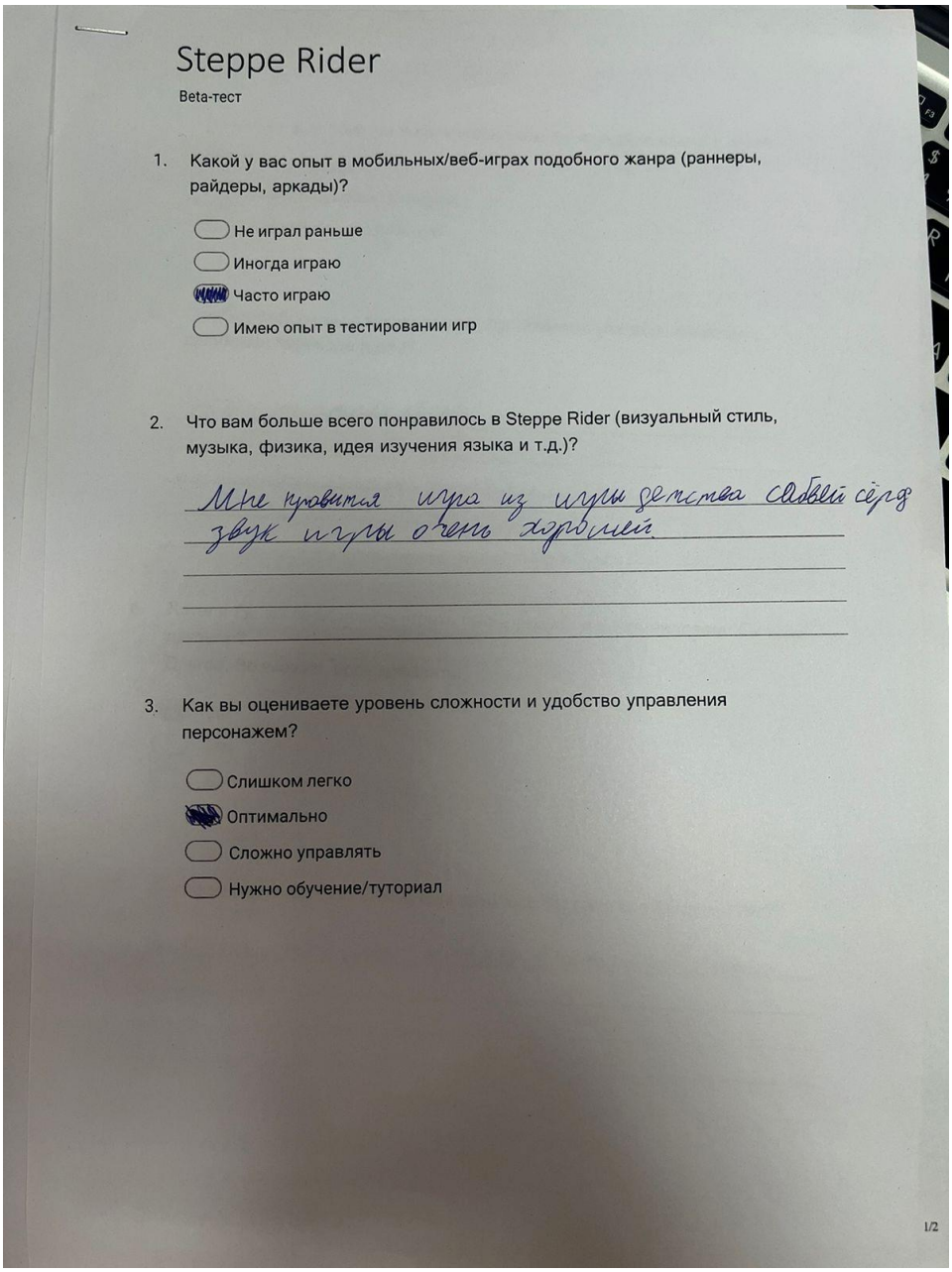
- Ойын барысында әртүрлі функционалдық элементтер тестіленді: күрделілік параметрлері, кейіпкерді басқару және қазақ тіліндегі аударманың дұрыстығы.



Пікірлерді жинау және бақылау

Тестілеу барысында қатысушылардан белсенді түрде кері байланыс жиналды:

- Тест соңында оқушыларға сауалнамалар ұсынылды, онда олар ойынды әртүрлі параметрлер бойынша бағалай алды: қызықтылығы, күрделілігі, басқарудың ыңғайлылығы.



Steppe Rider

Beta-тест

1. Какой у вас опыт в мобильных/веб-играх подобного жанра (раннеры, райдеры, аркады)?

   - Не играл раньше
   - Иногда играю
   - Часто играю
   - Имею опыт в тестировании игр

2. Что вам больше всего понравилось в Steppe Rider (визуальный стиль, музыка, физика, идея изучения языка и т.д.)?

   Мне нравится игра из игры degustaa сайын сёрф звук игры очень хорошей.

3. Как вы оцениваете уровень сложности и удобство управления персонажем?

   - Слишком легко
   - Оптимально
   - Сложно управлять
   - Нужно обучение/туториал

1/2

4. Насколько вам понятен и интересен элемент изучения языка в игре?

- [x] Очень полезно
- [ ] Нормально, можно улучшить
- [ ] Не понял, как это работает

5. Столкнулись ли вы с техническими проблемами (лагами, вылетами, ошибками перевода и др.)?

Нет не сталкевался

6. Хотели бы вы продолжить играть после релиза и рекомендовали бы игру друзьям?
Другое: Возможно, если добавить...

- [x] Да
- [ ] Нет
- [ ] Другое: _____

7. Какие функции/локации/темы вам хотелось бы видеть в Steppe Rider?

разные локаций кроме внеш него мира

2/2

Steppe Rider

Beta-тест

1. Какой у вас опыт в мобильных/веб-играх подобного жанра (раннеры, райдеры, аркады)?

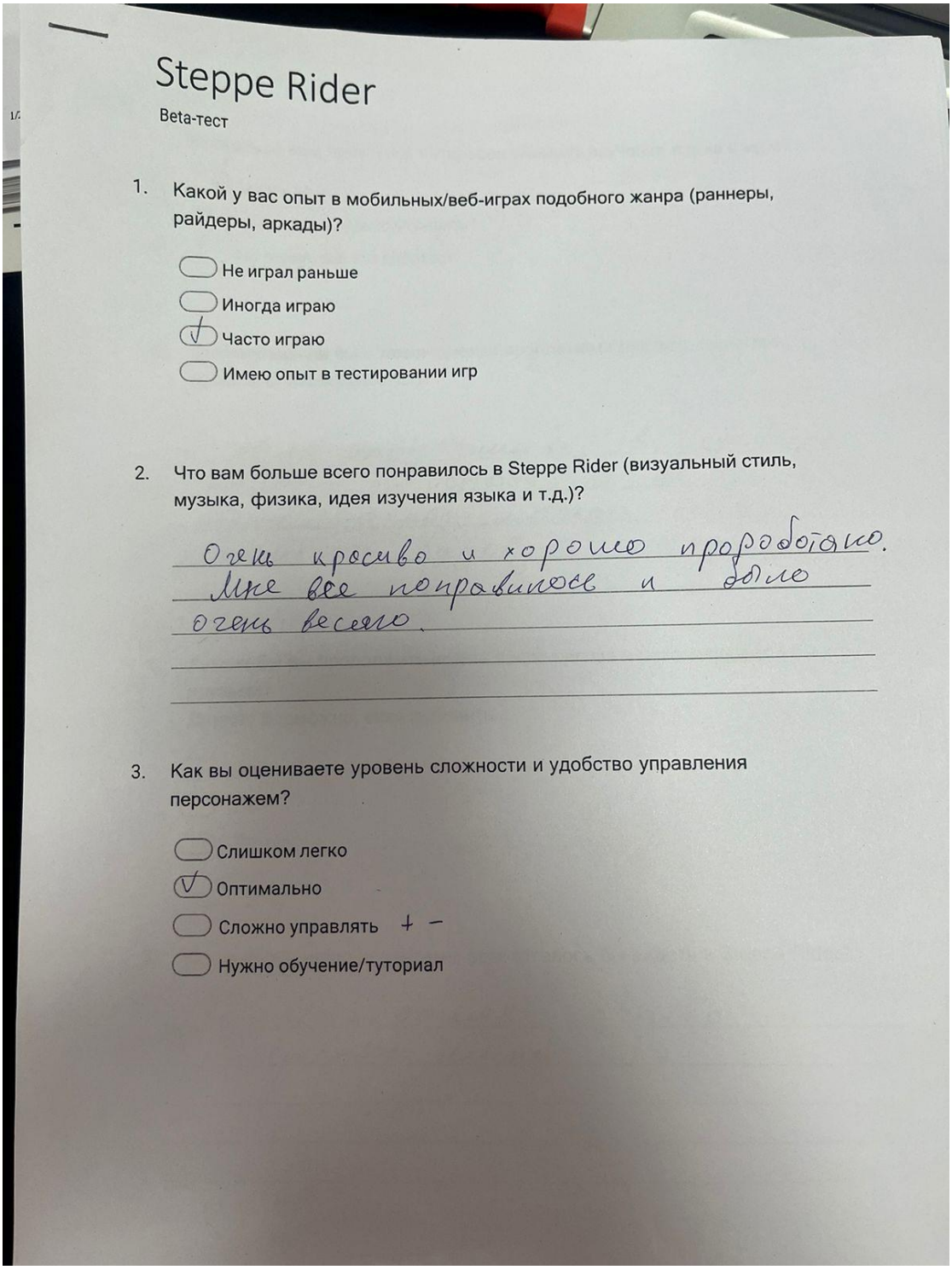
- [ ] Не играл раньше
- [ ] Иногда играю
- [x] Часто играю
- [ ] Имею опыт в тестировании игр

2. Что вам больше всего понравилось в Steppe Rider (визуальный стиль, музыка, физика, идея изучения языка и т.д.)?

Очень красиво и хорошо проработано. Мне все понравилось и было очень весело.

3. Как вы оцениваете уровень сложности и удобство управления персонажем?

- [ ] Слишком легко
- [x] Оптимально
- [ ] Сложно управлять + −
- [ ] Нужно обучение/туториал



Steppe Rider

Beta-тест

1. Какой у вас опыт в мобильных/веб-играх подобного жанра (раннеры, райдеры, аркады)?

- [ ] Не играл раньше
- [ ] Иногда играю
- [x] Часто играю
- [ ] Имею опыт в тестировании игр

2. Что вам больше всего понравилось в Steppe Rider (визуальный стиль, музыка, физика, идея изучения языка и т.д.)?

Очень красиво и хорошо проработано. Мне все понравилось и было очень весело.

3. Как вы оцениваете уровень сложности и удобство управления персонажем?

- [ ] Слишком легко
- [x] Оптимально
- [ ] Сложно управлять + –
- [ ] Нужно обучение/туториал

- Көбінесе оқушылар ойын процесіне жоғары қызығушылық танытып, ойынның механикасын тез меңгергені байқалды. Алайда, кейбір оқушылар қазақ тіліндегі сөздерді жинауда қиындықтарға тап болды, бұл оқыту экрандарын және нұсқауларды әрі қарай жетілдіру қажеттілігін растайды.

- Интерфейстегі аудармаларда кейбір қиындықтар байқалды: кейбір сөздер немесе тіркестер оқушылар үшін жеткілікті түрде түсінікті болмаған.

- Жеке жағдайларда оқушылар анимациялардың кейбір баяулауын хабарлады, бұл әлсіз конфигурациялы компьютерлер үшін графиканы оңтайландыру қажеттігін көрсетті.

Тестілеу аяқталғаннан кейін жүргізілген іс-шаралар:

- Барлық сауалнама деректері талдау үшін жинақталды.

- Пайдаланушылардың тілектері мен сыни пікірлері қаралды.

- Ойын процесіндегі қателіктер мен кемшіліктер назарға алынып, есепке енгізілді.

Мектепте тестілеудің артықшылықтары

Мектептік ортада бета-тестілеуді өткізу бірнеше негізгі артықшылықтарды берді:

- Ойын процесін нақты аудиторияда тексеру, бұл оқу мен түсіну кезінде туындайтын мүмкін қиындықтарды анықтауға мүмкіндік берді.

- Жасөспірімдерден кері байланыс алу, олар ойын тәжірибесін және интерфейсті жақсартуға құнды идеялар ұсына алады.

- Мектеп жабдықтарында ойынның жұмыс қабілетін бағалау мүмкіндігі.

- Оқушылардың қазақ тілін ойын арқылы оқуға ынтасы, бұл ойынның қызықты және тартымды екендігі туралы оң пікірлермен растады.

11. ҚОРЫТЫНДЫ ТЕКСЕРУЛЕР

Steppe Rider ойынын жасаудың соңғы кезеңінде ойын процесінің барлық жүйелерінің тұрақтылығын, сапасын және сәйкестігін қамтамасыз ету үшін соңғы тексерістер жүргізілді. Бұл тексерістер ойынның барлық аспектілерін қамтыды - функционалдық мүмкіндіктерден бастап, өнімділік, пайдаланушы интерфейсі және деректер қауіпсіздігіне дейін.

Соңғы тексерістердің мақсаты - ойын барлық қолдаулы құрылғыларда дұрыс жұмыс істейтінін, ал алдыңғы кезеңдерде анықталған қателер түзетілгенін және ойын функционалы қойылған талаптарға сай екенін растау болды.

Ойын функционалдығын тексеру
Ойын механикалары

- Барлық негізгі ойын механикалары (әріп жинау, сөз құрастыру, бонус пен кедергілерді пайдалану) дұрыс және ақаусыз жұмыс істейтіні тексерілді.

- Ойын прогресін сақтау логикасы тексерілді: ойыншылар тоқтаған жерден деректер жоғалмай жалғастыра алады.

- Жетістіктер жүйесі (Achievements) және ішкі ойын валютасы (асықтар) тексерілді - есептеу ойын әрекеттеріне сәйкес жүргізіледі.

- Ойыншылар деңгейлер арасында қателіктерсіз ауыса алатынына көз жеткізілді, ал алғашқы іске қосу кезінде оқыту экрандары дұрыс көрсетіледі.

Басқару және интерфейс

- Мобильді құрылғыларда (сенсорлық басқару) және ДК-де (пернетақта/тышқан) басқарудың ыңғайлылығы тексерілді.

- Анимациялар мен визуалды эффектілердің бірқалыпты өтуі тексерілді, кейбір элементтердің жылдам қозғалысы кезінде тоқтап қалмауы қамтамасыз етілді.

- Локализациялар тексерілді - барлық мәтіндер дұрыс аударылған және орыс және қазақ тілдерінде дұрыс көрсетіледі.

Өнімділікті тексеру
Әртүрлі құрылғылардағы өнімділік

- Ойынның түрлі құрылғылардағы өнімділігі тексерілді: әлсіз ДК және бюджеттік смартфондардан бастап, қуатты құрылғыларға дейін.

- Ойын экрандағы объектілер көп болғанда да, тұрақты FPS (60 кадр/секунд) көрсеткішін сақтауы қажет.

- Деңгейлердің және ойынның толық жүктелу уақыты көбінесе 5 секундтан аспайтыны тексерілді.

Тұрақтылық және оңтайландыру тексерісі

- Ойын жадты дұрыс пайдаланады: ұзақ сессияларда жады ағып кетпейді және тұрақты жұмыс істейді.

- Жүктеме сынақтары жүргізілді - ұзақ ойын уақыты, деңгейлер арасында жылдам ауысулар кезіндегі стресс-тест.

- Ресурстарды тұтыну оңтайландырылды - процессор мен графикалық картаны минималды пайдалану арқылы төмен сипаттамалы құрылғыларда тұрақты жұмыспен қамтамасыз етілді.

Деректер қауіпсіздігі мен құпиялылығын тексеру
Пайдаланушылар деректерін қорғау

- Пайдаланушылар деректері қорғалғаны тексерілді: барлық ойын деректері (прогресс, жетістіктер, статистика) локалдьді жадта сақталады және рұқсатсыз қолжетімділіктен қорғалған.

- Жеке ақпарат желіге берілмейтіні және ойын толық қауіпсіздік стандарттарына сәйкес келетіні расталды.

Жаңартулар және резервтік көшіру

- Автоматты жаңарту жүйесі тексерілді: ойын платформалар арқылы дұрыс жаңартылады (Google Play, App Store, Steam), сақталған прогресті бұзбайды.

Платформалармен сәйкестікті тексеру
Android, iOS және ДК платформалары

- Ойын Android және iOS-тің әртүрлі нұсқаларында дұрыс жұмыс істейтіні тексерілді.

- ДК-нұсқасында Windows 10/11 жүйелерінде тұрақтылық және әртүрлі экран ажыратымдылығы мен графикалық баптаулардағы өнімділік тексерілді.

4.2. Әртүрлі экран рұқсатында тестілеу

- Интерфейс пен басқару элементтерінің әртүрлі экран ажыратымдылығында дұрыс көрсетілуі тексерілді (төмен ажыратымдылықтағы мобильді құрылғылардан бастап, ДК-де толық экран режимдеріне дейін).

5. Соңғы тексерістердің қорытындысы

Өткізілген соңғы тексерістердің қорытындысы бойынша:

- Барлық негізгі ойын механикалары тұрақты және дұрыс жұмыс істейді.

- Өнімділік мәселелері түзетілді, ойын әлсіз құрылғыларда да бірқалыпты жұмыс істейді.

- Локализация және интерфейс мақсатты аудитория талаптарына бейімделген.

- Барлық ойын деректері қорғалған, жаңартулар мен резервтік көшірмелер дұрыс жұмыс істейді.

- Ойын толықтай релизге және Google Play, App Store, Steam сияқты цифрлық платформаларда таратуға дайын.

Steppe Rider барлық тестілеу кезеңінен өтті және кең аудиторияға қолжетімді болуға дайын.

12. ЖАРНАМАЛЫҚ-АҚПАРАТТЫҚ НАУҚАН

Steppe Rider ойынының жарнамалық-ақпараттық науқаны қазақ тілін цифрлық форматта үйретудің жаңа мүмкіндіктерін кең аудиторияға таныстыруға бағытталған кешенді коммуникациялық шаралар жүйесінен тұрады. Науқанның негізгі мақсаты - ойынның функционалдық артықшылықтарын, білім беру құндылығын, инновациялық мүмкіндіктерін көпшілікке түсіндіру және мемлекеттік тіл саясаты аясында әзірленген өнімнің маңызын кеңінен насихаттау. Ақпараттық жұмыс барысында әлеуметтік желілер, медиа жарияланымдар, бейнероликтер мен офлайн таныстырылым форматтары қолданылып, ойынның қолжетімділігі мен қоғамдық қызығушылығын арттыру көзделеді.

БАСПАСӨЗ ПАРАҒЫ

Қазақ тілін үйретуге арналған ойын әзірленді

Қазақстан Республикасы Ғылым және жоғары білім министрлігінің Тіл саясаты комитетінің қолдауымен «Steppe Rider» ойыны әзірленді.

Ойынның мақсаты: 6-25 жас аралығындағы пайдаланушыларға қазақ тілінің А1 және А2 деңгейлерін үйрету және ұлттық болмысты дәріптейтін мазмұнды, сапалы ойын кеңістігін қалыптастыру.

Ойынның компьютерлік және мобильді нұсқалары қазақ тілін инновациялық форматта меңгеруге арналған жаңа құрал ретінде ұсынылып отыр.

Ойынның барлық функционалдық мүмкіндіктерін Астана қаласындағы №104 мектептің оқушылары тестілеп, пайдаланушылардан алынған кері байланыс негізінде ойын жетілдірілді. Бета-тестілеуге қатысушылар «Steppe Rider» жобасының графикасын, ойнау динамикасын және мазмұндық тұтастығын жоғары бағалап, оң пікір білдірді.

Жобаны жүзеге асыруға жиырмадан астам маман: бағдарламашылар, дизайнерлер, тестілеушілер, деректер базасының сарапшылары және басқа да жоғары білікті мамандар қатысты. Ойынның мазмұндық және оқыту компоненттерін әзірлеуге тәжірибелі педагог-әдіскерлер тартылып, білім беру стандарттарына сәйкестігі қамтамасыз етілді.

Директор
Абрашкина Н.

ПРЕСС-РЕЛИЗ

Разработана игра, обучающая казахскому языку

При поддержке Комитета языковой политики Министерства науки и высшего образования Республики Казахстан разработана игра «Steppe Rider».

Цель игры – формирование содержательного и качественного игрового пространства, позволяющего пользователям в возрасте от 6 до 25 лет изучать уровни А1 и А2 казахского языка, применяя национально-культурные элементы.

Компьютерная и мобильная версии игры предлагаются как новый инструмент для изучения казахского языка в инновационном формате.

Все функциональные возможности игры прошли тестирование учащимися школы №104 города Астаны, и на основе полученной от пользователей обратной связи игра была улучшена. Участники бета-тестирования высоко оценили графику проекта «Steppe Rider», динамику игрового процесса и содержательную целостность, выразив положительное мнение.

В реализации проекта участвовали более двадцати специалистов: программисты, дизайнеры, тестировщики, аналитики данных и другие высококвалифицированные сотрудники. В разработке содержательных и обучающих компонентов игры были задействованы опытные педагоги.

Директор
Абрашкина Н.

Жарнамалық-ақпараттық науқан жөнінде

Steppe Rider ойынының кең аудиторияға таныстырылуы мақсатында **желтоқсан айында ауқымды жарнамалық-ақпараттық науқан жүзеге асырылды**. Бұл кампания бірнеше бағытты қамтиды:

Цифрлық платформалардағы ілгерілету (желтоқсан айы)

— Әлеуметтік желілерде ойынның мүмкіндіктерін түсіндіретін арнайы бейнероликтер/суреттер жарияланды.

— Ресми сайт пен медиа беттерде ақпараттық мақалалар мен пресс-релиздер жарияланды.

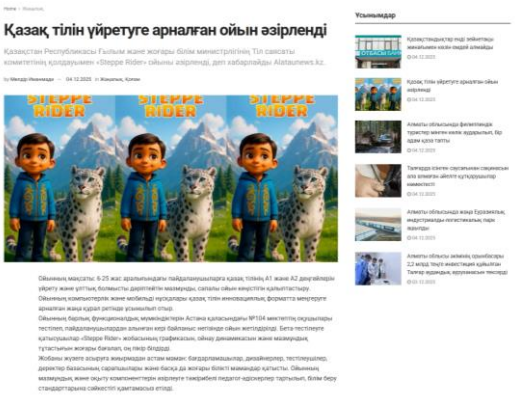
— Баспасөз өкілдеріне арналған арнайы медиа-нұсқа дайындалып, журналистерге толық ақпарат ұсынылды.

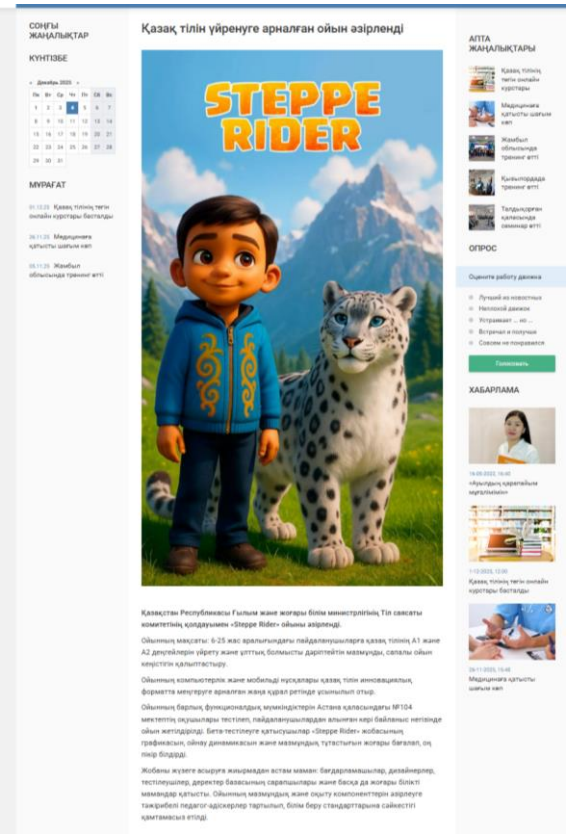
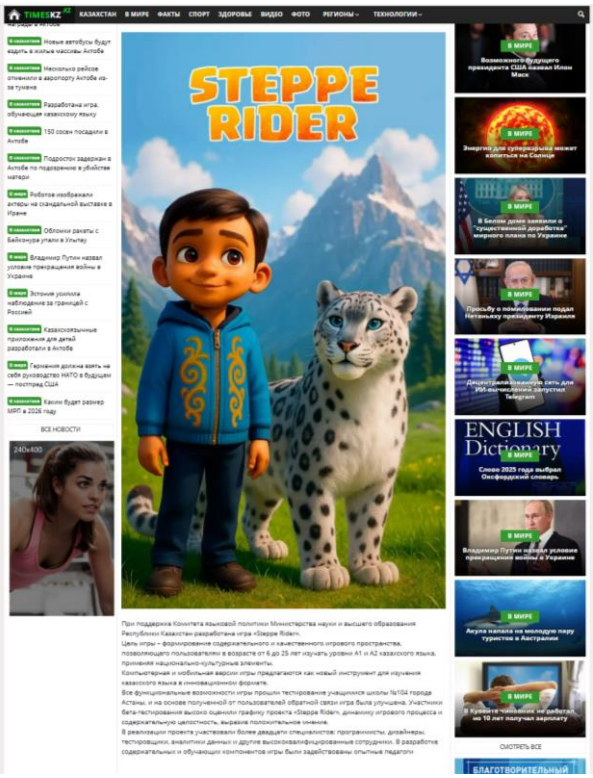
Оқу орындарында ақпарат тарату (қаңтар-ақпан айлары)

— Мектептер мен қосымша білім беру орталықтарына Steppe Rider ойынын таныстыратын әдістемелік парақшалар жеткізіледі.
— Педагогтерге арналған шағын демонстрациялық сессиялар өткізіледі.
— Оқушылар арасында танымдық челлендждер мен ойын-турнирлер ұйымдастыру жоспарланған.

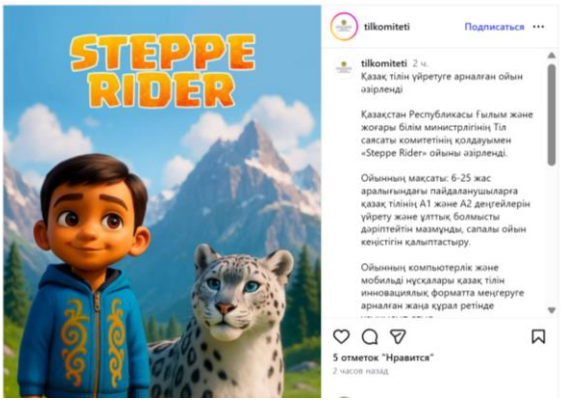
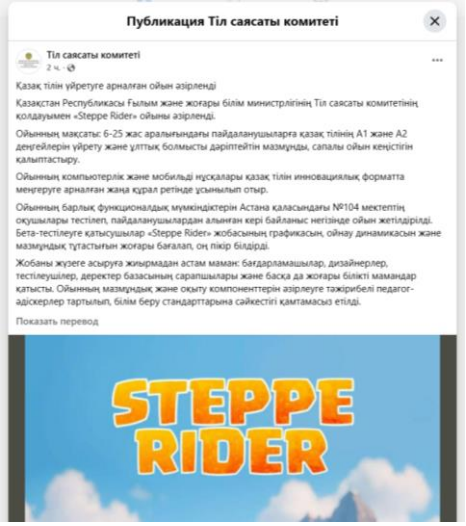
Жарнамалық-ақпараттық науқан Steppe Rider ойынын кең көлемде таныстырып, оны қазақ тілін оқыту жүйесіне тиімді енгізу үшін маңызды қадам болмақ. Кампания жобаның ауқымын кеңейтіп, мемлекеттік тілді меңгеруге арналған заманауи цифрлық шешім ретінде қоғам тарапынан жоғары қолдау қалыптастыруға бағытталған.

Жарнамалық-ақпараттық науқан барысында БАҚ құралдарында және де Тапсырыс берушінің ресми парақшаларында жасалған жарияланымдардың сілтемелері:

https://www.alataunews.kz/zhanalyq/qazaq-tilin-uejretwge-arnalghan-ojyn-aezirlendi/	
https://www.instagram.com/reel/DR1K_FJDKi1/?igsh=bjExaG90d3kzc2c3	

https://kerek-info.kz/index.php?newsid=459	
https://timeskz.kz/v-kazahstane/4269-razrabotana-igra-obuchajuschaja-kazahskomu-jazyku.html	

https://www.gov.kz/memleket/entities/language/press/news/details/1118876?lang=kk	 Қазақ тілін үйретуге арналған ойын әзірленді 1 / 1 04 желтоқсан 2025 12:15 22 Қазақ тілін үйретуге арналған ойын әҚазақстан Республикасы Ғылым және жоғары білім министрлігінің Тіл саясаты комитетінің қолдауымен «Steppe Rider» ойыны әзірленді. Ойынның мақсаты: 6 - 25 жас аралығындағы пайдаланушыларға қазақ тілінің А1 және А2 деңгейлерін үйрету және ұлттық болмысты дәріптейтін мазмұнды, сапалы ойын кеңістігін қалыптастыру. Ойынның компьютерлік және мобильді нұсқалары қазақ тілін инновациялық форматта меңгеруге арналған жаңа құрал ретінде ұсынылып отыр. Ойынның барлық функционалдық мүмкіндіктерін Астана қаласындағы №104 мектептің оқушылары тестілеп, пайдаланушылардан алынған кері байланыс негізінде ойын жетілдірілді. Бета - тестілеуге қатысушылар «Steppe Rider» жобасының графикасын, ойнау динамикасын және мазмұндық тұтастығын жоғары бағалап, оң пікір білдірді. Жобаны жүзеге асыруға жиырмадан астам маман: бағдарламашылар, дизайнерлер, тестілеушілер, деректер базасының сарапшылары және басқа да жоғары білікті мамандар қатысты. Ойынның мазмұндық және оқыту компоненттерін әзірлеуге тәжірибелі педагог - әдіскерлер тартылып, білім беру стандарттарына сәйкестігі қамтамасыз етілді.
https://t.me/tilsayasatykomiteti/3015	 Тіл саясаты комитеті STEPPE RIDER Қазақ тілін үйретуге арналған ойын әзірленді

https://www.instagram.com/p/DR1S4MwjHKB/?igsh=azlteTAwaXR5bnVp	
https://web.facebook.com/story.php?story_fbid=1428348449291214&id=100063482171237&mibextid=wwXIfr&rdid=E9o0qPmOInuVbJbj#	

13. ОЙЫНДЫ БАСҚАРУ ЖҮЙЕСІНЕ, ӘКІМШІЛІК ПАНЕЛІНЕ ҚОЛ ЖЕТКІЗУДІ ҰСЫНУ

Steppe Rider ойынын әзірлеу және релиз кезеңінде ойынға және оның қолдауына басқару процесі ұйымдастырылды, мұнда стандартты әкімшілік панель қолданылмайды. Оның орнына негізгі деректерге қол жеткізу, ойын мониторингі және жаңарту үшін арнайы құралдар мен процедуралар қарастырылған.

Ойынға басқару жүйесіне қол жеткізуді ұйымдастыру

Орталықтандырылған әкімшілік панельдің орнына ойын деректеріне қол жеткізу жүйесі стандартты құралдар мен интеграциялар арқылы әзірленді. Бұл әзірлеу тобы мен тапсырыс берушіге ойынның бірнеше аспектісін жедел басқаруға және бақылауға мүмкіндік береді.

Ойын деректеріне қол жеткізу

- Сөздік базасы, статистика және жетістіктер деректерімен жұмыс істеу үшін жергілікті файлдық дерекқор жүйесі жүзеге асырылды. Ойын ақпараттары (мысалы, жиналған сөздер саны, жетістіктер) файлдарда сақталады, оларды ойын эксплутациясы барысында жаңартуға немесе модификациялауға болады.

- Осы деректерге қол жеткізу кірістірілген әзірлеу құралдары арқылы жүзеге асады, бұл серверге қосылмай-ақ жаңартулармен жұмыс істеуге мүмкіндік береді.

Жаңартулар мен түзетулерді жүйелі жүргізу

- Ойын жаңартулары орналастыру платформалары арқылы жүргізіледі (мысалы, Google Play Console, App Store Connect, Steamworks), мұнда багтарды түзету, контент жаңарту және жетілдірулер бар жаңа жинақтар жүктеледі.

- Пайдаланушылар пікірлері мен анықталған қателерге жедел жауап беру үшін осы платформалардағы кірістірілген есеп беру жүйесі және барлық маңызды және орташа деңгейдегі мәселелерді қадағалау үшін баг-трекинг жүйелері қолданылады.

Пайдаланушылардың кері байланысы және қолдауы

- Пайдаланушылардың барлық кері байланысы Google Play, App Store және Steam арқылы жиналады. Бұл деректер ойынның жетілдірілуі үшін талдау мен приоритеттерді анықтауға қолданылады.

- Пікірлерді жинау үшін сондай-ақ ойынға енгізілген сауалнама жүйесі пайдаланылады, бұл әр жаңарту соңында ойыншылардың пікірлерін жинауға мүмкіндік береді.

Ойынға қол жетімділікті қамтамасыз ету және бақылау тәртібі

Ойынға басқару жүйесіне қол жеткізу әзірлеушілерге, тестілеушілерге және тапсырыс берушіге рөлдік құқықтар жүйесі арқылы беріледі. Әр қолданушы тек өз міндетін орындау үшін қажетті деректер мен құралдарға ғана қол жеткізе алады.

Қол жеткізу құқықтарын бөлуді реттеу

- Әзірлеушілер өзгерістер енгізу үшін бастапқы кодқа және дерекқорға қол жеткізе алады, сондай-ақ ресми дүкендер арқылы жаңартулар жүктей алады.

- Тестілеушілер мен менеджерлер қателер туралы есептерді талдай алады, ойынның жаңа нұсқаларын сынай алады және кері байланыс бере алады.

- Тапсырыс беруші жобаның күйі мен статистикасы туралы есептерге қол жеткізе алады, сондай-ақ жаңартулар мен енгізілген өзгерістердің сапасын тексере алады.

Ойын деректеріне қол жеткізу

Пайдаланушылардың прогресі, жиналған сөздер және ойын валютасы туралы барлық деректер жергілікті түрде ойыншылардың құрылғыларында сақталады, бірақ олар ойын процессін оңтайландыру үшін шығарылып, талдануы мүмкін. Бұл ағымдағы трендтерді бақылауға, пайдаланушылардың әрекетін талдауға және ойын процесін жақсарту үшін жаңартулар енгізуге мүмкіндік береді.

Қазіргі басқару жүйесінің артықшылықтары

- **Икемділік** - команда өзгерістерді жедел енгізе алады, арнайы әкімшілік панель жасауды қажет етпейді.

- **Қарапайымдылық** - жаңартуларды жүктеу және статистиканы бақылау үшін стандартты құралдар (Play Console, App Store Connect, Steam) пайдаланылады.

- **Шығынды азайту** - бөлек әкімшілік панель әзірлеу қажеттілігінің жоқтығы қолдау және әзірлеу шығындарын төмендетеді.

- **Жеделдік** - платформалар мен баг-трекинг жүйелері арқылы өзгерістерді жылдам енгізу кері байланысқа шұғыл жауап беруге мүмкіндік береді.

14. ТАПСЫРЫС БЕРУШІГЕ БАРЛЫҚ БАСТАПҚЫ КОДТАРДЫ ТАПСЫРУ

Steppe Rider ойынын әзірлеудің соңғы кезеңінде және релизге дайындық барысында барлық бастапқы кодтар тапсырыс берушіге берілді. Бұл қадам жобаны толық бақылау, оның одан әрі дамуы және мүмкін жетілдірулерін қамтамасыз ету үшін міндетті болып табылады.

Бастапқы кодтарды беру келесі негізгі кезеңдерді қамтыды:

Жобаның барлық файлдарын жинау

- Ойынның барлық бастапқы кодтары, соның ішінде ойын логикасы, модельдер, текстуралар, анимациялық ресурстар және медиафайлдар (аудио және видео) бір архивке жинақталды.
- Жобамен жұмыс істеуді және келесі кезеңдерде навигацияны жеңілдету үшін каталогтардың құрылымы индустрия стандарттарына сәйкес қамтамасыз етілді.

Кодты тазалау және жүйелеу

- Бастапқы код тестілік деректерден және қажет емес файлдардан (мысалы, уақытша қалталар немесе ескірген нұсқалар) тазартылды.
- Барлық комментарийленген код блоктары, тестілік жолдар және финалдық нұсқамен байланысы жоқ ішкі ескертпелер жойылды, осылайша тек таза және оңтайландырылған код берілді.

Версияларды бақылау жүйесі арқылы беру (Git)

- Кодпен жұмыс істеуді жеңілдету және одан әрі өзгерістер енгізу үшін GitHub платформасында Git-репозиториі жасалып, оған тапсырыс берушіге қол жеткізу берілді.
- Кодтағы барлық өзгерістер версияларды бақылау жүйесі арқылы тіркелді, бұл әрбір модификацияны, жаңартуды және ойынды жетілдіруді қадағалауға мүмкіндік береді.
- Репозиторий жобаның менеджерлері мен әзірлеушілер үшін рұқсаттармен бірге тапсырыс берушіге берілді.

Қосымша барлық ресурстарды қосу

- Кодпен бірге ойынның барлық ресурстары берілді, соның ішінде:
    - 3D-объектілерге арналған модельдер мен текстуралар;
    - Барлық анимациялар мен эффектілер;
    - Ойын логикасына арналған сценарийлер;
    - Дыбыстық және музыкалық файлдар.
- Барлық файлдар жеке архивтерге салынды және келесі пайдалануды жеңілдету үшін нақты құрылымдалды.

15. ЗИЯТКЕРЛІК МЕНШІК ҚҰҚЫҚТАРЫН ТАПСЫРУ

02-04-2025 жылғы жасасқан № 47 мемлекеттік сатып алу туралы шартына сәйкес «Балаларға арналған мемлекеттік тілді үйрететін компьютерлік ойын әзірлеу» жобасының аясында, бір тараптан «United Kingdom» ЖШС, бұдан әрі – *"Құқықты беруші"*, және екінші тараптан Қазақстан Республикасы Ғылым және жоғары білім министрлігі Тіл саясаты комитеті, бұдан әрі – *"Құқықты алушы"*, бірге – *"Тараптар"*, төмендегілер туралы акті жасады:

1. Құқықты беруші актіге сәйкес ««**STEPPE RIDER»** балаларға арналған мемлекеттік тілді үйрететін компьютерлік ойыны» объектісіне қатысты барлық айрықша құқықтарды Құқықты алушыға берді.
2. ««**STEPPE RIDER»** балаларға арналған мемлекеттік тілді үйрететін компьютерлік ойыны» объектісіне айрықша құқықтарды беруімен қатар, осы актіге сәйкес Құқықты беруші келесі құжаттарды тапсырды, ал Құқықты алушы оларды қабылдап алды:

Атауы	Қосымша мәлімет
Ойынның бастапқы және орындалатын кодтары	қосымша-1
Ойын жарияланған платформалардың логиндері мен құпия сөздері	қосымша-2
Серверлік бөлігінің конфигурациясы	қосымша-3
Ойын кейіпкерлерінің бейнесі	ойынға интеграцияланған
UI/UX дизайн элементтері	ойынға интеграцияланған
Ойын ішіндегі барлық визуалды контент (иллюстрациялар, анимациялар, 3D модельдер, карталар)	ойынға интеграцияланған
Ойын ішіндегі барлық аудио контент (дыбыс эффектілері, музыка, дикторлық жазбалар)	ойынға интеграцияланған
«Steppe Rider» атауы	ойынға интеграцияланған
Дизайн стилі, түс палитрасы, фирмалық графикалық элементтер	ойынға интеграцияланған
Қазақ тілін оқытуға арналған модульдік жүйе мен тілдік тапсырмалар банкі	ойынға интеграцияланған

Құқықтарды алушының ««**STEPPE RIDER»** балаларға арналған мемлекеттік тілді үйрететін компьютерлік ойыны» объектісіне берілген айрықша құқықтарға және тапсырылған құжаттарға қатысты Құқықты берушіге ешқандай талаптары жоқ. Тапсырыс беруші ойынға қатысты **барлық элементтердің толық иесі** атанады.

16. ОЙЫННЫҢ МОБИЛЬДІ НҰСҚАСЫН ҰСЫНУ

Steppe Rider ойынының ДК-нұсқасын әзірлеу және сәтті тестілеуден өткізу кезеңінен кейін Android және iOS платформаларына арналған мобильді нұсқасы дайындалып, ұсынылды. Мобильді нұсқа сенсорлық басқару, графика мен өнімділікті оңтайландыру, сондай-ақ әртүрлі экран рұқсаттарын ескере отырып бейімделді.

Ойынның мобильді нұсқасын әзірлеу және бейімдеу

Мобильді нұсқаны жасау үшін келесі қадамдар орындалды:

Графика мен интерфейсті дайындау

- Барлық графикалық элементтер мобильді экрандарға бейімделді, төмен рұқсаттан (HD) бастап жоғары (Full HD, 4K) рұқсатқа дейін.

- Интерфейс оңтайландырылды: басқару элементтері, батырмалар, мәзірлер және ойын индикаторлары сенсорлық басқаруды ескере отырып қайта жасалды, бұл ойыншыларға ойынды интуитивті пайдалануына мүмкіндік береді.

- Барлық интерфейс элементтері (батырмалар мен мәтіндер) кіші экрандарда қолдануға ыңғайлы етіп қайта өңделді, сонымен қатар экранның әртүрлі ориентацияларын (портрет және альбом) ескеріп бейімделді.

Мобильді құрылғыларға оңтайландыру

- Ойынның әртүрлі қуаты бар құрылғыларда (бюджеттіктен премиумға дейін) жұмыс істеу өнімділігі оңтайландырылды. Тесттер жүргізіліп, ойын тұрақты FPS-пен, процессорды жүктемей және көп жады пайдаланбай, бірқалыпты жұмыс істейтіні тексерілді.

- Сенсорлық экранды пайдалану: ойынды басқару қолмен жылжытуға бейімделді, бұл ойыншыларға кейіпкерді жылжыту, әріптерді таңдау және кедергілерден аулақ болуды қарапайым свайптар мен экранді түрту арқылы орындауға мүмкіндік береді.

- Адаптивті графика енгізілді, бұл жылдам рендерингке және минималды ресурстарды тұтынуға мүмкіндік береді.

Ойынның мобильді нұсқаны жариялау

Ойынның әзірлеу және оңтайландыру кезеңінен кейін мобильді нұсқа Google Play Store және Apple App Store платформаларына жариялауға дайындалды.

Google Play платформасында жариялау

- Ойынның Android-нұсқасы үшін Google Play Console-де әзірлеуші аккаунты тіркеліп, қолданба беті дайындалды.
- Ойын қауіпсіздік пен сапа талаптарына сәйкестігін тексеру кезеңінен өтті.
- Тексерістен сәтті өткеннен кейін ойынның мобильді нұсқасы Google Play Store-де жарияланып, Android құрылғылары пайдаланушыларына қолжетімді болды.

Apple App Store платформасында жариялау

- Ойынның iOS-нұсқасы үшін Apple Developer Account тіркеліп, App Store Connect-де ойын беті дайындалды.
- Ойын App Store Review процесінен өтті, бұл қауіпсіздік, контент және функционалдық стандарттарға сәйкестігін тексеруді қамтиды.
- Тексерістен сәтті өткеннен кейін мобильді нұсқа Apple App Store-де жарияланып, iOS пайдаланушылары ойынды жүктеп, ойнай бастады.

Мобильді нұсқаның артықшылықтары

Кең аудиторияға қолжетімділік

Steppe Rider мобильді нұсқасы смартфондар мен планшеттер сияқты барлық заманауи мобильді құрылғыларда қолжетімді. Бұл аудиторияны кеңейтуге және мобильді құрылғылар арқылы оқытуды жөн көретін ойыншыларды тартуға мүмкіндік береді.

Ыңғайлылық және портативтік

Пайдаланушылар Steppe Rider ойынын кез келген уақытта және кез келген жерде ойнай алады, өйткені тұрақты интернетке қосылу міндетті емес.

Бұл әсіресе мобильді нұсқаны сынып ішінде және сыртында өздігінен оқыту құралы ретінде қолданатын білім беру мекемелері үшін маңызды.

Интерфейс бейімделгендігі және басқару

Мобильді нұсқа сенсорлық басқаруға бейімделген, бұл оны барлық пайдаланушыларға ыңғайлы және түсінікті етеді. Қарапайым басқару және түсінікті интерфейс ойыншыларға басқаруды емес, оқыту мен ойын процесіне көңіл бөлуге мүмкіндік береді.

Қорытынды

Steppe Rider мобильді нұсқасы Android және iOS құрылғыларына сәтті әзірленіп, бейімделді. Ол барлық тестілеу кезеңінен өтті және Google Play Store мен Apple App Store-де жарияланды. Мобильді нұсқаны жариялау ойынға кең аудиторияның қолжетімділігін арттырады, сенсорлық басқару арқылы пайдаланушы тәжірибесін жақсартады және кез келген уақытта ойнауға мүмкіндік береді.

Ойынның мобильді нұсқасын әзірлеу кезеңі аяқталып, енді Steppe Rider тек ДК-де ғана емес, мобильді платформаларда да қолжетімді болып, оның танымалдығы мен білім беруде қолданылуын кеңейтуге жаңа мүмкіндіктер ашты.

17. РЕЛИЗ ТУРАЛЫ АНОНС ЖАСАУ

Steppe Rider балаларға қазақ тілін ойын форматында үйретуге арналған заманауи цифрлық өнім ретінде толық әзірленіп, релизге шығуға дайын. Ойынның релизін жариялау (анонс жасау) - жобаның ең маңызды кезеңдерінің бірі болып табылады. Бұл қадам көпшілікке жаңа өнім туралы уақтылы ақпарат беруді, қоғамның қызығушылығын арттыруды және пайдаланушыларды алдағы шығатын нұсқаға алдын ала дайындауды мақсат етеді.

Анонс жұмыстары аясында Steppe Rider ойынының негізгі артықшылықтары, оқу-әдістемелік мазмұны, инновациялық тәсілдері, қолжетімді платформалары (ПК, Android, iOS), сондай-ақ білім беру саласына қосатын үлесі туралы кең көлемді ақпарат таратылды. Сонымен қатар, ойын әзірлеу барысындағы маңызды фактілер, команданың еңбегі, жүргізілген тестілеу нәтижелері және пайдаланушылардың оң пікірлері туралы ақпарат та жарияланды. Бұл шара Steppe Rider жобасының маңыздылығын арттырып қана қоймай, мемлекеттік тіл саясатын қолдаудағы рөлін ерекше айқындай түседі.

Релизге дейінгі кезеңде ақпараттық хабарламалар, медиаға арналған материалдар, әлеуметтік желілердегі жарияланымдар, арнайы промо-видео (трейлер), инфографика және фото-контент дайындалып, жүйелі түрде жарияланды. Анонстың басты міндеті - кең аудиторияны қамту, ойынның мәні мен қажеттілігін түсіндіру және ресми релиз күнін күтуді қызықты, интерактивті ақпараттық процесс ретінде ұйымдастыру.

Ақпаратты тарату арналары

Анонсты жариялау келесі платформалар арқылы жүзеге асады:

- Тапсырыс берушінің ресми сайттары
- Әлеуметтік желідегі парақшалар (Facebook, Instagram, Telegram)
- БАҚ-қа тарату үшін пресс-релиз

Визуалды материалдар дайындау

Анонсты жариялау кезінде міндетті түрде:

- ойын интерфейсінің скриншоттары,
- трейлерден қысқа үзінділер,
- негізгі визуалды баннерлер қолданылады.

18. КӨРСЕТІЛГЕН ҚЫЗМЕТ БОЙЫНША ТОЛЫҚ (ШЫҒАРМАШЫЛЫҚ ЖӘНЕ ҚАРЖЫЛЫҚ) ЕСЕП БЕРУ

Steppe Rider жобасы бойынша атқарылған жұмыстар жөніндегі толық есеп беру процесі шарт талаптарына сәйкес үш кезеңде жүзеге асуы тиіс болатын. Әрбір кезең аяқталғаннан кейін орындаушы тарап тапсырыс берушіге шығармашылық және қаржылық есептерді ұсынуға міндеттелген. Бүгінгі күні атқарылған жұмыстар бойынша екі кезеңнің есебі ұсынылып, тараптармен келісіліп, ресми түрде қабылданған.

1-кезең: Шілде айындағы есеп (орындалған және қабылданған)

Бірінші кезең шілде айында аяқталып, тапсырыс берушіге толық есеп ұсынылды. Бұл кезеңде келесі жұмыстар орындалды:

- Ойынның бастапқы концепциясы мен идеялық негізі әзірленді.
- Оқу-әдістемелік блоктың бастапқы құрылымы жасалды, контенттің лингвистикалық негіздері бекітілді.
- ПК және мобильді нұсқа үшін техникалық талаптар (ТЗ) толық әзірленді және мақұлданды.
- Алғашқы прототип пен ойын әлемінің визуалды моделдері дайындалды.
- Шығармашылық бөлікте - сценарий, жалпы механика, негізгі дизайн бағыттары бекітілді.
- Қаржылық есеп бойынша - осы кезеңде игерілген қаражаттың сметалық бөлінісі ұсынылды.

Тапсырыс беруші тарап барлық материалдарды қарап, актілерге қол қойды.

2-кезең: Қыркүйек айындағы есеп (орындалған және қабылданған)

Екінші кезең қыркүйекте аяқталды және келесі жұмыстарды қамтыды:

- Ойынның толық техникалық архитектурасы жасалды.
- Программистер тарапынан негізгі модульдер: қозғалыс механикасы, интерфейс шаблондары, тілдік тапсырмалар алгоритмдері дайындалды.
- Әдістемелік топ оқу материалының толық құрылымын жасап, деңгейлер мен тарауларды бекітті.
- Графикалық активтер пакеті (кейіпкерлер, локациялар, объектілер) жасалды.
- Мобильді және ПК нұсқаларының алғашқы жұмыс істейтін версиясы құрастырылып, ішкі тестілеуден өтті.
- QA тобы барлық функция бойынша қателерді анықтап, техникалық топқа түзетуге берді.
- Қаржылық есеп бойынша - кезеңге сәйкес жұмсалған қаражаттың толық негіздемесі, смета және қаржылық құжаттар ұсынылды.

Есеп тыңдалды, материалдар толық қабылданып, акт ресми түрде бекітілді.

3-кезең: Қорытынды кезең

Үшінші кезең - жобаның толық аяқталғанын растайтын негізгі есеп. Ол төмендегі жұмыстарды қамтиды:

- Steppe Rider ойынының толық дайын нұсқасының жасалуы.
- Ойынның барлық контентін финалдық өңдеу (қателерді түзету, графиканы жетілдіру, интерфейсті бейімдеу).
- Ойынның техникалық және әдістемелік сараптамадан сәтті өтуі.
- Android, iOS және Windows платформаларына толық бейімдеу.
- Бета-тестілеу:
    - Астана қаласының №104 мектебінде өткізілді.
    - 6–9 сынып оқушылары ойынмен толық танысып, қолдану тәжірибесін тексерді.
    - Пайдаланушылардан ресми кері байланыс жиналып, талданып, ойынға енгізілді.
- Жоба бойынша барлық финалдық материалдарды тапсыру:
    - толық ойын файлы,
    - әдістемелік құрылым,
    - контенттік модульдер,
    - техникалық құжаттама,
    - пайдаланушы нұсқаулығы,
    - тестілеу актілері.
- Соңғы қаржылық есеп - барлық қаражаттың мақсатты жұмсалуын растайтын сметалық және бухгалтерлік құжаттар.

Қазіргі уақытта үшінші кезең бойынша шығармашылық және қаржылық есеп дайындалып, соңғы редакциялау мен құжаттарды рәсімдеу жүзеге асып, тапсырыс берушіге жолданды. Есеп уақытылы тапсырылды.

Steppe Rider - балаларға қазақ тілін үйретуге арналған толыққанды жаңа буын цифрлық білім беру ойын өнімі. Жоба шеңберінде 20-дан астам маман еңбек етті: бағдарламашылар, дизайнерлер, әдіскерлер, сценаристер, QA инженерлер және тағы басқа салалық мамандар.
Жобаның шығармашылық және техникалық нәтижелері:

- қазақ тілін геймификация арқылы үйретуге арналған инновациялық платформа;
- толық функционалдық мобильді және компьютерлік ойын нұсқалары;
- мемлекеттік тіл саясаты талаптарына сәйкес әдістемелік жүйе;
- кең визуалды әлем, сценарий, деңгейлік құрылым;
- дайын релиздік нұсқа және сынақтан өткен өнім.

19. 2026 ЖЫЛДЫҢ ЖЕЛТОҚСАН АЙЫНА ДЕЙІН ТЕГІН ТЕХНИКАЛЫҚ ҚОЛДАУ КӨРСЕТУ

Жобаның аясында тапсырыс берушіге 2026 жылдың желтоқсанына дейін тегін техникалық қолдау көрсетіледі. Бұл қолдау келесі жұмыстарды қамтиды:

- **Қателерді түзету:** Ойында техникалық мәселелер немесе қателер туындаған жағдайда олар тез арада түзетіледі. Бұл ойын ішіндегі багфикстерді, сондай-ақ платформалармен өзара әрекеттесу процесінде немесе пайдаланушы тәжірибесінде туындаған мәселелерді шешуді қамтуы мүмкін.
- **Серверлік инфрақұрылымды қолдау:** Ойынның бэкенд бөлігін қамтамасыз ететін серверлердің тұрақтылығын сақтау жұмыстарын қамтиды. Серверлерде ақаулардың болуын үнемі тексеру, сондай-ақ артатын жүктемелер жағдайында ойынның тұрақтылығы мен өнімділігін қамтамасыз ету үшін серверлік жүйелерді оңтайландыру жүргізіледі.
- **Мониторинг және жаңартулар:** Ойынды үнемі жаңартып, оның барлық функцияларының өзекті болуын және жаңа операциялық жүйелерде дұрыс жұмыс істеуін қамтамасыз ету. Патчтар мен жаңартуларды енгізу арқылы пайдаланушы тәжірибесін және қауіпсіздікті жақсарту.
- **Пайдаланушыларға қолдау көрсету:** Ойыншылардан келетін сұрауларды өңдеу, техникалық мәселелерді шешуге көмектесу, ойын функционалын қолдану бойынша кеңестер беру. Бұл пайдаланушыларға арналған білім базасын жасау және жаңарту жұмыстарын қамтуы мүмкін.
- **Өнімділікті оңтайландыру:** Ойынды әртүрлі құрылғыларда тиімді жұмыс істету, ресурстарды минималды пайдалану және жылдамдықты арттыру мақсатында тұрақты оңтайландыру жұмыстарын жүргізу.
- **Қауіпсіздік:** Ойыншылардың деректерін қорғау, сервер бөлігіндегі мүмкін болатын шабуылдардан сақтау, деректердің ағуын болдырмау және сыртқы қауіптерден қорғау.
- **Жаңа функционалды қолдау:** Ойынға жаңа функциялар немесе қосымшалар енгізу қажет болған жағдайда, техникалық қолдау осы өзгерістерді ойынның ағымдағы функционалына зиян келтірмей интеграциялауды қамтуы мүмкін.

Қолдау ойынның барлық пайдалану кезеңі бойы көрсетілетін болады, және барлық жұмыстар жүйенің тұрақты жұмысын қамтамасыз етуге және туындайтын мәселелерді уақытылы шешуге бағытталған.

20. ӘЗІРЛЕУ БАРЫСЫНДАҒЫ ҚАТЕЛЕРДІ ТҮЗЕТУ

Ойынды әзірлеу барысында әртүрлі кезеңдерде анықталған қателерді түзету жұмыстары жүргізілді. Процесс келесі әрекеттерді қамтиды:

- **Қателерді анықтау және талдау:**
  Әзірлеу кезінде анықталған барлық қателер баг-трекинг жүйесіне тіркеліп, приоритетіне қарай жіктелді. Бұл олардың түзетілуіне жүйелі көзқарасты қамтамасыз етіп, ойынның функционалдығына әсер ететін мәселелерді жедел шешуге мүмкіндік берді.

- **Қателерді классификациялау:**
  Қателер критикалық, жоғары приоритетті және маңызды емес деп бөлінді. Бұл ең маңызды проблемаларға бірінші кезекте көңіл бөлуді қамтамасыз етті. Негізгі функционалды бұзатын критикалық қателер алдымен түзетілді, осылайша ойынның жұмысында бұзушылықтардың болмауы қамтамасыз етілді.

- **Қателерді түзету:**

  — **Кодтағы қателер:** Ойынның логикасындағы қателер түзетіліп, функционалдық багтар жойылды.

  — **Интерфейстегі қателер:** Интерфейс қайта қаралып, элементтердің дұрыс көрсетілуі мен пайдаланушымен өзара әрекеттесу мәселелері түзетілді.

  — **Серверлік бөлік:** Сервер инфрақұрылымындағы қателер түзетіліп, бэкендтің тұрақты жұмысы мен пайдаланушылар деректерінің синхрондалуы қамтамасыз етілді.

- **Түзетуден кейінгі тестілеу:**
  Қателер түзетілгеннен кейін, жаңа проблемалардың туындамауын және ойынның функционалдығының қалпына келгенін тексеру үшін қайта тестілеу жүргізілді. Тұрақтылықты тексеру үшін әртүрлі сценарийлер мен құрылғылар қолданылды.

- **Жаңартулар және релиздер:**
  Барлық түзетулер ойынның тұрақты жаңартуларына енгізілді. Қателерді түзету патчтары пайдаланушыларға жетілдірілген ойын нұсқасына жедел

қол жеткізу мүмкіндігін беру мақсатында тұрақты релиздер аясында шығарылды.

- **Құжаттандыру және кері байланыс:**
  Барлық қателерді түзету жұмыстары құжатталды, бұл әрі қарайғы талдау мен қателердің қайталануын болдырмауға мүмкіндік береді.

Бұл шаралар ойынның сапасын жақсартуға, оның тұрақтылығын арттыруға және пайдаланушыларға жағымды тәжірибе қамтамасыз етуге мүмкіндік берді.

21. ФУНКЦИОНАЛҒА ӨЗГЕРІС ЕНГІЗБЕСТЕН ҰСАҚ ЖЕТІЛДІРУЛЕР ЖАСАУ

Болашақта ойынның негізгі құрылымын өзгертпей, бірақ ойын тәжірибесін жақсартып, контентті әртараптандыруға мүмкіндік беретін бірқатар жақсартулар мен функционалды кеңейтулер жоспарланған. Олар келесі жұмыстарды қамтиды:

- **Жаңа локацияларды қосу:**
  Ойынға географиялық әртүрлілікті кеңейту мақсатында жаңа локациялар енгізілетін болады. Бұл ерекше ойын аймақтары болуы мүмкін, мысалы, ормандар, таулар, шөлейттер және басқа тақырыптық аймақтар, олар ойынның әртүрлі деңгейлерінде қолданылады. Әр жаңа локация ойын процесін байытып, бірегей атмосфера мен визуалды элементтер қосады.

- **Викторина типіндегі жаңа мини-ойындарды қосу:**
  Ойынға викторинаға ұқсас қосымша мини-ойындар енгізілетін болады, олар оқытудың әртүрлі аспектілеріне және ойынның танымдық-ойын элементтеріне бағытталады. Бұл ойыншылардың қызығушылығын арттыруға және ойын механикаларын әртараптандыруға мүмкіндік береді, оқу үшін қосымша нұсқалар қосылады.

- **Басты кейіпкердің әйел нұсқасын қосу:**
  Ойынға басты кейіпкердің балама нұсқасы - әйел кейіпкер енгізіледі, бұл ойыншыларға еркек пен әйел кейіпкерді таңдауға мүмкіндік береді. Бұл жақсарту персонализацияны арттыруға және кеңірек аудиторияны тартуға бағытталған.

- **Бала кейіпкерге жаңа скиндерді қосу:**
  Басты кейіпкердің еркек бейнесіне бірнеше жаңа скиндер енгізіледі. Бұл ойыншыларға кейіпкердің сыртқы келбетін әртараптандыруға, ойын процесін дараландыруға мүмкіндік береді.

- **Барсқа арналған скиндерді қосу:**
  Басты кейіпкерге арналған скиндерден бөлек, кейіпкердің серігі Барсқа арналған скиндер енгізу жоспарланған. Жаңа скиндер ойыншыларға Барстың бірегей сыртқы бейнесін таңдауға мүмкіндік береді, бұл кастомизация элементін қосып, ойын әлемінің көрінісін әртараптандырады.

- **Лидерлер кестесін жақсарту мүмкіндігін қосу:**
  Лидерлер кестесінде ойыншылар өз позицияларын тек жинаған ұпайлары мен прогресске қарай емес, сонымен қатар қосымша жетістіктер арқылы да жақсарта алады. Ойыншылар бірегей титулдарды иеленіп, ойынның әртүрлі аспектілеріндегі жетістіктерге байланысты рейтингтерін көтере алады, бұл оларды тұрақты түрде ойнауға және жетістіктерге жетуге ынталандырады.

Бұл жақсартулар ойыншылардың қызығушылығын арттыруға, ойын процесін персонализациялауды жақсартуға және ойынның негізгі функционалына әсер етпей оның функционалдығын кеңейтуге бағытталған.

Қосымша-1

Ойынның бастапқы және орындалатын кодтары

```
using UnityEngine;
using System.Collections;
public class FadeController : MonoBehaviour
{
   [SerializeField] private CanvasGroup fadePanel;
   [SerializeField] private float fadeDuration = 1f;

   public IEnumerator FadeIn()
   {
      yield return StartCoroutine(Fade(1.2f));
   }

   public IEnumerator FadeOut()
   {
      yield return StartCoroutine(Fade(0f));
   }

   private IEnumerator Fade(float targetAlpha)
   {
      float startAlpha = fadePanel.alpha;
      float time = 0f;

      while (time < fadeDuration)
      {
         time += Time.deltaTime;
         fadePanel.alpha = Mathf.Lerp(startAlpha, targetAlpha, time /
fadeDuration);
         yield return null;
      }

      fadePanel.alpha = targetAlpha;
   }
}



using System.Collections.Generic;
using System.Linq;
using Quiz.Scripts;
using UnityEngine;

public static class GlobalVar
{
   public static GameObject player;
   public static Animator irbisPlayer;
   public static Animator boyPlayer;
   public static PlayerController  playerController;
   public static string IdentId;

//==========================================================
   public static bool needWordReload = false;
   public static bool isMoving = false; // Игрок движется
   public static int difficulty = 0; // Уровень сложности: 0 - легко, 1 - средней
паршивости, 2 - Тяжко
   public static int levelType = 0;// Тип выбора слов. 0 - нет генерации.
   public static int landscapeType = 0; // 1 - степь, 2 - пустыня, 3 - лес, 4 -
горная местность
   /// <summary>
   /// Сколько слов нужно собрать перед тем как появится новый портал
   /// </summary>
   public static int CorrectNumWords=0;




   public static float moveSpeed = 35; //Скорость бега
   /// <summary>
   /// Обычная скорость бега
   /// </summary>
   public static float deffSpeed = 35;
   /// <summary>
   /// Скорость замедления при столкновении с препятствием
   /// </summary>
   public static float slowSpeed = 25;
   public static bool isShock;
   //public static float startSpeed = 20;
   public static bool autoDestroy;
   public static int currScine = 0;
  //=====================ОЧКОВАЯ=========================//
  /// <summary>
  /// конвертируемые мослы
  /// </summary>
   public static int Asyq = 0;
   /// <summary>
   /// Собираемые монетки
   /// </summary>
   public static int Coin = 0;
   /// <summary>
   /// Количество монеток собираемых при включенном бустере и
выключенном (по умолчанию: 1)
   /// </summary>
   public static int CoinAmount = 1;
   /// <summary>
   /// Макс жизней
   /// </summary>
   public static int MaxHealth = 5;
   /// <summary>
   /// Настоящие значение жизней
   /// </summary>
   public static float Health = 0;
   /// <summary>
   /// Необходимое количество сбора костей для конвертации в монету
   /// </summary>
   public static int AsyqtoCoin = 50;
   /// <summary>
   /// Цена за воскрешение
   /// </summary>
   public static int PriceOfResurrection=5;
   /// <summary>
   /// Цена за ошибку
   /// </summary>
   public static int PriceOfError = 25;

   //====================ЯЗЫК================================//
   public static int Lang = 1;
   //==================БУКВЫ=======================//
   public static bool isWordFinished = false; // корректировка на
самоуничтожение если количество слов собрано
   /// <summary>
   /// количество слов собранных во время игровой сессии
   /// </summary>
   public static int currCollectWord;
   public static List<WordData> collectedWords;
   public static int localRecord;
   //=================БУСТЕР=======================//
   public static float busterSpawnChance = 0.1f; //шанс генерации бустера
   public static bool IsAnyBusterActive = false; //Запущен ли какой либо
бустер
   public static bool isGoldenAsyq; // При true суммируемое значение
собираемой монетки увеличивается
   //public static bool isSheld; // При true суммируемое значение собираемой
монетки увеличивается
   public static bool isSpeedUp = false; //Запущен ли бустер ускорения (т.к.
перед порталом скрость снижается, при ускорении этот момент будет
игнорироваться)
   public static bool isImunno = false;

   //================Порталы========================//
   public static bool shouldSpawnPortals = false;
   public static bool portalsSpawned = false;
   //==================Список Тем=========================//
   public static List<ThemeData> allThemes;
   //==================Скины=======================/
   public const string SaveKey = "SkinSaveData";
   public static SkinSaveData data;

   public static float LastCoinTime;
   public static float CoinPitch;
   public static bool WordVoiceover;

   //Выводим формат текста по типу ->  00000123
   public static string FormatScore(int score, int maxValue= 999999999)
   {
      int totalDigits = maxValue.ToString().Length;
      return score.ToString("D" + totalDigits);
   }
   //-----------------------------------------------------------------
   public static readonly string apiUrl = "https://api.stepperider.kz/";

   /// <summary>
   /// Кол-во слов собранных за все время
   /// </summary>
   /// <returns></returns>
   public static int GetCollectedWordsCount()
   {
      var words = FileLoader.LoadWordsFromJSON();
      var collected = words.Where(w => w.IsCollected).ToList();
      return collected.Count;
   }

}
```

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class PanelQueueManager : MonoBehaviour
{
    [SerializeField] float delayBetween = 0.5f;

    readonly Queue<(UIPanel panel, System.Action onBeforeOpen,
System.Action onAfterOpen)> _queue = new();

    bool _running;

    readonly HashSet<UIPanel> _queued = new();   // дедупликация
                                        //bool _running;

    public void Enqueue(UIPanel panel, System.Action onBeforeOpen = null,
System.Action onAfterOpen = null)
    {
        if (!panel || _queued.Contains(panel)) return;
        _queue.Enqueue((panel, onBeforeOpen, onAfterOpen));
        _queued.Add(panel);
        TryRun();
    }

    //public void Enqueue(params UIPanel[] panels)
    //{
    //    if (panels == null) return;
    //    foreach (var p in panels)
    //    {
    //        if (!p || _queued.Contains(p)) continue;
    //        _queue.Enqueue(p);
    //        _queued.Add(p);
    //    }
    //    TryRun();
    //}



    void TryRun()
    {
        if (_running) return;
        _running = true;
        StartCoroutine(Run());
    }

    IEnumerator Run()
    {
        try
        {
            while (_queue.Count > 0)
            {
                var (panel, onBeforeOpen, onAfterOpen) = _queue.Dequeue();
                _queued.Remove(panel);

                EnsureChainActive(panel.transform);

                bool closed = false;
                void Handler() => closed = true;
                panel.Closed += Handler;

                onBeforeOpen?.Invoke();              // ← здесь запускаем
звук/частицы при желании
                if (!panel.IsOpen) yield return panel.OpenRoutine();
                onAfterOpen?.Invoke();               // ← или строго после появления
(после фейда)

                yield return new WaitUntil(() => closed || panel == null ||
!panel.gameObject || !panel.gameObject.activeInHierarchy || !panel.IsOpen);
                panel.Closed -= Handler;

                if (delayBetween > 0f) yield return new
WaitForSecondsRealtime(delayBetween);
            }
        }
        finally { _running = false; }
    }

    static void EnsureChainActive(Transform t)
    {
        var stack = new Stack<Transform>();
        for (var p = t; p != null; p = p.parent) stack.Push(p);
        while (stack.Count > 0)
        {
            var x = stack.Pop();
            if (!x.gameObject.activeSelf) x.gameObject.SetActive(true);
        }
    }
}
```

```
using UnityEngine;
using UnityEngine.Audio;

namespace Scenes.GameScript
{
    /// <summary>
    /// Loads persisted settings and initializes GlobalVar at application start.
    /// Attach to a bootstrap GameObject or rely on
RuntimeInitializeOnLoadMethod.
    /// </summary>
    [DefaultExecutionOrder(-1000)]
    public class SettingLoader : MonoBehaviour
    {
        [Header("Audio Mixer (optional)")]
        [SerializeField] private AudioMixer audioMixer;
        [SerializeField] private float volumeMin = -80f; // dB
        [SerializeField] private float volumeMax = 0f;   // dB

        private void Awake()
        {
            Initialize();
            // Also initialize volumes into the AudioMixer if assigned
            ApplyVolumesToMixerIfAvailable();
        }


[RuntimeInitializeOnLoadMethod(RuntimeInitializeLoadType.BeforeSceneLoad)
]
        private static void InitializeOnLoad()
        {
            Initialize();
        }

        /// <summary>
        /// Reads PlayerPrefs and assigns values into GlobalVar so the rest of the
app can rely on it.
        /// </summary>
        public static void Initialize()
        {
            // IdentId
            GlobalVar.IdentId = PlayerPrefs.GetString("IdentId", "");

            GlobalVar.Asyq = PlayerPrefs.GetInt("Asyq", 0);
            GlobalVar.localRecord = PlayerPrefs.GetInt("LocalRecord", 0);

            // Language
            GlobalVar.Lang = PlayerPrefs.GetInt("Lang", 1);

            GlobalVar.WordVoiceover = PlayerPrefs.GetInt("WordVoiceover", 1) >
0;

            // Difficulty (be tolerant to legacy key name casing)
            if (PlayerPrefs.HasKey("difficulty"))
                GlobalVar.difficulty = PlayerPrefs.GetInt("difficulty", 0);
            else
                GlobalVar.difficulty = PlayerPrefs.GetInt("Difficulty", 0);

            // Other GlobalVar defaults are already set in GlobalVar.cs static fields.
            // Add here when more settings need to be loaded from PlayerPrefs.
        }

        /// <summary>
        /// Apply persisted volume settings to the provided AudioMixer using the
given dB range.
        /// Defaults assume exposed mixer parameters named: MasterVolume,
MusicVolume, SoundVolume, VoiceVolume.
        /// </summary>
        public static void ApplyVolumesToMixer(AudioMixer mixer, float
volumeMin = -80f, float volumeMax = 0f)
        {
            if (mixer == null) return;
            float Master = PlayerPrefs.GetFloat("MasterVolume", 1f);
            float Music  = PlayerPrefs.GetFloat("MusicVolume", 1f);
            float Sound  = PlayerPrefs.GetFloat("SoundVolume", 1f);
            float Voice  = PlayerPrefs.GetFloat("VoiceVolume", 1f);

            mixer.SetFloat("MasterVolume", Mathf.Lerp(volumeMin, volumeMax,
Master));
```

```
        mixer.SetFloat("MusicVolume",  Mathf.Lerp(volumeMin, volumeMax,
Music));
        mixer.SetFloat("SoundVolume",  Mathf.Lerp(volumeMin, volumeMax,
Sound));
        mixer.SetFloat("VoiceVolume",  Mathf.Lerp(volumeMin, volumeMax,
Voice));
    }

    private void ApplyVolumesToMixerIfAvailable()
    {
        if (audioMixer == null) return;
        ApplyVolumesToMixer(audioMixer, volumeMin, volumeMax);
    }
  }
}



using System;
using System.Collections.Generic;
using UnityEngine;

namespace Scenes.GameScript
{
  /// <summary>
  /// Centralized loader/saver and notifier for player skins.
  /// Other systems can subscribe to OnSkinSelected to react to skin changes.
  /// </summary>
  public static class SkinLoader
  {
    /// <summary>
    /// Fired when a skin is selected/applied. Passes the skin index.
    /// </summary>
    public static event Action<int> OnSkinSelected;

    /// <summary>
    /// Notify listeners that a skin has been selected/applied.
    /// Also updates GlobalVar.currScine.
    /// </summary>
    public static void NotifySkinSelected(int index)
    {
        GlobalVar.currScine = index;
        try
        {
            OnSkinSelected?.Invoke(index);
        }
        catch (Exception e)
        {
            Debug.LogException(e);
        }
    }

    /// <summary>
    /// Saves skins and active index to PlayerPrefs and updates
GlobalVar.data.
    /// </summary>
    public static void Save(List<Skin> skins, int activeSkinIndex)
    {
        if (skins == null)
        {
            Debug.LogWarning("SkinLoader.Save called with null skins list");
            return;
        }

        var data = new SkinSaveData
        {
            activeSkinIndex = Mathf.Clamp(activeSkinIndex, 0,
Math.Max(skins.Count - 1, 0)),
            skins = skins
        };

        string json = JsonUtility.ToJson(data);
        PlayerPrefs.SetString(GlobalVar.SaveKey, json);
        PlayerPrefs.Save();
        GlobalVar.data = data;

        Debug.Log("[SkinLoader] Saved: " + json);
    }

    public static int GetSelectedSkin()
    {
        if (!PlayerPrefs.HasKey(GlobalVar.SaveKey))
        {
            return 0;
        }

        string json = PlayerPrefs.GetString(GlobalVar.SaveKey);
        var data = JsonUtility.FromJson<SkinSaveData>(json) ?? new
SkinSaveData();
        GlobalVar.data = data;
        GlobalVar.currScine = data.activeSkinIndex;

        return data.activeSkinIndex;
    }

    /// <summary>
    /// Loads skin data from PlayerPrefs into the provided skins list.
    /// Returns the active skin index (0 if not found). Also updates
GlobalVar.currScine and GlobalVar.data.
    /// </summary>
    public static int LoadInto(List<Skin> skins)
    {
        int activeIndex = 0;

        if (!PlayerPrefs.HasKey(GlobalVar.SaveKey))
        {
            // Default: first skin purchased
            if (skins != null && skins.Count > 0)
                skins[0].isPurchased = true;

            GlobalVar.currScine = 0;
            GlobalVar.data = new SkinSaveData
            {
                activeSkinIndex = 0,
                skins = skins ?? new List<Skin>()
            };

            return 0;
        }

        string json = PlayerPrefs.GetString(GlobalVar.SaveKey);
        var data = JsonUtility.FromJson<SkinSaveData>(json) ?? new
SkinSaveData();
        GlobalVar.data = data;

        // Clamp and apply active index
        if (skins != null && skins.Count > 0)
            activeIndex = Mathf.Clamp(data.activeSkinIndex, 0, skins.Count - 1);
        else
            activeIndex = Mathf.Max(0, data.activeSkinIndex);

        GlobalVar.currScine = activeIndex;

        // Copy purchase flags into provided list
        if (skins != null && data.skins != null)
        {
            for (int i = 0; i < skins.Count; i++)
            {
                if (i < data.skins.Count)
                    skins[i].isPurchased = data.skins[i].isPurchased;
            }
        }

        Debug.Log("[SkinLoader] Loaded: " + json);
        return activeIndex;
    }
  }
}


using System.Collections.Generic;
using UnityEngine;

public class StartGame : MonoBehaviour
{
  [SerializeField] public Animator boyPlayer;
  [SerializeField] public Animator irbisPlayer;
  [SerializeField] private GameObject moveCamera; // ********
****** (******** + ********)
  [SerializeField] private float moveSpeed = 5f; // *********
**************
  [SerializeField] private float rotationThreshold = 0.1f;
  [SerializeField] private float rotateSmooth = 5f;
  [SerializeField] private PlayerController player;
  [SerializeField] public Transform target; // ******** ******
(******** + ********)

  void Awake()
  {
    GlobalVar.irbisPlayer = irbisPlayer;
    GlobalVar.boyPlayer = boyPlayer;
  }

  void Update()
  {
    if (target == null) return;
    // *********, ** ******* ***************
```

```
        moveCamera.transform.position =
Vector3.MoveTowards(moveCamera.transform.position, target.position, 3.0f *
Time.deltaTime);
        // ********* ********** * ************ *****
        moveCamera.transform.rotation =
Quaternion.Slerp(moveCamera.transform.rotation, target.rotation,
Time.deltaTime * rotateSmooth);

        // ********** *************
        float distance = Vector3.Distance(moveCamera.transform.position,
target.position);
        float angle = Quaternion.Angle(moveCamera.transform.rotation,
target.rotation);

        if (distance < 0.5f && angle < rotationThreshold) // **********
*****
        {
            moveCamera.transform.position = target.position;
            moveCamera.transform.rotation = target.rotation;
            GlobalVar.isMoving = true;
            GlobalVar.moveSpeed = moveSpeed;
            GlobalVar.deffSpeed = moveSpeed;
            player.IsDead = false;
            //GlobalVar.Asyq = 0;
            target = null;
        }
    }

    public void MoveToTarget(Transform newTarget)
    {
        target = newTarget;
        GlobalVar.currCollectWord = 0;
        GlobalVar.collectedWords ??= new List<WordData>();
        GlobalVar.collectedWords.Clear();

        boyPlayer.speed = 0f;
        irbisPlayer.speed = 0f;

        boyPlayer.SetTrigger("Run");
        irbisPlayer.SetTrigger("Run");

        irbisPlayer.SetFloat("RunSpeed", 0.6f);
        boyPlayer.SetFloat("RunSpeed", 0.6f);

        boyPlayer.speed = 1f;
        irbisPlayer.speed = 1f;

        GlobalVar.irbisPlayer.SetFloat("RunSpeed", 0.6f);
        GlobalVar.boyPlayer.SetFloat("RunSpeed", 0.6f);

        GlobalVar.isShock = false;
    }

}



using System;
using System.Collections;
using UnityEngine;
using Game;
using UnityEngine.UI;

namespace Scenes.GameScript
{
    public class TutorialController : MonoBehaviour
    {
        [Header("Refs")]
        [SerializeField] private GameObject player;
        [SerializeField] private MobileInputHandler inputHandler;

        [SerializeField] private float lookAheadDistance = 12f; // how far to detect
upcoming obstacles
        [SerializeField] private float criticalDistance = 10f; // how far to detect
critical obstacles
        [SerializeField] private float castRadius = 0.6f;       // spherecast radius to
"feel" the lane
        [SerializeField] private float hintHoldAfterClear = 0.25f; // keep hint a bit
after clear to avoid flicker
        [SerializeField] private Button pauseButton;
        private float _lastThreatTime = -10f;

        public bool IsTutorialActive { get; private set; }

        private enum Step { None, Left, Right, Jump, Done }
        private bool isHintActive = false;
        private Step _step = Step.None;
        private bool onCooldown;

        private void Awake()
        {
            // Ensure hints are hidden by default in editor play via UI listener
            OnShowHint?.Invoke(Dir.None);
        }

        private void OnEnable()
        {
            TrySubscribe();
        }

        private void OnDisable()
        {
            Unsubscribe();
        }

        private void Start()
        {
            IsTutorialActive = PlayerPrefs.GetInt("TutorialComplete", 0) < 1;

            if (IsTutorialActive)
            {
                BeginTutorial();
            }
            else
            {
                // If tutorial is completed, keep all hints hidden initially
                OnShowHint?.Invoke(Dir.None);
            }
        }

        private void BeginTutorial()
        {
            _step = Step.Left;
            isHintActive = false;
            inputHandler.SetMoveLeftEnabled(false);
            inputHandler.SetMoveRightEnabled(false);
            inputHandler.SetJumpEnabled(false);
            pauseButton.enabled = false;
            UpdateHints();
            TrySubscribe();
        }

        public void CompleteTutorial()
        {
            PlayerPrefs.SetInt("TutorialComplete", 1);
            PlayerPrefs.Save();
            IsTutorialActive = false;
            _step = Step.Done;
            OnShowHint?.Invoke(Dir.None);
            inputHandler.EnableAllEvents();
            pauseButton.enabled = true;
            Unsubscribe();
        }

        private void TrySubscribe()
        {
            if (!IsTutorialActive) return;
            if (inputHandler == null) return;

            inputHandler.OnMoveLeft.AddListener(OnMoveLeft);
            inputHandler.OnMoveRight.AddListener(OnMoveRight);
            inputHandler.OnJump.AddListener(OnJump);
        }

        private void Unsubscribe()
        {
            if (inputHandler == null) return;
            inputHandler.OnMoveLeft.RemoveListener(OnMoveLeft);
            inputHandler.OnMoveRight.RemoveListener(OnMoveRight);
            inputHandler.OnJump.RemoveListener(OnJump);
        }

        private void OnMoveLeft()
        {
            if (!IsTutorialActive) return;
            if (_step != Step.Left) return;

            UpdateStep(Step.Right);
        }

        private void OnMoveRight()
        {
            if (!IsTutorialActive) return;
            if (_step != Step.Right) return;
            UpdateStep(Step.Jump);
        }

        private void OnJump()
```

```
{
    if (!IsTutorialActive) return;
    if (_step != Step.Jump) return;
    UpdateStep(Step.Done);
    CompleteTutorial();
}

private void UpdateStep(Step nextStep)
{
    if (isHintActive)
    {
        inputHandler.SetMoveLeftEnabled(false);
        inputHandler.SetMoveRightEnabled(false);
        inputHandler.SetJumpEnabled(false);

        _step = nextStep;
        Time.timeScale = 1f;
        isHintActive = false;
        StartCoroutine(Cooldown());
        OnShowHint?.Invoke(Dir.None);
    }
}

private IEnumerator Cooldown()
{
    onCooldown = true;
    yield return new WaitForSeconds(hintHoldAfterClear);
    onCooldown = false;
}

private void Update()
{
    if (!IsTutorialActive) return;

    if (player == null) return;

    // Only when game is actually running
    if (!GlobalVar.isMoving) return;

    bool threatAhead = IsEnemyAhead(out var hit);

    if (IsCritical() && isHintActive && !onCooldown)
    {
        Time.timeScale = 0f;
    }

    if (threatAhead && !isHintActive && !onCooldown)
    {
        _lastThreatTime = Time.time;



        // prefer a free side: left first, then right
        bool canGoLeft = true;
        bool canGoRight = true;

        if (_step == Step.Left && canGoLeft)
        {
            Time.timeScale = 0.1f;
            isHintActive = true;
            ShowDirectionalHint(Dir.Left);
        }

        if (_step == Step.Right && canGoRight)
        {
            Time.timeScale = 0.1f;
            isHintActive = true;
            ShowDirectionalHint(Dir.Right);
        }

        if (_step == Step.Jump)
        {
            Time.timeScale = 0.1f;
            isHintActive = true;
            ShowDirectionalHint(Dir.Jump);
        }
    }
}


public enum Dir { None, Left, Right, Jump }

public event Action<Dir> OnShowHint;

private void ShowDirectionalHint(Dir dir)
{
    switch (dir)
    {
        case Dir.None:
            break;
        case Dir.Left:
            inputHandler.SetMoveLeftEnabled(true);
            break;
        case Dir.Right:
            inputHandler.SetMoveRightEnabled(true);
            break;
        case Dir.Jump:
            inputHandler.SetJumpEnabled(true);
            break;
        default:
            throw new ArgumentOutOfRangeException(nameof(dir), dir, null);
    }
    OnShowHint?.Invoke(dir);
}

private bool IsCritical()
{
    // Build an origin positioned on the lane
    var pos = player.transform.position + Vector3.forward * criticalDistance +
Vector3.up * 0.5f;

    // Use a sphere cast forward to detect obstacles/NPCs
    RaycastHit hit;
    bool hasHit = Physics.SphereCast(pos, castRadius, Vector3.forward,
out hit, criticalDistance);
    if (!hasHit || hit.collider == null) return false;

    return hit.collider.CompareTag("Obstacle") ||
hit.collider.CompareTag("NPC");
}
public bool IsEnemyAhead(out RaycastHit hit, float
lookAheadDistanceOverride = -1000f)
{
    if (Mathf.Approximately(lookAheadDistanceOverride, -1000f))
lookAheadDistanceOverride = lookAheadDistance;
    // Build an origin positioned on the lane
    var pos = player.transform.position + Vector3.forward *
lookAheadDistanceOverride + Vector3.up * 0.5f;

    // Use a sphere cast forward to detect obstacles/NPCs
    bool hasHit = Physics.SphereCast(pos, castRadius, Vector3.forward,
out hit, lookAheadDistance);
    if (!hasHit || hit.collider == null) return false;

    return hit.collider.CompareTag("Obstacle") ||
hit.collider.CompareTag("NPC");
}

public bool IsCoinAhead(float lookAheadDistanceOverride, out RaycastHit
hit)
{
    // Build an origin positioned on the lane
    var pos = player.transform.position + Vector3.forward *
lookAheadDistanceOverride + Vector3.up * 0.5f;

    // Use a sphere cast forward to detect obstacles/NPCs
    bool hasHit = Physics.SphereCast(pos, castRadius, Vector3.forward,
out hit, lookAheadDistance);
    if (!hasHit || hit.collider == null) return false;

    return hit.collider.CompareTag("Coin");
}

private void UpdateHints()
{
    if(!isHintActive) return;

    switch (_step)
    {
        case Step.Left:
            OnShowHint?.Invoke(Dir.Left);
            break;
        case Step.Right:
            OnShowHint?.Invoke(Dir.Right);
            break;
        case Step.Jump:
            OnShowHint?.Invoke(Dir.Jump);
            break;
        default:
            OnShowHint?.Invoke(Dir.None);
            break;
    }
}

private void OnDrawGizmos()
{
    //draw collision sphere ahead of the player
```

```
        Gizmos.color = Color.green;

        var pos = player.transform.position + Vector3.forward *
lookAheadDistance + Vector3.up * 0.5f;

        Gizmos.DrawWireSphere(pos, castRadius);

        Gizmos.color = Color.red;

        pos = player.transform.position + Vector3.forward * criticalDistance +
Vector3.up * 0.5f;

        Gizmos.DrawWireSphere(pos, castRadius);
    }
  }
}


using System;
using System.Collections;
using UnityEngine;
using UnityEngine.UI;

public class UIPanel : MonoBehaviour
{
  [Header("Optional")]
  [SerializeField] Button closeButton;
  [SerializeField] float showTime = 0.15f;
  [SerializeField] float hideTime = 0.15f;
  [SerializeField] CanvasGroup cg;
  [SerializeField] private AudioSource openAudioSource;
  public event Action Closed;
  public bool IsOpen { get; private set; }

  void Awake()
  {
    if (!cg) cg = GetComponent<CanvasGroup>();
    // ******* ** ********** ******* ***
    if (cg) { cg.alpha = 0f; cg.interactable = false; cg.blocksRaycasts = false; }
  }

  protected virtual void OnEnable()
  {
    if (closeButton) closeButton.onClick.AddListener(OnCloseClicked);
  }

  protected virtual void OnDisable()
  {
    if (closeButton) closeButton.onClick.RemoveListener(OnCloseClicked);
  }

  protected void OnCloseClicked()
  {
    // ******* ***** ********, *****
************* ********* ** ****
    if (isActiveAndEnabled)
      StartCoroutine(CloseRoutine());
  }

  public IEnumerator OpenRoutine()
  {
    // ******* *** ******** (** *** **
************ * Awake)
    if (cg)
    {
      cg.blocksRaycasts = true;
      cg.interactable = true;
      yield return Fade(cg, 0f, 1f, showTime);
    }

    if (openAudioSource) openAudioSource.Play();

    IsOpen = true;
  }

  public IEnumerator CloseRoutine()
  {
    if (!IsOpen) yield break;
    IsOpen = false;

    if (cg)
    {
      cg.interactable = false;
      cg.blocksRaycasts = false;
      yield return Fade(cg, 1f, 0f, hideTime);
    }
    // ******* * ********* *********
(************ *** UI),
    // *** ********* ***, ** ** * Awake:
    gameObject.SetActive(false);

    Closed?.Invoke();
  }

  IEnumerator Fade(CanvasGroup g, float a, float b, float t)
  {
    float time = 0f;
    while (time < t)
    {
      time += Time.unscaledDeltaTime;
      g.alpha = Mathf.Lerp(a, b, time / t);
      yield return null;
    }
    g.alpha = b;
  }
}


using Scenes.Intro;
using UnityEngine;
using UnityEngine.SceneManagement;

public class IntroScene : MonoBehaviour
{
  private void Start()
  {

SceneLoader.Instance.StartLoadScene(SceneManager.GetActiveScene().buildI
ndex + 1);
  }
}


using System;
using System.Collections;
using UnityEngine;
using UnityEngine.SceneManagement;

namespace Scenes.Intro
{
  public class SceneLoader : MonoBehaviour
  {
    public static SceneLoader Instance;
    [SerializeField]
    private float loadTime = 1f;

    private void Awake()
    {
      DontDestroyOnLoad(gameObject);
      Instance = this;
    }

    public void StartLoadScene(int sceneIndex)
    {
      StartCoroutine(LoadScene(sceneIndex));
    }

    private IEnumerator LoadScene(int sceneIndex)
    {
      var currentScene = SceneManager.GetActiveScene();
      yield return SceneManager.LoadSceneAsync("Loading",
LoadSceneMode.Additive);
      yield return
SceneManager.UnloadSceneAsync(currentScene.buildIndex);
      yield return SceneManager.LoadSceneAsync(sceneIndex,
LoadSceneMode.Additive);
      var loadedScene = SceneManager.GetSceneByBuildIndex(sceneIndex);
      SceneManager.SetActiveScene(loadedScene);
      yield return new WaitForSecondsRealtime(loadTime);
      yield return SceneManager.UnloadSceneAsync("Loading");
    }

    public void StartLoadScene(string sceneName)
    {
      StartCoroutine(LoadScene(sceneName));
    }

    private IEnumerator LoadScene(string sceneName)
    {
      var currentScene = SceneManager.GetActiveScene();
      yield return SceneManager.LoadSceneAsync("Loading",
LoadSceneMode.Additive);
      yield return
SceneManager.UnloadSceneAsync(currentScene.buildIndex);
      yield return SceneManager.LoadSceneAsync(sceneName,
LoadSceneMode.Additive);
```

```
            var loadedScene = SceneManager.GetSceneByName(sceneName);
            SceneManager.SetActiveScene(loadedScene);
            yield return new WaitForSecondsRealtime(loadTime);
            yield return SceneManager.UnloadSceneAsync("Loading");
        }
    }
}


using UnityEngine;

public class SkyboxRotate : MonoBehaviour
{
    [SerializeField]
    private Material skyboxMaterial;
    [SerializeField]
    private float rotationSpeed;


    private float rotation = 0f;
    // Update is called once per frame
    void Update()
    {
        rotation += rotationSpeed * Time.deltaTime;
        rotation %=  360f;

        skyboxMaterial.SetFloat("_Rotation", rotation);
    }
}



using System;
using System.Collections;
using System.Text;
using TMPro;
using UnityEngine;
using UnityEngine.Networking;

#nullable enable

[Serializable]
public class CreateUser
{
    public string Login = string.Empty;
}

[Serializable]
public class SetLeader
{
    public int wordCount;
    public string gameSession = string.Empty;
    public string ident = string.Empty;

}

[Serializable]
public class ReturnResponse
{
    public int status;
    public string? Error;
}

[Serializable]
public class CreateUserData
{
    public string? identId;
}
[Serializable]
public class CreateUserResponse : ReturnResponse
{
    public new int status;
    public string error = string.Empty;
    public CreateUserData? data;
}

[Serializable]
public class LeaderItem
{
    public long userId;
    public string login = string.Empty;
    public int words;
    public string session = string.Empty;
}

[Serializable]
public class LeaderWrap { public LeaderItem[]? items ; }

[Serializable]
public class SessionDto
{
    public string ident = string.Empty;
}

public class ApiSender : MonoBehaviour
{
    //private readonly string apiUrl = "https://localhost:44349/";
    LeaderItem[]? ParseLeaders(string json)
    {
        var wrapped = "{\"items\":" + json + "}";
        return JsonUtility.FromJson<LeaderWrap>(wrapped).items;
    }
    private bool postOk ;
    private string? body;
    private string identPlayer = "";

    /// <summary>
    /// Создаем пользователя при старте если игрок не зареган
    /// </summary>

    public IEnumerator CreateUser(string login, Action<bool> onDone)
    {
        if (string.IsNullOrEmpty(login)) { onDone?.Invoke(false); yield break; }

        bool pingOk = false;
        yield return StartCoroutine(ServicePing(ok => pingOk = ok));
        if (!pingOk) {onDone?.Invoke(false); yield break; }

        var user = new CreateUser { Login = login };
        var json = JsonUtility.ToJson(user);

        yield return StartCoroutine(PostRequest(GlobalVar.apiUrl + "Registr", json,
(ok, b) => { postOk = ok; body = b; }));

        if (!postOk)
        {
            onDone?.Invoke(false);
            yield break;
        }

        var resp = JsonUtility.FromJson<CreateUserResponse>(body);
        if (resp != null && resp.status == 0 && resp.data != null &&
!string.IsNullOrEmpty(resp.data.identId))
        {
            identPlayer = resp.data.identId;
        }

        if (!string.IsNullOrEmpty(identPlayer))
        {
            PlayerPrefs.SetString("IdentId", identPlayer);
            GlobalVar.IdentId = identPlayer;
        }

        onDone?.Invoke(true);
    }
    public IEnumerator CreateUser(TMP_InputField inputField, Action<bool>
onDone)
    {
        var login = inputField != null ? inputField.text.Trim() : "";
        yield return CreateUser(login, onDone);
    }

    #region Тащим ТОП-10
        public void TopPlayers(Action<LeaderItem[]?> onDone)
        {
            StartCoroutine(ServicePing(isConnected =>
            {
                if (!isConnected) { Debug.LogError("нет интернета");
onDone?.Invoke(null); return; }
                StartCoroutine(GetTopPlayers(GlobalVar.apiUrl + "TopPlayer",
onDone));
            }));
        }
        private IEnumerator GetTopPlayers(string url, Action<LeaderItem[]?>
onDone)
        {
            using var req = UnityWebRequest.Get(url);
            yield return req.SendWebRequest();

            if (req.result == UnityWebRequest.Result.Success)
            {
                var json = req.downloadHandler.text;
                var leaders = ParseLeaders(json);   // твой парсер
                onDone?.Invoke(leaders);
            }
            else
            {
                Debug.LogError("❌ Ошибка: " + req.error);
```

```
            onDone?.Invoke(null);
        }
    }
#endregion

#region Отправляем свой локальный рекорд в API
    public void SetLeader(int wordCount, string gameSession)
    {
        StartCoroutine(SetLeaderFlow(wordCount, gameSession));
    }
    private IEnumerator SetLeaderFlow(int wordCount, string gameSession)
    {
        bool pingOk = false;
        yield return StartCoroutine(ServicePing(ok => pingOk = ok));
        if (!pingOk)
        {
            Debug.LogError("❌ Нет интернета");
            yield break;
        }

        if (string.IsNullOrEmpty(GlobalVar.IdentId))
        {
            Debug.LogError("❌ Пользователь не зарегистрирован");
            yield break;
        }

        var dto = new SetLeader
        {
            wordCount = wordCount,
            gameSession = gameSession,
            ident = GlobalVar.IdentId
        };
        string json = JsonUtility.ToJson(dto);

        bool postOk = false;
        yield return StartCoroutine(PostRequest(GlobalVar.apiUrl +
"SetLeader", json, (ok, b) => { postOk = ok; body = b; }));

        if (!postOk)
            Debug.LogError("❌ Ошибка при отправке SetLeader");
    }
#endregion

/// <summary>
/// Универсальный метод POST-запросов
/// </summary>
public IEnumerator PostRequest(string url, string json, Action<bool, string>
onDone)
{
    using var req = new UnityWebRequest(url, "POST");
    var bodyRaw = Encoding.UTF8.GetBytes(json);
    req.uploadHandler = new UploadHandlerRaw(bodyRaw);
    req.downloadHandler = new DownloadHandlerBuffer();
    req.SetRequestHeader("Content-Type", "application/json");

    yield return req.SendWebRequest();

    bool ok = req.result == UnityWebRequest.Result.Success;
    string? body = req.downloadHandler?.text;
    onDone?.Invoke(ok, body!);
}

/// <summary>
/// Ведем статистику времени (как самой игры в целом так и время
работы викторины)
/// </summary>
public IEnumerator SendSessionTime(string methodName /*,string session*/)
{
    bool ok = false;
    yield return StartCoroutine(ServicePing(s => ok = s));
    if (!ok) yield break;

    var dto = new SessionDto { ident = GlobalVar.IdentId/*, session =
session*/ };
    string json = JsonUtility.ToJson(dto);
    yield return StartCoroutine(PostRequest(GlobalVar.apiUrl + methodName,
json, (ok, b) => { postOk = ok; body = b; }));
}

/// <summary>
/// Проверка подключения к интернету
/// </summary>
IEnumerator ServicePing(Action<bool> action)
    {
        if (Application.internetReachability ==
NetworkReachability.NotReachable)
        {
            action(false);
            yield break;
        }

        string[] probes = {
            "https://www.gstatic.com/generate_204",
            "https://www.google.com/generate_204"
        };

        foreach (var url in probes)
        {
            using var req = UnityWebRequest.Get(url);
            req.timeout = 3;
            yield return req.SendWebRequest();

            bool ok =
                req.result != UnityWebRequest.Result.ConnectionError &&
                req.result != UnityWebRequest.Result.DataProcessingError &&
                req.responseCode > 0 && req.responseCode < 400;

            if (ok) { action(true); yield break; }
        }
        action(false);
    }

}



using System.IO;
using UnityEngine;

public static class DeviceIdProvider
{
    private const string FileName = "device_id.txt";

    public static string GetDeviceId()
    {
        string id;
        // 1) ►►► ►►►►►►►►►?
        var path = Path.Combine(Application.persistentDataPath, FileName);
        if (File.Exists(path))
        {
            try {
                id = File.ReadAllText(path).Trim();
                if (!string.IsNullOrEmpty(id))
                return id;
            }
            catch { }
        }

        // 2) ►►►►►►►►►►►
        id = null;

#if UNITY_IOS && !UNITY_EDITOR
        // IDFV ► ►►►►►►►►►►►► ►►► ►►►► ►►►►►►►►►►►►►
►►►►►►► ►►►►►►►►►►►►►
```

```
        // 4) ◆◆◆◆◆◆◆◆◆
        try { File.WriteAllText(path, id); } catch { }
        PlayerPrefs.SetString("device_id", id);
        PlayerPrefs.Save();

        return id;
    }

    public static string GetPlatformTag()
    {
#if UNITY_ANDROID
        return "Android";
#elif UNITY_IOS
        return "iOS";
#elif UNITY_STANDALONE_WIN
        return "Windows";
#elif UNITY_STANDALONE_OSX
        return "macOS";
#elif UNITY_STANDALONE_LINUX
        return "Linux";
#else
        return Application.platform.ToString();
#endif
    }
}


using System;
using Unity.VisualScripting;
using UnityEngine;
using UnityEngine.Events;

public class NPCTrigger : MonoBehaviour
{
    public UnityEvent OnPlayerEnter;

    private void OnTriggerEnter(Collider other)
    {
        if(other.CompareTag("Player")) OnPlayerEnter?.Invoke();
    }
}



using System;
using UnityEngine;

public class CameraController : MonoBehaviour
{
    [SerializeField] Transform player;
    [SerializeField] private Transform startCameraPosition;
    [SerializeField] private float horizontalSmooth = 5f; // smoothing speed for
left-right follow
    private Vector3 offset;
    void Start()
    {
        offset = startCameraPosition.position - player.position;
    }
    void FixedUpdate()
    {
        if (!GlobalVar.isMoving) return; // <-- only follow when the game is moving

        Vector3 newPosition = new Vector3(transform.position.x,
transform.position.y, offset.z + player.position.z);
        transform.position = newPosition;
    }

    private void LateUpdate()
    {
        if (!GlobalVar.isMoving) return;

        // Smoothly follow the player's X (left-right) while keeping current Y and Z
already adjusted in FixedUpdate
        float targetX = offset.x + player.position.x;
        float t = Mathf.Clamp01(horizontalSmooth * Time.deltaTime);
        float newX = Mathf.Lerp(transform.position.x, targetX, t);

        Vector3 pos = transform.position;
        pos.x = newX;
        transform.position = pos;
    }
}



using System;
using System.Collections;
using System.Linq;
using DG.Tweening;
using Scenes.Object.Player;
using UnityEngine;

public class DamageController : MonoBehaviour
{
    [SerializeField]
    private Renderer mainRenderer;
    [SerializeField]
    private Busters busters;
    [SerializeField]
    private PlayerSettings playerSettings;

    private bool isPlaying = false;
    private Material damageMaterial;
    private float animationDuration;
    public bool IsInvulnerable => isPlaying;

    private void Awake()
    {
        damageMaterial = playerSettings.DamageMaterial;
        animationDuration = playerSettings.DamageAnimationDuration -
playerSettings.DamageTransitionDuration * 2;
    }

    public void DoDamageAnimation()
    {
        StartCoroutine(DoDamageAnimationCoroutine());
    }
    private IEnumerator DoDamageAnimationCoroutine()
    {
        if (isPlaying) yield break;

        isPlaying = true;
        var blendColor = mainRenderer.material.color;
        damageMaterial.SetColor( "_BlendColor", blendColor);
        damageMaterial.SetFloat("_FadePower", 0f);
        mainRenderer.material = damageMaterial;
        damageMaterial.DOFloat(1f, "_FadePower",
playerSettings.DamageTransitionDuration);
        yield return new WaitForSeconds(animationDuration);
        damageMaterial.DOFloat(0f, "_FadePower",
playerSettings.DamageTransitionDuration);
        var mat = busters.GetCurrentBusterMaterial();
        blendColor = mat.color;
        damageMaterial.SetColor( "_BlendColor", blendColor);
        yield return new WaitForSeconds(0.2f);

        mainRenderer.material = busters.GetCurrentBusterMaterial();

        isPlaying = false;
    }
}



using System.Collections;
using Quiz.Scripts;
using Scenes.UI.Scripts;
using Scenes.GameScript;
using Scenes.Object.Player;
using UnityEngine;
using UnityEngine.UI;

public class PlayerController : MonoBehaviour
{
    [SerializeField] private Transform playerVisual;
    [SerializeField] private GameObject boy;
    [SerializeField] private GameObject irbis;

    [Header("Контроллеры")] [SerializeField]
    private TileGeneration tileGenerator;

    [SerializeField] private FadeController fadeController;
    [SerializeField] protected MenuController menuController;
    [SerializeField] private DamageController damageController;
    [SerializeField] private CollectWordUIController collectWordUIController;
    [SerializeField] private NavBarController navBarController;
    [SerializeField] private SessionTracker sessionTracker;
    [SerializeField] private CoinFlyEffect coinFlyEffect;
    [SerializeField] private ApiSender apiSender;

    [Header("Монет за Асык")] [SerializeField]
    private int AsyqtoCoin = 30;

    [Header("Панели")] [SerializeField] private GameObject fade;
    //[SerializeField] private GameObject winnerPanel;

    [SerializeField] public GameObject respawnCanvas;
```

```
    [SerializeField] private GameObject cameras;

    [SerializeField] private float healthValue;
    [SerializeField] private float lineDistance = 4;
    [SerializeField] private float jumpForce = 10;
    [SerializeField] private float jumpDelay = 0.2f;
    [SerializeField] private float gravity = -20;
    [SerializeField] private GameObject losePanel;
    [SerializeField] private float duration = 15f; // Время действия бустера
    [SerializeField] public Animator irbisPlayer;
    [SerializeField] public Animator boyPlayer;

    [Space]
    [Header("Анимации")]
    [SerializeField] private float lineMoveSpeed = 25f;
    [SerializeField] protected float tiltStrengthZ;
    [SerializeField] protected float tiltStrengthY;

    [SerializeField] private float tiltDuration;
    [SerializeField] private float returnDuration;


    [Header("Цена за воскрешение")] [SerializeField]
    private int PriceOfResurrection = 5;

    [Header("Цена за ошибку")] [SerializeField]
    private int PriceOfError = 25;


    [Header("Шанс выпадения бустера")] [SerializeField]
    private float busterSpawnChance = 0.3f;

    [Header("Звуковые эффекты")] [SerializeField]
    private AudioClip[] audioClips;

    [Header("Скины")] [SerializeField] private GameObject[] scinsBoy;
    [SerializeField] private int currScine;

    [Header("Зап.Части пацана")] [SerializeField]
    private GameObject[] partsBoy;

    private Vector3 startPosition;
    private Vector3 startCameraPosition;
    private Vector3 dir;

    protected CharacterController controller;
    private Busters buffs;
    private AudioSource boyAudioSource;
    private AudioSource irbisAudioSource;
    private Quaternion baseRotation;

    protected int lineToMove = 1;
    private readonly float joystickThreshold = 0.5f;
    private bool stickInUse;
    protected bool canJump = true;

    // When false, built-in input reading in Update() is disabled to allow injected
input from subclasses
    protected bool enableBuiltInInput = true;

    // Настройки дефолтов
    private const int TotalSkins = 10;
    private static readonly int[] DefaultPrices = { 0, 20, 30, 40, 50, 60, 70, 80, 90,
110 };


//==========================================================================//
    [SerializeField] private float hitSlowPercent = 0.10f; // 10% замедление
    [SerializeField] private float hitSlowDuration = 0.6f; // сколько держать
замедление
    [SerializeField] private float hitSlowRamp = 0.12f; // плавный вход/выход
(сек)

    private float hitSlowMul = 1f; // текущий множитель (1..0.9)
    private float hitSlowEndTime = -1f; // когда закончится замедление
    private Coroutine hitSlowCo;

//==========================================================================//

    [SerializeField] private Button btnRespawn;
    [SerializeField] private Button btnDie;

    public bool IsDead;

    //Игровая сессия

    #region Скины

    private void UpdateSkins()
    {
        if (currScine != GlobalVar.currScine)
        {
            scinsBoy[currScine].SetActive(false);
            scinsBoy[GlobalVar.currScine].SetActive(true);
            currScine = GlobalVar.currScine;
            switch (currScine)
            {
                case 0:
                    partsBoy[0].SetActive(true);
                    partsBoy[1].SetActive(true);
                    break;
                case 1:
                    partsBoy[0].SetActive(true);
                    partsBoy[1].SetActive(false);
                    break;
                case 2:
                    partsBoy[0].SetActive(true);
                    partsBoy[1].SetActive(true);
                    break;
                case 3:
                    partsBoy[0].SetActive(true);
                    partsBoy[1].SetActive(true);
                    break;
                case 4:
                    partsBoy[0].SetActive(true);
                    partsBoy[1].SetActive(false);
                    break;
                case 5:
                    partsBoy[0].SetActive(true);
                    partsBoy[1].SetActive(true);
                    break;
                case 6:
                    partsBoy[0].SetActive(false);
                    partsBoy[1].SetActive(false);
                    break;
                case 7:
                    partsBoy[0].SetActive(false);
                    partsBoy[1].SetActive(false);
                    break;
                case 8:
                    partsBoy[0].SetActive(false);
                    partsBoy[1].SetActive(false);
                    break;
                case 9:
                    partsBoy[0].SetActive(false);
                    partsBoy[1].SetActive(true);
                    break;
            }
        }
    }

    #endregion

    public void ApplySkin(int index)
    {
        if (scinsBoy == null || scinsBoy.Length == 0) return;
        if (index < 0 || index >= scinsBoy.Length) index = 0;

        // Activate only selected skin
        for (int i = 0; i < scinsBoy.Length; i++)
            scinsBoy[i].SetActive(i == index);

        currScine = index;
        GlobalVar.currScine = index;

        switch (index)
        {
            case 0:
                partsBoy[0].SetActive(true);
                partsBoy[1].SetActive(true);
                break;
            case 1:
                partsBoy[0].SetActive(true);
                partsBoy[1].SetActive(false);
                break;
            case 2:
                partsBoy[0].SetActive(true);
                partsBoy[1].SetActive(true);
                break;
            case 3:
                partsBoy[0].SetActive(true);
                partsBoy[1].SetActive(true);
                break;
            case 4:
                partsBoy[0].SetActive(true);
                partsBoy[1].SetActive(false);
```

```
            break;
        case 5:
            partsBoy[0].SetActive(true);
            partsBoy[1].SetActive(true);
            break;
        case 6:
            partsBoy[0].SetActive(false);
            partsBoy[1].SetActive(false);
            break;
        case 7:
            partsBoy[0].SetActive(false);
            partsBoy[1].SetActive(false);
            break;
        case 8:
            partsBoy[0].SetActive(false);
            partsBoy[1].SetActive(false);
            break;
        case 9:
            partsBoy[0].SetActive(false);
            partsBoy[1].SetActive(true);
            break;
    }
}

private void OnEnable()
{
    SkinLoader.OnSkinSelected += ApplySkin;
}

private void OnDisable()
{
    SkinLoader.OnSkinSelected -= ApplySkin;
}

private void Start()
{
    GlobalVar.PriceOfResurrection = PriceOfResurrection;
    buffs = GetComponent<Busters>();
    controller = GetComponentInChildren<CharacterController>();
    boyAudioSource = boy.GetComponent<AudioSource>();
    irbisAudioSource = irbis.GetComponent<AudioSource>();

    GlobalVar.busterSpawnChance = busterSpawnChance;
    GlobalVar.AsyqtoCoin = AsyqtoCoin;
    startCameraPosition = cameras.transform.position;
    startPosition = transform.position;
    GlobalVar.Health = healthValue;
    GlobalVar.MaxHealth = (int)healthValue;


    GlobalVar.player = gameObject;
    GlobalVar.playerController = GetComponent<PlayerController>();
    baseRotation = playerVisual.rotation;

    GlobalVar.PriceOfError = PriceOfError;

    // Apply the active skin at scene start
    ApplySkin(SkinLoader.GetSelectedSkin());
}

private void Update()
{
    if (enableBuiltInInput)
    {
        var horizontalInput = Input.GetAxisRaw("Horizontal");
        if (Input.GetKeyDown(KeyCode.Escape) && GlobalVar.isMoving)
        {
            if (Time.timeScale == 1f)
                menuController.PauseGame();
            else
                menuController.ResumeGame();
        }

        if (GlobalVar.isMoving && !GlobalVar.isShock)
        {
            if (horizontalInput > joystickThreshold && !stickInUse)
            {
                if (lineToMove < 2)
                {
                    lineToMove++;
                    StartCoroutine(TiltPlayer(-tiltStrengthZ, tiltStrengthY)); // Вправо
— наклон влево
                }

                stickInUse = true;
            }
            else if (horizontalInput < -joystickThreshold && !stickInUse)
            {
                if (lineToMove > 0)
                {
                    lineToMove--;
                    StartCoroutine(TiltPlayer(tiltStrengthZ, -tiltStrengthY)); // Влево
— наклон вправо
                }

                stickInUse = true;
            }

            if (Mathf.Abs(horizontalInput) < 0.1f) stickInUse = false;

            //irbisPlayer.SetTrigger("Jump");
            if ((Input.GetKeyDown(KeyCode.Space) ||
Input.GetButtonDown("Jump")) && controller.isGrounded &&
                canJump) Jump();
        }
    }


    // Надежная фиксация позиции по осям (Z — текущее положение, X —
смещение по линии)
    var targetX = 0f;
    if (lineToMove == 0)
        targetX = -lineDistance;
    else if (lineToMove == 2)
        targetX = lineDistance;

    var targetPosition = new Vector3(targetX, transform.position.y,
transform.position.z);
    var diff = targetPosition - transform.position;
    var moveDir = diff.normalized * (lineMoveSpeed * Time.deltaTime);

    if (diff.sqrMagnitude > 0.001f)
    {
        if (moveDir.sqrMagnitude < diff.sqrMagnitude)
            controller.Move(moveDir);
        else
            controller.Move(diff);
    }
}

//=========Поворот игрока при повороте===================//
protected IEnumerator TiltPlayer(float tiltZ, float tiltY)
{
    var tiltRotation = baseRotation * Quaternion.Euler(0, tiltY, tiltZ);

    // Наклон
    var elapsed = 0f;
    while (elapsed < tiltDuration)
    {
        playerVisual.localRotation = Quaternion.Slerp(baseRotation, tiltRotation,
elapsed / tiltDuration);
        elapsed += Time.deltaTime;
        yield return null;
    }

    playerVisual.localRotation = tiltRotation;

    // Возврат
    elapsed = 0f;
    while (elapsed < returnDuration)
    {
        playerVisual.localRotation = Quaternion.Slerp(tiltRotation, baseRotation,
elapsed / returnDuration);
        elapsed += Time.deltaTime;
        yield return null;
    }

    playerVisual.localRotation = baseRotation;
}

protected void Jump()
{
    dir.y = jumpForce;

    irbisPlayer.SetTrigger("Jump");
    boyPlayer.SetTrigger("Jump");

    canJump = false;
    StartCoroutine(JumpReset());
    irbisAudioSource.Play();
}

private IEnumerator JumpReset()
{
    yield return new WaitForSeconds(jumpDelay);
    canJump = true;
}
```

```
    private void FixedUpdate()
    {
        // контроллер не инициализирован или временно отключён — вообще
ничего не двигаем
        if (controller == null || !controller.enabled)
            return;

        if (!GlobalVar.isMoving || GlobalVar.isShock)
        {
            dir = Vector3.zero;
            if (controller.isGrounded)
                controller.Move(new Vector3(0f, -2f, 0f) * Time.fixedDeltaTime);
            return;
        }

        dir.z = GlobalVar.moveSpeed;
        dir.y += gravity * Time.fixedDeltaTime;
        controller.Move(dir * Time.fixedDeltaTime);

        if (controller.isGrounded && dir.y < 0f)
            dir.y = -2f;
    }

    private void OnTriggerEnter(Collider other)
    {
        HandleCoin(other);

        if (HandleObstacles(other)) return;

        HandleBoosters(other);
    }

    private void HandleCoin(Collider other)
    {
        if (!other.CompareTag("Coin")) return;


        GlobalVar.Coin += GlobalVar.CoinAmount;
        if (GlobalVar.Coin >= GlobalVar.AsyqtoCoin)
        {
            GlobalVar.Asyq += 1;
            GlobalVar.Coin = 0;
            PlayerPrefs.SetInt("Asyq", GlobalVar.Asyq);
        }

        coinFlyEffect.PlayAnimation();
    }

    private bool HandleObstacles(Collider other)
    {
        //===========Минусуем жизнь при столкновении с
НеПиСем=============\\
        if (other.CompareTag("Obstacle") || other.CompareTag("NPC"))
        {
            if (IsDead)
            {
                other.gameObject.SetActive(false);
                return true;
            }

            if ((buffs != null && buffs.IsInvulnerable()) ||
damageController.IsInvulnerable)
            {
                Debug.Log("💥 Столкновение проигнорировано — игрок
неуязвим");
                //Destroy(other.gameObject); // всё равно убираем препятствие
                other.gameObject.SetActive(false);
                return true;
            }

            if (other.CompareTag("Obstacle") && !GlobalVar.isSpeedUp)
            {
                GlobalVar.irbisPlayer.SetFloat("RunSpeed", 0.85f);
                GlobalVar.boyPlayer.SetFloat("RunSpeed", 0.85f);
                boyPlayer.SetFloat("RunSpeed", 0.85f);
                irbisPlayer.SetFloat("RunSpeed", 0.85f);
                GlobalVar.moveSpeed = GlobalVar.slowSpeed;
                TriggerHitSlow();
            }


            if (GlobalVar.Health > 0)
            {
                GlobalVar.Health -= 10;
                GlobalVar.Health = Mathf.Clamp(GlobalVar.Health, 0, healthValue); //
чтобы не уходило в минус
                damageController.DoDamageAnimation();
            }
            if (GlobalVar.Health <= 0)
            {
                IsDead = true;
                GlobalVar.isShock = true;
                GlobalVar.isMoving = false;
                irbisPlayer.SetTrigger("Shock");
                boyPlayer.SetTrigger("Shock");

                if (GlobalVar.Asyq < PriceOfResurrection)
                {
                    buffs.ResetAllBoosters(); //
                    StartCoroutine(RestartAfterDeath());
                }
                else
                {
                    respawnCanvas.SetActive(true);
                }
            }
        }

        return false;
    }

    private void HandleBoosters(Collider other)
    {

//================Бустеры=================================
====\\
        //Иммунитет
        if (other.CompareTag("Imunno"))
        {
            buffs.ActivateImmunity(duration);
            buffs.ActivateBusterUI("Иммунитет", duration);
        }

        //Неуязвимость
        if (other.CompareTag("Invulnerability"))
        {
            buffs.ActivateInvulnerability(duration);
            buffs.ActivateBusterUI("Неуязвимость", duration);
        }

        //Магнит
        if (other.CompareTag("Magnetto"))
        {
            buffs.ActivateMagnet(duration);
            buffs.ActivateBusterUI("Магнит", duration);
        }

        //Умножение монет
        if (other.CompareTag("Asyq"))
        {
            buffs.ActivateGoldenAsyq(duration);
            buffs.ActivateBusterUI("Золото X2", duration);
        }

        //Ускорение
        if (other.CompareTag("SpeedUp"))
        {
            buffs.ActivateSpeed(duration);
            buffs.ActivateBusterUI("Скорость", duration);
        }
    }

    private void TriggerHitSlow()
    {
        var targetMul = 1f - hitSlowPercent; // 0.9 при 10%
        // не давим ниже порога: просто обновляем таймер
        hitSlowEndTime = Time.time + hitSlowDuration;

        if (hitSlowMul > targetMul)
        {
            // впервые замедляемся (или возвращались) — запустить плавный
вход
            if (hitSlowCo != null) StopCoroutine(hitSlowCo);
            hitSlowCo = StartCoroutine(HitSlowRoutine(targetMul));
        }
        // если уже на пороге (== targetMul), корутина уже крутится — просто
продлили время
    }

    private IEnumerator HitSlowRoutine(float targetMul)
    {
        // плавный вход
        var start = hitSlowMul;
        var t = 0f;
        while (t < hitSlowRamp)
        {
            t += Time.deltaTime;
```

```
        float k = Mathf.Clamp01(t / hitSlowRamp);
        hitSlowMul = Mathf.Lerp(start, targetMul, k);
        yield return null;
    }

    hitSlowMul = targetMul;

    // держим пока не истечёт последний продлённый таймер
    while (Time.time < hitSlowEndTime)
        yield return null;

    // плавный выход к 1.0
    t = 0f;
    start = hitSlowMul; // обычно targetMul
    while (t < hitSlowRamp)
    {
        t += Time.deltaTime;
        var k = Mathf.Clamp01(t / hitSlowRamp);
        hitSlowMul = Mathf.Lerp(start, 1f, k);
        yield return null;
    }

    hitSlowMul = 1f;
    hitSlowCo = null;


    GlobalVar.irbisPlayer.SetFloat("RunSpeed", 1.0f);
    GlobalVar.boyPlayer.SetFloat("RunSpeed", 1.0f);
    boyPlayer.SetFloat("RunSpeed", 1.0f);
    irbisPlayer.SetFloat("RunSpeed", 1.0f);
    GlobalVar.moveSpeed = GlobalVar.deffSpeed;
}

public void EndGame()
{
    sessionTracker.StopTimer();
    StartCoroutine(DisableAfterSound());
    respawnCanvas.SetActive(false);
    StartCoroutine(RestartAfterDeath());
    menuController.StopGame();
}

public void RespawnGame()
{
    GlobalVar.Asyq -= GlobalVar.PriceOfResurrection;
    UnpauseGame();
}

private void UnpauseGame()
{
    PlayerPrefs.SetInt("Asyq", GlobalVar.Asyq);
    GlobalVar.Health = healthValue;
    respawnCanvas.SetActive(false);
    GlobalVar.moveSpeed = GlobalVar.deffSpeed;
    GlobalVar.isShock = false;
    GlobalVar.isMoving = true;
    IsDead = false;

    boyPlayer.ResetTrigger("Jump");
    boyPlayer.ResetTrigger("Shock");

    irbisPlayer.SetTrigger("Run");
    boyPlayer.SetTrigger("Run");

    StartCoroutine(DisableAfterSound());
}

public IEnumerator RestartAfterDeath()
{
    // Стопим движение и показываем затемнение
    GlobalVar.isMoving = false;
    GlobalVar.isShock = true; // чтобы апдейты точно не дёргали движение
    fade.SetActive(true);


    navBarController.DisablePauseButton();

    collectWordUIController.PlayWordCloseAnimation();
    // Плавно затемняем экран
    yield return StartCoroutine(fadeController.FadeIn());

    // Анимации окончания (если нужны)
    irbisPlayer.SetTrigger("EndGame");
    boyPlayer.SetTrigger("EndGame");

    // Сброс всех бустеров/эффектов
    buffs.ResetAllBoosters();

    // Отключаем CC перед телепортом/изменениями позиций
    if (controller != null) controller.enabled = false;

    // Сброс внутренних векторов/флагов
    dir = Vector3.zero;
    lineToMove = 1;

    // Телепорт игрока в стартовую позицию
    transform.position = startPosition;
    // На всякий случай — если CC на дочернем объекте
    if (controller != null) controller.transform.position = startPosition;

    // Камера — на старт
    cameras.transform.position = startCameraPosition;
    cameras.transform.rotation = Quaternion.Euler(0f, 0f, 0f);

    // Глобальные значения
    GlobalVar.autoDestroy = true; // чтобы автоочистки спокойно прошли
    GlobalVar.moveSpeed = 0f;
    GlobalVar.isSpeedUp = false;
    GlobalVar.levelType = 0;

    GlobalVar.Health = healthValue;

    // Перезапуск зоны (тайлы/обстacles/буквы и т.д.)
    tileGenerator.RestartGameZone();

    // Подстраховка: короткая пауза на чёрном экране
    yield return new WaitForSeconds(1.2f);
    menuController.PlayMainTheam(MainTheme.Menu);
    // Очистка UI со словом
    tileGenerator.CleanUI();


     // Дадим кадр на применение всех позиций до включения CC
    yield return new WaitForEndOfFrame();

    // Включаем CC обратно
    if (controller != null) controller.enabled = true;

    // Готовим игру к ожиданию старта в меню
    GlobalVar.autoDestroy = false;
    GlobalVar.isShock = false; // игрок “жив”, но не бежит
    GlobalVar.isMoving = false; // запуск начнёте из меню

    navBarController.OpenNavBar();
    // Плавно возвращаем изображение
    yield return StartCoroutine(fadeController.FadeOut());

    fade.SetActive(false);

    if (GlobalVar.currCollectWord > 0)
    {
        menuController.ShowNewWordPanel(GlobalVar.collectedWords);
    }

    if (GlobalVar.currCollectWord > GlobalVar.localRecord)
    {
        menuController.ShowWinnerPanel(GlobalVar.currCollectWord);
        GlobalVar.localRecord = GlobalVar.currCollectWord;
        PlayerPrefs.SetInt("LocalRecord", GlobalVar.currCollectWord);
        apiSender.SetLeader(GlobalVar.currCollectWord,
sessionTracker.GetFormatedTime());
        PlayerPrefs.Save();
    }

    GlobalVar.currCollectWord = 0;
}


public IEnumerator LoadAndPlayAudio(WordData word)
{
    bool isPlaying = false;

    yield return FileLoader.GetWordAudioClipStreaming(word, (clip) =>
    {
        boyAudioSource.clip = clip;
        boyAudioSource.Play();
        isPlaying = true;
    });

    while (!isPlaying) yield return new WaitForEndOfFrame();

    var clipLength = boyAudioSource.clip.length;

    yield return new WaitForSeconds(clipLength + 0.1f);
}

private IEnumerator DisableAfterSound()
{
```

```
        yield return new WaitForSeconds(0.1f);
        //Time.timeScale = 1f;
        respawnCanvas.SetActive(false);
    }

    public void OnIncorrectLetter()
    {
        if (GlobalVar.Coin > 0)
        {
            GlobalVar.Coin -= GlobalVar.PriceOfError;
            if (GlobalVar.Coin < 0) GlobalVar.Coin = 0;
        }
    }

    public int GetCurrentLine()
    {
        return lineToMove;
    }

    public void StopGame()
    {
        EndGame();
    }

    public void ContinueGame()
    {
        UnpauseGame();
    }
}



using UnityEngine;

namespace Scenes.Object.Player
{
    [CreateAssetMenu(fileName = "PlayerSettings", menuName =
"Player/PlayerSettings", order = 1)]
    public class PlayerSettings : ScriptableObject
    {
        [field: SerializeField]
        public Material DamageMaterial { get; private set; }
        [field: SerializeField]
        public float DamageAnimationDuration { get; private set; } = 1f;

        [field: SerializeField] public float DamageTransitionDuration { get; private
set; } = 0.2f;
    }
}


using System;
using UnityEngine;

namespace Scenes.Object.Player
{
    public class SessionTracker : MonoBehaviour
    {
        public float ElapsedTime { get; private set; }

        [SerializeField]
        private bool useUnscaledTime;

        private bool isRunning;

        public void ResetTimer()
        {
            ElapsedTime = 0f;
        }

        public void StartTimer()
        {
            isRunning = true;
            ElapsedTime = 0f;
        }

        private void Update()
        {
            if(!isRunning) return;

            ElapsedTime += useUnscaledTime ? Time.unscaledDeltaTime :
Time.deltaTime;
        }

        public void StopTimer()
        {
            isRunning = false;
        }

        public string GetFormatedTime()
        {
            //format elapsed time as "hh:mm:ss.ms"v
            var span = TimeSpan.FromSeconds(ElapsedTime);
            return span.ToString(@"mm\:ss\.fff");
        }
    }
}


using UnityEngine;

public class SwipeController : MonoBehaviour
{
    public static bool tap, swipeLeft, swipeRight, swipeUp, swipeDown;
    private bool isDraging = false;
    private Vector2 startTouch, swipeDelta;

    private void Update()
    {
        tap = swipeDown = swipeUp = swipeLeft = swipeRight = false;
#region -
        if (Input.GetMouseButtonDown(0))
        {
            tap = true;
            isDraging = true;
            startTouch = Input.mousePosition;
        }
        else if (Input.GetMouseButtonUp(0))
        {
            isDraging = false;
            Reset();
        }
#endregion

#region
        if (Input.touches.Length > 0)
        {
            if (Input.touches[0].phase == TouchPhase.Began)
            {
                tap = true;
                isDraging = true;
                startTouch = Input.touches[0].position;
            }
            else if (Input.touches[0].phase == TouchPhase.Ended ||
Input.touches[0].phase == TouchPhase.Canceled)
            {
                isDraging = false;
                Reset();
            }
        }
#endregion

        //
        swipeDelta = Vector2.zero;
        if (isDraging)
        {
            if (Input.touches.Length < 0)
                swipeDelta = Input.touches[0].position - startTouch;
            else if (Input.GetMouseButton(0))
                swipeDelta = (Vector2)Input.mousePosition - startTouch;
        }

        //
        if (swipeDelta.magnitude > 100)
        {
            //
            float x = swipeDelta.x;
            float y = swipeDelta.y;
            if (Mathf.Abs(x) > Mathf.Abs(y))
            {

                if (x < 0)
                    swipeLeft = true;
                else
                    swipeRight = true;
            }
            else
            {

                if (y < 0)
                    swipeDown = true;
                else
                    swipeUp = true;
            }

            Reset();
        }
```

```
    }

    private void Reset()
    {
        startTouch = swipeDelta = Vector2.zero;
        isDraging = false;
    }
}



using System.Collections;
using System.Linq;
using Scenes.Quiz.Prefabs.Scripts;
using TMPro;
using Unity.VisualScripting;
using UnityEngine;
using UnityEngine.Audio;

public class Portal : MonoBehaviour
{
    [SerializeField] public int levelIndex;
    [SerializeField] private Vector2 Ranges;
    [SerializeField] private TileGeneration tileGeneration;
    [SerializeField] private TMP_Text theameText;
    [SerializeField] private AudioSource  audioSource ;
    private bool portalEnter;
    private Animator irbisPlayer;
    private Animator boyPlayer;

    private void Start()
    {
        levelIndex = RandomThemePicker.GetRandomThemeId();

        irbisPlayer = GlobalVar.irbisPlayer;
        boyPlayer = GlobalVar.boyPlayer;


        GameObject groundTileObj = GameObject.Find("GroundTileGeneration");
        if (groundTileObj != null)
        {
            tileGeneration = groundTileObj.GetComponent<TileGeneration>();
            if (tileGeneration == null)
            {
                Debug.LogError("TileGeneration ########## ##
####### ## GroundTileGeneration.");
            }
        }
        else
        {
            Debug.LogError("####### 'GroundTileGeneration' ##
####### # ######.");
        }
        if (GlobalVar.allThemes != null)
        {
            //yield return new WaitForSeconds(0.2f);

            StartCoroutine(StartDelay());
        }
    }


    private void Update()
    {
        if (Vector3.Distance(gameObject.transform.position,
irbisPlayer.transform.position) < 50 && !portalEnter && !GlobalVar.isSpeedUp)
        {
            GlobalVar.irbisPlayer.SetFloat("RunSpeed", 0.6f);
            GlobalVar.boyPlayer.SetFloat("RunSpeed", 0.6f);
            boyPlayer.SetFloat("RunSpeed", 0.6f);
            irbisPlayer.SetFloat("RunSpeed", 0.6f);
            //GlobalVar.moveSpeed = GlobalVar.slowSpeed;
        }

    }

    //####### ####### ### ### ####### ##
######### ############. ######## #######
#########
    private IEnumerator StartDelay()
    {
        yield return new WaitForSeconds(0.3f);
        if (GlobalVar.allThemes != null)
        {
            string theame= GlobalVar.allThemes.SingleOrDefault(a => a.Id ==
levelIndex).RussianWord;
            theameText.text = theame;
        }
    }

    private void OnTriggerEnter(Collider other)
    {
        if (other.CompareTag("Player"))
        {
            GlobalVar.levelType = levelIndex;
            GlobalVar.needWordReload = true;
            GlobalVar.landscapeType = Random.Range(1, 5);

            //=============##########
############===============
            portalEnter = true;
            GlobalVar.irbisPlayer.SetFloat("RunSpeed", 1.0f);
            irbisPlayer.SetFloat("RunSpeed", 1.0f);

            GlobalVar.boyPlayer.SetFloat("RunSpeed", 1.0f);
            boyPlayer.SetFloat("RunSpeed", 1.0f);
            if (!GlobalVar.isSpeedUp) GlobalVar.moveSpeed =
GlobalVar.deffSpeed;
            //=============================================
            audioSource.Play();
            tileGeneration.ResetWordState();
        }
    }
}



using System.Collections.Generic;
using UnityEngine;

[System.Serializable]
public class QuizData
{
    public List<Topic> topics;
}

[System.Serializable]
public class Topic
{
    public string name;
    public List<Question> questions;
}

[System.Serializable]
public class Question
{
    public string question;
    public List<string> answers;
    public int correctIndex;
}



using System.Collections.Generic;
using System.IO;
using TMPro;
using UnityEngine;
using UnityEngine.UI;

public class QuizLoader : MonoBehaviour
{
    [SerializeField] private QuizData quizData;
    [SerializeField] private List<Button> buttons;
    private Question currentQuestion;

    void Start()
    {
        string path = Path.Combine(Application.streamingAssetsPath, "quiz.json");
        if (File.Exists(path))
        {
            string json = File.ReadAllText(path);
            quizData = JsonUtility.FromJson<QuizData>(json);
            Debug.Log(" : " + quizData.topics.Count);
        }
        else
        {
            Debug.LogError("   !");
        }
    }

    public void OnAnswerSelected(int index)
    {
        bool isCorrect = index == currentQuestion.correctIndex;

        // ,  :
        buttons[index].GetComponent<Image>().color = isCorrect ? Color.green :
Color.red;
```

```
        //   ,
        if (!isCorrect)
        {
            buttons[currentQuestion.correctIndex].GetComponent<Image>().color =
Color.green;
        }

        //
    }


}


using System.Collections;
using System.Collections.Generic;
using TMPro;
using UnityEngine;
using UnityEngine.UI;

public class QuizManager : MonoBehaviour
{
    [SerializeField] private GameObject quizPanel;
    [SerializeField] private TextMeshProUGUI questionText;
    [SerializeField] private List<Button> answerButtons;

    private List<Question> allQuestions;      //
    private List<Question> remainingQuestions; //

    private List<Question> currentQuestions;
    //private int currentQuestionIndex = 0;
    private Question currentQuestion;

    private string selectedTopic;
    private QuizData quizData;

    public void Initialize(string topic, QuizData data)
    {
        selectedTopic = topic;
        quizData = data;
        quizPanel.SetActive(true);

        allQuestions = quizData.topics.Find(t => t.name ==
selectedTopic).questions;
        remainingQuestions = new List<Question>(allQuestions); //
        ShowQuestion();
    }

    private void ShowQuestion()
    {
        if (remainingQuestions.Count == 0)
        {
            //
            remainingQuestions = new List<Question>(allQuestions);
            Debug.Log("  . .");
        }

        int randomIndex = Random.Range(0, remainingQuestions.Count);
        currentQuestion = remainingQuestions[randomIndex];
        remainingQuestions.RemoveAt(randomIndex); //

        questionText.text = currentQuestion.question;

        for (int i = 0; i < answerButtons.Count; i++)
        {
            int index = i;
            answerButtons[i].interactable = true;
            answerButtons[i].GetComponent<Image>().color = Color.white;
            answerButtons[i].GetComponentInChildren<TextMeshProUGUI>().text =
currentQuestion.answers[i];
            answerButtons[i].onClick.RemoveAllListeners();
            answerButtons[i].onClick.AddListener(() => OnAnswerSelected(index));
        }
    }

    private void OnAnswerSelected(int index)
    {
        for (int i = 0; i < answerButtons.Count; i++)
        {
            answerButtons[i].interactable = false;
        }

        bool isCorrect = index == currentQuestion.correctIndex;

        answerButtons[index].GetComponent<Image>().color = isCorrect ?
Color.green : Color.red;

        if (!isCorrect)
        {
            answerButtons[currentQuestion.correctIndex].GetComponent<Image>().color =
Color.green;
        }

        StartCoroutine(NextQuestionDelayed());
    }

    private IEnumerator NextQuestionDelayed()
    {
        yield return new WaitForSeconds(2f);
        //currentQuestionIndex++;
        ShowQuestion();
    }
}


using System.Collections;
using System.Collections.Generic;
using System.IO;
using TMPro;
using UnityEngine;
using UnityEngine.UI;

public class ThemeRoulette : MonoBehaviour
{
    [SerializeField] private QuizManager quizManager;
    [SerializeField] private Button btnGenerate;
    [SerializeField] private Button btnOk;
    [SerializeField] private TextMeshProUGUI txtThemeName;

    private List<string> topicNames = new();
    private string selectedTopic;
    private QuizData quizData;

    private void Start()
    {
        btnOk.interactable = false;
        btnGenerate.onClick.AddListener(StartRoulette);
        btnOk.onClick.AddListener(OnThemeSelected);

        LoadQuizData();
    }

    private void LoadQuizData()
    {
        string path = Path.Combine(Application.streamingAssetsPath, "quiz.json");

        if (File.Exists(path))
        {
            string json = File.ReadAllText(path);
            quizData = JsonUtility.FromJson<QuizData>(json);

            foreach (Topic topic in quizData.topics)
            {
                topicNames.Add(topic.name);
            }
        }
        else
        {
            Debug.LogError("quiz.json    StreamingAssets");
        }
    }

    private void StartRoulette()
    {
        if (topicNames.Count == 0) return;

        btnOk.interactable = false;
        btnGenerate.interactable = false; //
        StartCoroutine(RouletteCoroutine());
    }

    private IEnumerator RouletteCoroutine()
    {
        float delay = 0.05f;
        int spins = Random.Range(20, 40);

        for (int i = 0; i < spins; i++)
        {
            int index = Random.Range(0, topicNames.Count);
            txtThemeName.text = topicNames[index];

            yield return new WaitForSeconds(delay);
            delay *= 1.05f;
        }

        selectedTopic = txtThemeName.text;
        btnOk.interactable = true;
```

```
        btnGenerate.interactable = true; //     ,
    }

    private void OnThemeSelected()
    {
        //Debug.Log(" : " + selectedTopic);
        //
        quizManager.Initialize(selectedTopic, quizData);
    }

    public string GetSelectedTopic() => selectedTopic;
}



using System;
using UnityEngine;

namespace Scenes.Object.Bonus_System.Coin
{
    public class BoosterRotate : MonoBehaviour
    {
        [SerializeField] private Vector3 rotationSpeed;


        private void Update()
        {
            transform.rotation = Quaternion.Euler(transform.rotation.eulerAngles +
rotationSpeed * Time.deltaTime);
        }
    }
}


using System;
using System.Collections;
using UnityEngine;
using UnityEngine.Audio;

public class Coin : MonoBehaviour
{
    [SerializeField] private AudioSource audioSource;
    [SerializeField] private MeshRenderer meshRenderer;
    [SerializeField] private GameObject visuals;
    [SerializeField] private Collider  col;
    [SerializeField] private AudioMixer mixer;
    [SerializeField] private float rotationSpeed;
    [SerializeField] private float distanceMult;
    [SerializeField] protected GameObject effect;

    [Header("Звук монеток")]
    [Space]
    [SerializeField] private float maxPitch;
    [SerializeField] private float pitchStep;
    [SerializeField] private float minPitch;
    [Space]
    [SerializeField] private float pitchResetTime;


    void Update()
    {
        var y = ((Time.time + transform.position.z * distanceMult)%360f) *
rotationSpeed;

        visuals.transform.rotation = Quaternion.Euler(0, y, 0);
        if (GlobalVar.autoDestroy) gameObject.SetActive(false);

        effect.SetActive(GlobalVar.isGoldenAsyq);
    }

    private void UpdatePitch(float pitch)
    {
        mixer.SetFloat("CoinPitch", pitch);
    }

    private void OnTriggerEnter(Collider other)
    {
        if (other.CompareTag("Player"))
        {
            if(GlobalVar.isGoldenAsyq) UpdatePitch(maxPitch);
            else
            {
                if (Time.time - GlobalVar.LastCoinTime > pitchResetTime)
                {
                    GlobalVar.CoinPitch = minPitch;
                    UpdatePitch(minPitch);
                }
                else
                {
                    if (GlobalVar.CoinPitch < maxPitch) GlobalVar.CoinPitch +=
pitchStep;
                    UpdatePitch(GlobalVar.CoinPitch);
                }
            }



            audioSource.Play();
            meshRenderer.enabled = false;
            col.enabled = false;

            GlobalVar.LastCoinTime = Time.time;
            StartCoroutine(DisableAfterSound());
        }
    }

    private IEnumerator DisableAfterSound()
    {
        yield return new WaitForSeconds(audioSource.clip.length);
        gameObject.SetActive(false);
    }
}



using System.Collections;
using DG.Tweening;
using UnityEngine;

public class CoinFlyEffect : MonoBehaviour
{
    [SerializeField] private RectTransform coinTargetIcon;
    [SerializeField] private GameObject coinFlyPrefab;
    [SerializeField] private Canvas canvas;
    [SerializeField] private Camera mainCamera;
    [SerializeField] float duration = 0.2f;
    [SerializeField] private SpriteRenderer coinParticle;
    public void AnimateCoin(Vector3 worldPosition)
    {
        DoAnimation();

        return;
        // ******** ********** ** ***** * ******
        Vector3 screenPos = mainCamera.WorldToScreenPoint(worldPosition);

        // ******** ****** *******
        GameObject coinObj = Instantiate(coinFlyPrefab, canvas.transform);
        coinObj.transform.position = screenPos;

        StartCoroutine(FlyToTarget(coinObj.GetComponent<RectTransform>()));
    }


    private void DoAnimation()
    {
        var rot = coinParticle.transform.rotation;
        coinParticle.transform.Rotate(Vector3.forward, Random.Range(0f, 360f));

        var oldColor = coinParticle.color;
        coinParticle.color = new Color(oldColor.r, oldColor.g, oldColor.b, 0.0f);

        coinParticle.DOFade(0f, 0.4f);

        var seq = DOTween.Sequence();

        seq.Append(coinParticle.DOFade(0.4f, 0.1f));
        seq.Join(coinParticle.transform.DOScale(Vector3.one * 1.4f, 0.1f));
        seq.Append(coinParticle.DOFade(0f, 0.4f));
        seq.Join(coinParticle.transform.DOScale(Vector3.one * 1.1f, 0.1f));
    }

    private IEnumerator FlyToTarget(RectTransform coin)
    {

        float time = 0f;

        Vector3 startPos = coin.position;
        Vector3 endPos = coinTargetIcon.position;

        while (time < duration)
        {
            time += Time.deltaTime;
            float t = time / duration;

            // ******** ******* ********** (******
********** ** Curve)
            coin.position = Vector3.Lerp(startPos, endPos, t);
            yield return null;
```

```
        }

        Destroy(coin.gameObject); // ******** ******** ******
**********
    }

    public void PlayAnimation()
    {
        DoAnimation();
    }
}



using System;
using UnityEngine;

namespace Scenes.Object.Bonus_System.Coin
{
    public class ScaleAnimation : MonoBehaviour
    {
        [SerializeField] private float maxScale;
        [SerializeField] private float minScale;

        [SerializeField] private float speed;

        private void Update()
        {
            // Animate the local scale between minScale and maxScale using a
ping-pong over time
            float low = Mathf.Min(minScale, maxScale);
            float high = Mathf.Max(minScale, maxScale);
            float range = high - low;
            float value = low + Mathf.PingPong(Time.time * Mathf.Max(speed, 0f),
range);
            transform.localScale = Vector3.one * value;
        }
    }
}


using System.Collections;
using TMPro;
using UnityEngine;

public class FinePopup : MonoBehaviour
{
    [SerializeField] private TMP_Text text;
    [SerializeField] private CanvasGroup canvasGroup;

    public void Show(string message, Vector2 startAnchoredPos, float rise = 80f,
float duration = 0.8f)
    {
        var rt = (RectTransform)transform;
        rt.anchoredPosition = startAnchoredPos;
        if (text != null) text.text = message;
        StartCoroutine(Play(rt, rise, duration));
    }

    private IEnumerator Play(RectTransform rt, float rise, float duration)
    {
        float t = 0f;
        Vector2 start = rt.anchoredPosition;
        Vector2 end = start + new Vector2(0f, rise);

        while (t < duration)
        {
            t += Time.deltaTime;
            float k = t / duration;
            //
            rt.anchoredPosition = Vector2.Lerp(start, end, k);
            canvasGroup.alpha = 1f - k;
            yield return null;
        }
        Destroy(gameObject);
    }
}



using System;
using System.Collections;
using Scenes.UI.Scripts;
using TMPro;
using UnityEngine;
using UnityEngine.UI;
/// Управление бустерами
public class Busters : MonoBehaviour
{
    [Header("ДурМашина")]
    [SerializeField]
    private GameObject player;
    private Renderer playerRender;
    private Material defaultMatPlayer;

    [Header("Магнит")]
    public float attractionRadius = 10f;
    public float attractionSpeed = 40f;
    public GameObject targetMagnit;


    [Header("Скорость")]
    public float speedMultiplier = 1.3f;

    public TileGeneration tileGen;
    public GameObject buster;

    [Header("Интерфейс")]
    [SerializeField] private BoosterUI  boosterUI;

    private Coroutine busterCoroutine;

    [Header("Иконки бустера")]
    [SerializeField] private Sprite icoSpeedUp;
    [SerializeField] private Sprite icoFrost;
    [SerializeField] private Sprite icoMagnetto;
    [SerializeField] private Sprite icoAsyq;
    [SerializeField] private Sprite icoShield;

    [Header("Звуки")]
    [SerializeField] AudioSource audioSource;

    [Header("Бустер Эффект")]
    [SerializeField] GameObject busterShield;
    [SerializeField] GameObject busterSpeed;
    [SerializeField] GameObject busterMagnet;
    [SerializeField] GameObject busterImunno;
    [SerializeField] GameObject busterGoldenAsyq;

    [Header("Бустер Материалы")]
    [SerializeField] Material materialSpeed;
    [SerializeField] Material materialImunno;
    [SerializeField] Material materialMagnetto;
    [SerializeField] Material materialGoldenAsyq;

    private Renderer _renderer;

    private Coroutine magnetCoroutine;
    private Coroutine speedCoroutine;
    private Coroutine invulnerabilityCoroutine;
    private Coroutine imunnoRoutine;
    private Coroutine goldenAsyqRoutine;

    private enum Booster
    {
        none,
        magnet,
        speed,
        invulnerability,
        imunno,
        goldenAsyq
    }

    private Booster currentBooster;
    public Material GetCurrentBusterMaterial()
    {
        return currentBooster switch
        {
            Booster.none => defaultMatPlayer,
            Booster.magnet => materialMagnetto,
            Booster.speed => materialSpeed,
            Booster.invulnerability => defaultMatPlayer,
            Booster.imunno => materialImunno,
            Booster.goldenAsyq => materialGoldenAsyq,
            _ => defaultMatPlayer
        };
    }

    private bool isInvulnerable = false;
    float originalSpeed;
    public bool IsInvulnerable()
    {
        return isInvulnerable;
    }
    private void Start()
    {
        _renderer = busterShield.GetComponent<Renderer>();
        playerRender = player.GetComponent<Renderer>();
```

```
    defaultMatPlayer = playerRender.material;
    currentBooster = Booster.none;
}

/// Активация примагничивания монет
public void ActivateMagnet(float duration)
{
    if (magnetCoroutine != null) StopCoroutine(magnetCoroutine);
    currentBooster = Booster.magnet;
    boosterUI.SetBoosterSprite(icoMagnetto);
    busterMagnet.SetActive(true);
    playerRender.material = materialMagnetto;
    audioSource.Play();
    magnetCoroutine = StartCoroutine(MagnetRoutine(duration));
}

/// Активация увеличения скорости
public void ActivateSpeed(float duration)
{
    if (speedCoroutine != null) StopCoroutine(speedCoroutine);
    currentBooster = Booster.speed;
    audioSource.Play();
    boosterUI.SetBoosterSprite(icoSpeedUp);
    busterSpeed.SetActive(true);
    GlobalVar.isSpeedUp = true;
    playerRender.material = materialSpeed;
    speedCoroutine = StartCoroutine(SpeedRoutine(duration));
}

/// Активация иммунитета к ошибкам
public void ActivateImmunity(float duration)
{
    if (imunnoRoutine != null) StopCoroutine(imunnoRoutine);

    currentBooster = Booster.imunno;
    busterImunno.SetActive(true);
    playerRender.material = materialImunno;
    boosterUI.SetBoosterSprite(icoFrost);
    audioSource.Play();
    imunnoRoutine=StartCoroutine(ImmunityRoutine(duration));
}

//Активация неуязвимости к препятствиям, npc и т.д.
public void ActivateInvulnerability(float duration)
{
    if (invulnerabilityCoroutine != null) StopCoroutine(invulnerabilityCoroutine);
    currentBooster = Booster.invulnerability;
    busterShield.SetActive(true);
    boosterUI.SetBoosterSprite(icoShield);
    audioSource.Play();
    StartCoroutine(InvulnerabilityActivation(duration));
}

/// Активация увеличенного сбора монет
public void ActivateGoldenAsyq(float duration)
{
    if (goldenAsyqRoutine != null) StopCoroutine(goldenAsyqRoutine);
    currentBooster = Booster.goldenAsyq;
    GlobalVar.isGoldenAsyq = true;
    busterGoldenAsyq.SetActive(true);
    GlobalVar.CoinAmount = 3;
    playerRender.material = materialGoldenAsyq;
    boosterUI.SetBoosterSprite(icoAsyq);
    audioSource.Play();
    goldenAsyqRoutine = StartCoroutine(GoldenAsyqRoutine(duration));
}

/// Сброс бустеров
public void ResetAllBoosters()
{
    currentBooster = Booster.none;
    // Остановить все активные корутины
    if (magnetCoroutine != null)
    {
        StopCoroutine(magnetCoroutine);
        magnetCoroutine = null;
    }

    if (speedCoroutine != null)
    {
        StopCoroutine(speedCoroutine);
        speedCoroutine = null;

    }

    if (invulnerabilityCoroutine != null)
    {
        StopCoroutine(invulnerabilityCoroutine);
        invulnerabilityCoroutine = null;
    }

    if (goldenAsyqRoutine != null)
    {
        StopCoroutine(goldenAsyqRoutine);
        goldenAsyqRoutine = null;
    }

    if (imunnoRoutine != null)
    {
        StopCoroutine(imunnoRoutine);
        imunnoRoutine = null;
    }

    isInvulnerable = false;
    boosterUI.ClearBoosterUI();

    if (busterCoroutine != null)
    {
        StopCoroutine(busterCoroutine);
        busterCoroutine = null;
    }


    //Imunno
    busterImunno.SetActive(false);
    playerRender.material = defaultMatPlayer;
    //Invulnerable
    _renderer.material.SetFloat("_Disolve", 1.0f);
    busterShield.SetActive(false);
    isInvulnerable = false;
    //Magnet
    playerRender.material = defaultMatPlayer;
    busterMagnet.SetActive(false);
    //Golden
    GlobalVar.isGoldenAsyq = false;
    busterGoldenAsyq.SetActive(false);
    playerRender.material = defaultMatPlayer;
    GlobalVar.CoinAmount = 1;
    //SpeedUp
    GlobalVar.moveSpeed = originalSpeed;
    GlobalVar.isSpeedUp = false;
    busterSpeed.SetActive(false);
    playerRender.material = defaultMatPlayer;



    Debug.Log("▼ Все бустеры сброшены после смерти");
}

private void AttractCoins()
{
    Collider[] coins = Physics.OverlapSphere(targetMagnit.transform.position,
attractionRadius);
    foreach (var col in coins)
    {
        if (col.CompareTag("Coin"))
        {
            Transform coin = col.transform;
            float distance = Vector3.Distance(coin.position,
targetMagnit.transform.position);

            if (distance > 0.5f) // ещё далеко — тянем
            {
                Vector3 dir = (targetMagnit.transform.position -
coin.position).normalized;
                coin.position += dir * (attractionSpeed * Time.deltaTime);
            }
        }
    }
}


#region Отсчет времени действия бустера
/// Отсчет времени на запускаемый бустер
private IEnumerator BusterTimer(float duration)
{
    float elapsed = 0f;
    GlobalVar.IsAnyBusterActive = true;
    while (elapsed < duration)
    {
        elapsed += Time.deltaTime;
        yield return null;
    }
    GlobalVar.IsAnyBusterActive = false ;
}

/// Бустер Иммунитет
private IEnumerator ImmunityRoutine(float duration)
```

```
    {
        tileGen.ActivateImmunity(duration);
        GlobalVar.isImunno = true;
        yield return new WaitForSeconds(duration);
        GlobalVar.isImunno = false;
        busterImunno.SetActive(false);
        playerRender.material = defaultMatPlayer;
        currentBooster = Booster.none;
    }
    /// Бустер Неуязвимость
    ///
    private IEnumerator InvulnerabilityActivation(float duration)
    {
        isInvulnerable = true;
        yield return new WaitForSeconds(duration);
        busterShield.SetActive(false);
        currentBooster = Booster.none;
        isInvulnerable = false;
    }

    /// Бустер Магнит
    private IEnumerator MagnetRoutine(float duration)
    {
        float elapsed = 0f;
        while (elapsed < duration)
        {
            AttractCoins();
            elapsed += Time.deltaTime;
            yield return null;
        }
        playerRender.material = defaultMatPlayer;
        busterMagnet.SetActive(false);
        currentBooster = Booster.none;
    }
    /// Бустер Скорость
    private IEnumerator SpeedRoutine(float duration)
    {
        originalSpeed = GlobalVar.moveSpeed;
        GlobalVar.moveSpeed *= speedMultiplier;
        yield return new WaitForSeconds(duration);
        GlobalVar.moveSpeed = originalSpeed;
        busterSpeed.SetActive(false);
        playerRender.material = defaultMatPlayer;
        currentBooster = Booster.none;
        GlobalVar.isSpeedUp = false;
    }
    /// Бустер 2X голды
    private IEnumerator GoldenAsyqRoutine(float duration)
    {
        yield return new WaitForSeconds(duration);
        GlobalVar.isGoldenAsyq = false;
        busterGoldenAsyq.SetActive(false);
        playerRender.material = defaultMatPlayer;
        GlobalVar.CoinAmount = 1;
        currentBooster = Booster.none;
    }
#endregion

    public void ActivateBusterUI(string boosterNane, float duration)
    {
        boosterUI.ActivateBoosterUI(boosterNane, duration);
        StartCoroutine(BusterTimer(duration));
    }
}
```

```
using UnityEngine;

public class GoldAsyq : MonoBehaviour
{
    void Update()
    {
        if (GlobalVar.IsAnyBusterActive) gameObject.SetActive(false);
    }

    private void OnTriggerEnter(Collider other)
    {
        if (other.CompareTag("Player"))
        {
            gameObject.SetActive(false);
        }
    }
}
```

```
using UnityEngine;

public class Immunity : MonoBehaviour
{

    void Update()
    {
        if (GlobalVar.IsAnyBusterActive) gameObject.SetActive(false);
    }


    private void OnTriggerEnter(Collider other)
    {
        if (other.CompareTag("Player"))
        {
            gameObject.SetActive(false);
        }
    }
}
```

```
using UnityEngine;
public class Magnetto : MonoBehaviour
{
    void Update()
    {
        if (GlobalVar.IsAnyBusterActive) gameObject.SetActive(false);
    }
    private void OnTriggerEnter(Collider other)
    {
        if (other.CompareTag("Player"))
        {
            gameObject.SetActive(false);
        }
    }
}
```

```
using UnityEngine;

public class Invulnerability : MonoBehaviour
{
    void Update()
    {
        if (GlobalVar.IsAnyBusterActive) gameObject.SetActive(false);
    }
    private void OnTriggerEnter(Collider other)
    {
        if (other.CompareTag("Player"))
        {
            gameObject.SetActive(false);
        }
    }
}
```

```
using UnityEngine;

public class SpeedUper : MonoBehaviour
{
    private GameObject portal;
    private void Start()
    {
        //PortalTheame
        portal = GameObject.Find("PortalTheame");
        //if (Vector3.Distance(gameObject.transform.position, transform.position) <
55) gameObject.SetActive(false);
    }

    void Update()
    {
        if (GlobalVar.IsAnyBusterActive) gameObject.SetActive(false);
    }

    private void OnTriggerEnter(Collider other)
    {
        if (other.CompareTag("Player"))
        {
            gameObject.SetActive(false);
        }
    }
}
```

```
using UnityEngine;
```

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
```

```
public class DissolveHeightSetter : MonoBehaviour
{
    Renderer _rend;
    // Start is called before the first frame update
    void Start()
    {
        _rend = GetComponent<Renderer>();
    }

    // Update is called once per frame
    void Update()
    {
        _rend.material.SetFloat("_DissolveStartHeight", transform.position.y);
    }
}
```

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class MagnifyingObject : MonoBehaviour
{
    Renderer _renderer;
    Camera _cam;
    void Start()
    {
        _renderer = GetComponent<Renderer>();
        _cam = Camera.main;
    }

    // Update is called once per frame
    void Update()
    {
        Vector3 screenPoint = _cam.WorldToScreenPoint(transform.position);
        screenPoint.x = screenPoint.x / Screen.width;
        screenPoint.y = screenPoint.y / Screen.height;
        _renderer.material.SetVector("_ObjScreenPos", screenPoint);
    }
}
```

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Projectile : MonoBehaviour
{
    [SerializeField] float _ProjectileSpeed;
    [SerializeField] GameObject _HitEffect;
    Vector3 _hitNormal;
    Vector3 _hitPos;
    Ray ray;
    RaycastHit hit;
    private void Start()
    {
        Destroy(gameObject, 3);
    }
    void Update()
    {
        Vector3 delta = (transform.forward * _ProjectileSpeed * Time.deltaTime);
        transform.position += delta;
        ray = new Ray(transform.position,transform.forward);
        if(Physics.Raycast(ray,out hit))
        {
            _hitNormal = hit.normal;
            _hitPos = hit.point;
            float distance = (transform.position - hit.point).magnitude;
            if(distance< delta.magnitude)
            {
                Hit(hit.collider);
            }
        }
    }

    void Hit(Collider collider)
    {
        GameObject hit = Instantiate(_HitEffect, _hitPos, Quaternion.identity);
        Shield shield = collider.GetComponentInParent<Shield>();
        if (shield != null)
        {
            shield.HitShield(_hitPos);
        }
        hit.transform.forward = _hitNormal;
        Destroy(gameObject);
    }
}
```

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class ProjectileController : MonoBehaviour
{
    [SerializeField] Projectile _Projectile;
    [SerializeField] List<Transform> _ProjectilePoints;
    private void Update()
    {
        if (Input.GetMouseButtonDown(0))
        {
            Ray ray = Camera.main.ScreenPointToRay(Input.mousePosition);
            RaycastHit hit;
            if(Physics.Raycast(ray,out hit))
            {
                Transform point = _ProjectilePoints[Random.Range(0,
_ProjectilePoints.Count)];
                Projectile projectile = Instantiate(_Projectile, point.position,
Quaternion.identity);
                projectile.transform.forward = (hit.point - point.position).normalized;
            }
        }
    }
}
```

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Shield : MonoBehaviour
{
    Renderer _renderer;
    [SerializeField] AnimationCurve _DisplacementCurve;
    [SerializeField] float _DisplacementMagnitude;
    [SerializeField] float _LerpSpeed;
    [SerializeField] float _DisolveSpeed;
    bool _shieldOn;
    Coroutine _disolveCoroutine;
    // Start is called before the first frame update
    void Start()
    {
        _renderer = GetComponent<Renderer>();
    }

    // Update is called once per frame
    void Update()
    {
        if (Input.GetMouseButtonDown(0))
        {
            Ray ray = Camera.main.ScreenPointToRay(Input.mousePosition);
            RaycastHit hit;
            if (Physics.Raycast(ray, out hit))
            {
                HitShield(hit.point);
            }
        }
        if (Input.GetKeyDown(KeyCode.F))
        {
            OpenCloseShield();
        }
    }

    public void HitShield(Vector3 hitPos)
    {
        _renderer.material.SetVector("_HitPos", hitPos);
        StopAllCoroutines();
        StartCoroutine(Coroutine_HitDisplacement());
    }

    public void OpenCloseShield()
    {
        float target = 1;
        if (_shieldOn)
        {
            target = 0;
        }
        _shieldOn = !_shieldOn;
        if (_disolveCoroutine != null)
        {
            StopCoroutine(_disolveCoroutine);
        }
        _disolveCoroutine = StartCoroutine(Coroutine_DisolveShield(target));
```

```
    }

    IEnumerator Coroutine_HitDisplacement()
    {
        float lerp = 0;
        while (lerp < 1)
        {
            _renderer.material.SetFloat("_DisplacementStrength",
_DisplacementCurve.Evaluate(lerp) * _DisplacementMagnitude);
            lerp += Time.deltaTime*_LerpSpeed;
            yield return null;
        }
    }

    IEnumerator Coroutine_DisolveShield(float target)
    {
        float start = _renderer.material.GetFloat("_Disolve");
        float lerp = 0;
        while (lerp < 1)
        {
            _renderer.material.SetFloat("_Disolve", Mathf.Lerp(start,target,lerp));
            lerp += Time.deltaTime * _DisolveSpeed;
            yield return null;
        }
    }
}



using UnityEngine;
using UnityEngine.Audio;

public class Obstacles : MonoBehaviour
{

    void Update()
    {
        //if (GlobalVar.autoDestroy) Destroy(gameObject);


    }




    private void OnTriggerEnter(Collider other)
    {
        //if (GlobalVar.isSheld) Destroy(gameObject);


        //if (other.CompareTag("LetterTower")) Destroy(gameObject);
        //if (other.CompareTag("Portal")) Destroy(gameObject);
        //if (other.CompareTag("Imunno")) Destroy(gameObject);
        //if (other.CompareTag("SpeedUp")) Destroy(gameObject);
        //if (other.CompareTag("Magnetto")) Destroy(gameObject);

    }

}



using System.Collections;
using System.Collections.Generic;
using System.Linq;
using UnityEngine;

public class ObjectGeneration : MonoBehaviour
{
    [SerializeField]
    private bool doIncrementalGeneration;
    [SerializeField]
    private int incrementalBatchSize;

    [SerializeField] private ObstacleGeneration obstacleGeneration;
    [SerializeField] private Transform wildObjectsParent;
    [SerializeField] public Transform letterParent;
    [SerializeField] Transform[] objectLocation;
    [SerializeField] GameObject[] wildObjects;
    [Header("ПОРТАЛЫ")]
    [SerializeField] GameObject portals;

    [SerializeField] private bool isPoolable;

    public int tileType;

    void Start()
    {
        if(isPoolable) return;
        Initialize();
    }

    public void Initialize()
    {
        if (wildObjects.Length == 0 || objectLocation.Length == 0) return;

        if (doIncrementalGeneration) StartCoroutine(IncrementalGeneration());
        else
        {
            foreach (var point in objectLocation)
            {
                var prefab = wildObjects[Random.Range(0, wildObjects.Length)];
                Instantiate(prefab, point.position, point.rotation, transform);
            }
        }

        obstacleGeneration.Generate();
    }

    public void SpawnPortal()
    {
        DisplayPortals(transform.position + new Vector3(0f, 1f, 70f));
    }

    private IEnumerator IncrementalGeneration()
    {
        var batchCount = 0;
        if (wildObjectsParent.childCount > 0)
        {
            var offset = Random.Range(0, wildObjectsParent.childCount);
            for (int i = 0; i < wildObjectsParent.childCount; i++)
            {
                var child = wildObjectsParent.GetChild(i);
                var pointIndex = (i + offset) % wildObjectsParent.childCount;

                var point = objectLocation[pointIndex];
                child.position = point.position;
                child.rotation = point.rotation;
                batchCount++;

                if (batchCount >= incrementalBatchSize)
                {
                    batchCount = 0;
                    yield return new WaitForEndOfFrame();
                }
            }
        }
        else
        {
            foreach (var point in objectLocation)
            {
                var prefab = wildObjects[Random.Range(0, wildObjects.Length)];
                Instantiate(prefab, point.position, point.rotation, wildObjectsParent);
                batchCount++;

                if (batchCount >= incrementalBatchSize)
                {
                    batchCount = 0;
                    yield return new WaitForEndOfFrame();
                }
            }
        }
    }
    private void DisplayPortals(Vector3 basePosition)
    {
        if (portals == null) return;
        Instantiate(portals, basePosition, Quaternion.identity);
    }

    public void StartPoolReturn()
    {
        for (int i = 0; i < letterParent.childCount; i++)
        {
            Destroy(letterParent.GetChild(i).gameObject);
        }

        obstacleGeneration.DestroyObstacles();

        gameObject.SetActive(false);
    }

    public void StartIncrementalDestroy()
    {
        StartCoroutine(IncrementalDestroy());
    }

    private IEnumerator IncrementalDestroy()
    {
```

```
        var childCount = transform.childCount;

        var batchCount = 0;
        while (childCount > 0)
        {
            Destroy(transform.GetChild(childCount-1).gameObject);
            childCount--;
            batchCount++;

            if (batchCount >= incrementalBatchSize)
            {
                yield return new WaitForEndOfFrame();
                batchCount = 0;
            }
        }

        if (!isPoolable) Destroy(gameObject);
        else gameObject.SetActive(false);
    }
}



using System.Collections.Generic;
using System.Linq;
using UnityEngine;

public class ObstacleGeneration : MonoBehaviour
{
    //==================== ПРЕПЯТСТВИЯ И МОНЕТЫ
====================
    [Header("ПРЕПЯТСТВИЯ")]
    [SerializeField] private GameObject[] obstaclePrefabs;
    [SerializeField] private int minObstacles = 2;
    [SerializeField] private int maxObstacles = 4;

    [Header("МОНЕТЫ")]
    [SerializeField] private GameObject coinPrefab;
    [SerializeField] private int minCoins = 5;
    [SerializeField] private int maxCoins = 10;

    //=================== БУСТЕРЫ ==================
    [Header("БУСТЕРЫ")]
    [SerializeField] private GameObject[] busterPrefabs;
    //[SerializeField] private float busterSpawnChance = 0.3f;

    [Header("Настройки генерации")]
    [SerializeField] private float[] laneX = { -3f, 0f, 3f };
    [SerializeField] private float[] zOffsets = { -50f, -40f, -30f, -20f, -10f, 0f, 10f,
20f, 30f, 40f, 50f };
    [SerializeField] private float tileLength = 140f;

    [SerializeField] private Transform container;

    public virtual void DestroyObstacles()
    {
        foreach (Transform child in container.transform)
        {
            Destroy(child.gameObject);
        }
    }

    public virtual void Generate()
    {
        Transform tile = transform;
        container.transform.parent = tile;
        container.transform.localPosition = Vector3.zero;

        float baseZ = tile.position.z;
        float platformY = tile.position.y;
        float maxObstacleZ = baseZ + tileLength - 10f;

        List<Vector3> availablePositions = new();

        foreach (float x in laneX)
        {
            foreach (float zOffset in zOffsets)
            {
                float globalZ = baseZ + zOffset;
                if (globalZ <= maxObstacleZ)
                {
                    availablePositions.Add(new Vector3(x, platformY, globalZ));
                }
            }
        }

        availablePositions = availablePositions.OrderBy(_ =>
Random.value).ToList();

        // Препятствия
        int obstacleCount = Random.Range(minObstacles, maxObstacles + 1);
        var obstaclePositions = availablePositions.Take(obstacleCount).ToList();
        foreach (var pos in obstaclePositions)
        {
            var prefab = obstaclePrefabs[Random.Range(0,
obstaclePrefabs.Length)];
            Instantiate(prefab, pos, Quaternion.identity, container.transform);
        }
        //Монеты
        var coinPositions = availablePositions.Skip(obstacleCount).ToList();
        int coinCount = Mathf.Min(Random.Range(minCoins, maxCoins + 1),
coinPositions.Count);
        for (int i = 0; i < coinCount; i++)
        {
            Vector3 coinPos = coinPositions[i] + Vector3.up * 1.1f;
            Instantiate(coinPrefab, coinPos, Quaternion.Euler(0f, 90f, 0f),
container.transform);
        }

        // Бустер
        if (busterPrefabs.Length > 0 && Random.value <
GlobalVar.busterSpawnChance &&  !GlobalVar.IsAnyBusterActive)
        {
            var busterPosCandidates = coinPositions.Skip(coinCount).ToList();
            if (busterPosCandidates.Count > 0)
            {
                Vector3 spawnPos = busterPosCandidates[0] + Vector3.up * 1.1f;
                var buster = busterPrefabs[Random.Range(0, busterPrefabs.Length)];
                Instantiate(buster, spawnPos, Quaternion.identity,
container.transform);
            }
        }
    }
}



using System.Collections.Generic;
using System.Linq;
using UnityEngine;

public class ObstacleGenerationDud : ObstacleGeneration
{
    public override void DestroyObstacles()
    {

    }

    public override void Generate()
    {

    }
}



using System;
using System.Collections;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Threading.Tasks;
using Quiz.Scripts;
using Scenes.GameScript;
using Scenes.Object.Nature.Ground.Scripts;
using Scenes.Object.UI.Translater;
using Scenes.UI.Scripts;
using TMPro;
using UnityEngine;
using UnityEngine.UI;
using Random = UnityEngine.Random;

public class TileGeneration : MonoBehaviour
{
    [SerializeField] private Transform player;
    private PlayerController playerController;
    [SerializeField] private TutorialController tutorialController;
    //private Busters playerBusters;

    [Header("Озвучка букв и слов")]
    [SerializeField] private AudioSource audioSource;
    [SerializeField] private AudioClip[] audioClip;


    [Header("Собираемые слова")]
    [SerializeField] public int correctNumWords = 1;

    [Header("ПЛАТФОРМЫ")]
```

```
[SerializeField] GameObject[] tilePrafabs;

[SerializeField] private TilePool[] pools;


private int startTiles = 4;
private float spawnPos = -50;
private float titleLenght = 140f;
private List<GameObject> activeTile = new();

[Header("СЛОВА")]
[SerializeField] private GameObject letter3DPrefab;
[SerializeField] private float letterSpawnChance = 0.3f;
[SerializeField] private CollectWordUIController  collectWordUIController;
[SerializeField] private TileUIController tileUIController;

private WordData currentWord;
private List<WordData> wordDictionary;
private bool portalQueued = false;

public char ExpectedLetter
{
    get
    {
        return GlobalVar.difficulty switch
        {
            0 when currentLetterIndex >= remainingLetters.Count => ' ',
            0 => remainingLetters[currentLetterIndex],
            1 => correctMissingLetter,
            2 => hiddenKazakhWord[currentLetterIndex],
            _ => ' '
        };
    }
}

private List<char> allKazakhLetters = new()
{
    'а', 'ә', 'б',  'г', 'ғ', 'д', 'е',  'ж', 'з', 'и',
    'й', 'к', 'қ', 'л', 'м', 'н', 'ң', 'о', 'ө', 'п', 'р', 'с',
    'т', 'у', 'ұ', 'ү', 'х', 'һ', 'ш', 'щ', 'ы', 'і', 'ю', 'я'
};
private string CleanWord(string word)
{
    return word.Replace(" ", "")
               .Replace("?", "")
               .Replace("!", "")
               .Replace(".", "")
               .Replace(",", "");
}
private List<char> collectedLetters = new();
private List<char> remainingLetters = new();
private List<Vector3> letterPositions = new();
private bool isFirstWord = true;
private int missingLetterIndex = -1;
private char correctMissingLetter;
private string hiddenKazakhWord = "";
private int currentLetterIndex = 0;
private bool isImmune = false;
private Coroutine immunityCoroutine;

void Start()
{
    LoadWordsFromJSON();
    LoadThemesFromJson();
    GlobalVar.CorrectNumWords = correctNumWords;

    for (var i = 0; i < pools.Length; i++)
    {
        var pool = pools[i];
        var pref = tilePrafabs[i];
        pool.InitPool(pref);
    }

    playerController = player.GetComponent<PlayerController>();
    if (correctNumWords > 0)
    {
        switch (GlobalVar.difficulty)
        {
            case 0: InitEasyMode(); break;
            case 1: InitMediumMode(); break;
            case 2: InitHardMode(); break;
        }
    }

    for (int i = 0; i < startTiles; i++)
    {
        int index = (i == 0) ? 4 : 0;
        if (tutorialController.IsTutorialActive && i == 1)
        {
            index = 5;
        }
        if (tutorialController.IsTutorialActive && i == 2)
        {
            index = 6;
        }
        SpawnTile(index);
    }
}
void Update()
{
    if (player.position.z - 200 > spawnPos - (startTiles * titleLenght))
    {
        float pp = spawnPos - (startTiles * titleLenght);
        float tt = player.position.z - 210;


        SpawnTile(GlobalVar.landscapeType - 1);
        DeleteTile();
    }

    if (GlobalVar.needWordReload)
    {
        GlobalVar.needWordReload = false;
        if (GlobalVar.levelType != 0) StartCoroutine(GenerateNextWord());

        collectWordUIController.RefreshWordDisplay();
    }
}


#region СЛУЖЕБНЫЕ
//Загружаем словарь
void LoadWordsFromJSON()
{
    wordDictionary = FileLoader.LoadWordsFromJSON();
}
//Загружаем темы
private void LoadThemesFromJson()
{
    GlobalVar.allThemes = FileLoader.LoadThemesFromJson();
}

//Тащим словарь по типу выбранного раздела
private List<WordData> GetFilteredWords()
{
    if (GlobalVar.levelType == 0) return new();
    var words = wordDictionary.Where(w => w.CategoryId ==
GlobalVar.levelType).ToList();
    //Фильтровать старое слово, чтобы оно не появилось дважды
    if (words.Contains(currentWord)) words.Remove(currentWord);

    return words;
}
//Сброс состояния слова
public void ResetWordState()
{
    correctNumWords = GlobalVar.CorrectNumWords;
    collectedLetters.Clear();
    remainingLetters.Clear();
    GlobalVar.portalsSpawned = false;
    GlobalVar.shouldSpawnPortals = false;
    isFirstWord = true;
}
#endregion


//==================== ГЕНЕРАЦИЯ ПЛАТФОРМ
================
private void SpawnTile(int tileIndex)
{
    try
    {
        if (tileIndex < 0) return;

        // Отображение названия локации

tileUIController.UpdateTileName(GetLandscapeName(GlobalVar.landscapeTyp
e));



        // Создание новой платформы
        GameObject newTile;

        if(tileIndex < pools.Length) newTile = pools[tileIndex].GetFromPool(new
Vector3(0f, 3.9f, spawnPos), Quaternion.identity);
        else newTile = Instantiate(tilePrafabs[tileIndex], new Vector3(0f, 3.9f,
spawnPos), Quaternion.identity);
```

```
            var objectGeneration = newTile.GetComponent<ObjectGeneration>();

            objectGeneration.tileType = tileIndex;

            if (portalQueued)
            {
                objectGeneration.SpawnPortal();
                portalQueued = false;
            }

            activeTile.Add(newTile);

            Vector3 spawnPoint = newTile.transform.position + new Vector3(0f, 1.2f,
titleLenght / 3f);

            // ОЧИСТКА и ГЕНЕРАЦИЯ букв
            letterPositions.Clear();
            if (tileIndex < 4 && currentWord != null)
            {
                TrySpawnLettersOnTile(objectGeneration.letterParent,
newTile.transform.position + new Vector3(0f, 1f, titleLenght / 2f));
            }

            // Смещение позиции спавна для следующей платформы
            spawnPos += titleLenght;


        }
        catch (System.Exception e)
        {
            Debug.LogError($"[TileGeneration] Ошибка при спавне тайла:
{e.Message}");
        }
    }
    private string GetLandscapeName(int type)
    {
        return type switch
        {
            1 => "plains".Localize(),
            2 => "desert".Localize(),
            3 => "forrest".Localize(),
            4 => "mountains".Localize(),
            _ => "lobby".Localize()
        };
    }
    private void DeleteTile()
    {
        if (activeTile.Count == 0) return;
        var tile = activeTile[0];
        var tileIndex = tile.GetComponent<ObjectGeneration>().tileType;

        if (tileIndex < pools.Length)
        {
            pools[tileIndex].ReturnToPool(tile);
        }
        else
        {
            tile.GetComponent<ObjectGeneration>().StartIncrementalDestroy();
        }

        activeTile.RemoveAt(0);
    }

    //==================== ГЕНЕРАЦИЯ БУКВ ====================
    void TrySpawnLettersOnTile(Transform tileTransform, Vector3 basePosition)
    {
        if (Random.value > letterSpawnChance || correctNumWords <= 0) return;

        Vector3[] spawnPoints =
        {
            basePosition + new Vector3(-3, 0, 0),
            basePosition + new Vector3(0, 0, 0),
            basePosition + new Vector3(3, 0, 0),
        };

        char correctLetter = '\0';

        if (GlobalVar.difficulty == 1)
        {
            correctLetter = correctMissingLetter;
        }
        else if (GlobalVar.difficulty == 2 && currentLetterIndex <
hiddenKazakhWord.Length)
        {
            correctLetter = hiddenKazakhWord[currentLetterIndex];
        }
        else if (GlobalVar.difficulty == 0 && currentLetterIndex <
remainingLetters.Count)
        {
            correctLetter = remainingLetters[currentLetterIndex];
        }

        if (correctLetter == '\0') return;

        List<char> choices;
        do
        {
            List<char> distractors =
                allKazakhLetters.Where(c => c != correctLetter).OrderBy(_ =>
Random.value).ToList();
            choices = new() { correctLetter, distractors[0], distractors[1] };
            choices = choices.OrderBy(_ => Random.value).ToList();
        } while (IsSwearWord(choices));



        for (int i = 0; i < spawnPoints.Length; i++)
        {
            Vector3 pos = spawnPoints[i];
            letterPositions.Add(pos);

            GameObject letterObj = Instantiate(letter3DPrefab, pos,
Quaternion.identity, tileTransform);
            letterObj.GetComponentInChildren<TextMeshPro>().text =
char.ToUpper(choices[i]).ToString();
            letterObj.GetComponent<Letter3D>().Setup(choices[i], this);
        }
    }


    private readonly string[] swearWords =
{"мал","хуй","кет","гей","ебу","нах","пох"};

    //Проверка слов на маты и прочее
    private bool IsSwearWord(List<char> letters)
    {
        var word = letters.Aggregate("", (current, let) => current + let);

        if(swearWords.Any(w => w.Equals(word,
StringComparison.CurrentCultureIgnoreCase))){
            print($"{word} - мат, генерируем новые буквы");
            return true;
        }

        return false;
    }

    //================ПРОВЕРКА БУКВЫ НА СООТВЕСТВИЕ ПОЗИЦИИ
==================

    #region Проверка букв
    public void OnLetterCollected(char selected)
    {
        if (GlobalVar.difficulty == 0)
        {
            UpdateLetterEasy(selected);
        }
        else if (GlobalVar.difficulty == 1)
        {
            UpdateLetterMedium(selected);
        }
        else if (GlobalVar.difficulty == 2)
        {
            UpdateLetterHard(selected);
        }
    }
    private void UpdateLetterEasy(char selected)
    {
        if (currentLetterIndex >= remainingLetters.Count)
            return;

        char expected = remainingLetters[currentLetterIndex];


        // Сравнение без приведения к верхнему регистру
        if (!LettersMatch(selected, expected))
        {
            WrongLetterCollected();
            return;
        }


        audioSource.clip = audioClip[0];
        audioSource.Play();
        Debug.Log($"[Level 0] Выбрана буква: {selected}, ожидается:
{expected}");
        collectedLetters.Add(selected);
```

```
        collectWordUIController.CollectLetter(currentLetterIndex);
        currentLetterIndex++;

        if (currentLetterIndex >= remainingLetters.Count)
        {
            HandleWordCollection();
        }
    }

    private void HandleWordCollection()
    {
        Debug.Log("🎉 Слово собрано!");
        collectWordUIController.PlayWordCloseAnimation();
        currentWord.IsCollected = true;
        audioSource.clip = audioClip[2];
        audioSource.Play();
        GlobalVar.currCollectWord++;
        GlobalVar.collectedWords.Add(currentWord);
        collectedLetters.Clear();
        remainingLetters.Clear();

        //============================Запускаем озвучивание
озвучуи==============================
        StartCoroutine(PlayAndGenerateNext(0.1f, currentWord));

//============================================================
==========================

        SaveWordsToJsonAsync();
    }

    private IEnumerator PlayAndGenerateNext(float delay, WordData oldWord)
    {
        yield return new WaitForSeconds(delay);

        if (GlobalVar.WordVoiceover)
        {
            yield return playerController.LoadAndPlayAudio(oldWord);
        }
        else
        {
            yield return new WaitForSeconds(1f);
        }


        yield return GenerateNextWord();
    }

    private void UpdateLetterMedium(char selected)
    {
        Debug.Log($"[Level 1] Выбрана буква: {selected}, ожидается:
{correctMissingLetter}");

        if (!selected.Equals(correctMissingLetter))
        {
            WrongLetterCollected();
            return;
        }

        collectWordUIController.CollectLetter(missingLetterIndex);

        Debug.Log("✅ Верная буква (средний уровень)");
        HandleWordCollection();
    }

    private void UpdateLetterHard(char selected)
    {
        int hiddenLen = hiddenKazakhWord.Length;

        if (currentLetterIndex >= hiddenLen)
            return;

        char expected = hiddenKazakhWord[currentLetterIndex];
        Debug.Log($"[Level 2] Выбрана буква: {selected}, ожидается:
{expected}");

        // ❌ Неправильная буква
        if (!LettersMatch(selected, expected))
        {
            WrongLetterCollected();
            return;
        }

        // ✅ Верная буква
        collectedLetters.Add(selected);

        collectWordUIController.CollectLetter(currentLetterIndex);

        currentLetterIndex++;

        audioSource.clip = audioClip[0]; // звук правильной буквы
        audioSource.Play();

        // 🎉 Проверка окончания слова
        if (collectedLetters.Count >= hiddenLen)
        {
            Debug.Log("🎉 Слово собрано (Hard)!");
            HandleWordCollection();
        }
    }

    private void WrongLetterCollected()
    {
        if (isImmune)
        {
            Debug.Log("⚠️ Неправильная буква, но есть иммунитет — игнор.");
            return;
        }
        audioSource.clip = audioClip[1];
        audioSource.Play();
        Debug.Log("❌ Неверная буква");
        if (player != null) playerController.OnIncorrectLetter();
    }

    #endregion

    #region Вспомогательные Методы

    private bool LettersMatch(char a, char b)
    {
        return string.Equals(a.ToString(), b.ToString(),
StringComparison.CurrentCultureIgnoreCase);
    }
    public bool IsLetterValid(char letter)
    {
        if (GlobalVar.difficulty == 0)
            return remainingLetters.Count > currentLetterIndex &&
                LettersMatch(letter, remainingLetters[currentLetterIndex]);

        if (GlobalVar.difficulty == 1)
            return LettersMatch(letter, correctMissingLetter);

        if (GlobalVar.difficulty == 2)
            return currentLetterIndex < hiddenKazakhWord.Length &&
                LettersMatch(letter, hiddenKazakhWord[currentLetterIndex]);

        return false;
    }

    #endregion

    #region Де-Генерация слов



    //==================== СЛЕДУЮЩЕЕ СЛОВО
====================
    private IEnumerator GenerateNextWord()
    {

        currentLetterIndex = 0;
        GlobalVar.isWordFinished = false;
        if (!isFirstWord)
        {
            correctNumWords--;
            if (correctNumWords <= 0)
            {
                SpawnPortal();
                yield break;
            }
        }
        else
        {
            isFirstWord = false;
        }

        collectWordUIController.PlayWordOpenAnimation();


        collectedLetters.Clear();
        remainingLetters.Clear();

        if (GlobalVar.levelType == 0) yield break;

        switch (GlobalVar.difficulty)
        {
            case 0:
                GenerateWordEasy();
```

```
            break;
        case 1:
            GenerateWordMedium();
            break;
        case 2:
            GenerateWordHard();
            break;
    }

    //Озвучка слова сразу после его генерации
    yield return new WaitForSeconds(0.5f);

    if (GlobalVar.WordVoiceover)
    {
        yield return playerController.LoadAndPlayAudio(currentWord);
    }
    else
    {
        yield return new WaitForSeconds(0.5f);
    }
}

private void SpawnPortal()
{
    portalQueued = true;
    GlobalVar.isWordFinished = true;
}
//======================= ПОРТАЛЫ =========================

//==================== ПРОБРОС ПО УРОВНЮ СЛОЖНОСТИ
====================
void GenerateWordEasy()
{
    currentLetterIndex = 0;
    List<WordData> filteredWords = GetFilteredWords();

    if (filteredWords.Count == 0) return;

    currentWord = filteredWords[Random.Range(0, filteredWords.Count)];
    collectWordUIController.SetUpWord(currentWord, Difficulty.Easy);
    //remainingLetters = currentWord.KazakhWord.Replace(" ",
"").ToCharArray().ToList();
    remainingLetters =
CleanWord(currentWord.KazakhWord).ToCharArray().ToList();
}

void GenerateWordMedium()
{
    currentLetterIndex = 0;
    List<WordData> filteredWords = GetFilteredWords();
    if (filteredWords.Count == 0) return;

    currentWord = filteredWords[Random.Range(0, filteredWords.Count)];
    //string kazakh = currentWord.KazakhWord.Replace(" ", "");
    string kazakh = CleanWord(currentWord.KazakhWord);
    missingLetterIndex = Random.Range(0, kazakh.Length);
    correctMissingLetter = kazakh[missingLetterIndex];

    collectWordUIController.SetUpWord(currentWord, Difficulty.Normal,
missingLetterIndex);
}

//private string hiddenKazakhWord = "";
private int hiddenLen = 0;

void GenerateWordHard()
{
    currentLetterIndex = 0;

    var filteredWords = GetFilteredWords();
    if (filteredWords.Count == 0) return;

    currentWord = filteredWords[Random.Range(0, filteredWords.Count)];
    hiddenKazakhWord = CleanWord(currentWord.KazakhWord);   // без
пробелов и знаков
    hiddenLen = hiddenKazakhWord.Length;

    collectWordUIController.SetUpWord(currentWord, Difficulty.Hard);

    // на всякий случай лог
    Debug.Log($"[HARD] Слово: {currentWord.KazakhWord} ->
'{hiddenKazakhWord}', len={hiddenLen}");
}


void InitEasyMode()
{
    List<WordData> filteredWords = GetFilteredWords();
    if (filteredWords.Count > 0)
    {
        currentWord = filteredWords[Random.Range(0, filteredWords.Count)];
        collectWordUIController.SetUpWord(currentWord, Difficulty.Easy);
        remainingLetters = currentWord.KazakhWord.Replace(" ",
"").ToCharArray().ToList();
    }
    else if (filteredWords.Count == 0 && GlobalVar.levelType != 0)
    {
        Debug.LogWarning("Нет слов в выбранной категории.");
    }
}

void InitMediumMode()
{
    GenerateWordMedium();
}


void InitHardMode()
{
    GenerateWordHard();
}

//================ИМУННА НА ОШИБКУ=============
public void ActivateImmunity(float duration)
{
    if (immunityCoroutine != null)
        StopCoroutine(immunityCoroutine);

    immunityCoroutine = StartCoroutine(ImmunityTimer(duration));
}

private IEnumerator ImmunityTimer(float duration)
{
    isImmune = true;
    Debug.Log("🛡 Иммунитет активирован");
    yield return new WaitForSeconds(duration);
    isImmune = false;
    Debug.Log("🛡 Иммунитет завершён");
}


#endregion

#region Рестарт

//====================РЕСТАРТ ИГРЫ==================//
public void RestartGameZone()
{
    foreach (GameObject tile in activeTile)
    {
        Destroy(tile);
    }
    activeTile.Clear();
    spawnPos = -50f;

    for (int i = 0; i < startTiles; i++)
    {
        int index = (i == 0) ? 4 : 0;
        SpawnTile(index);
    }

    GlobalVar.landscapeType = 0;


tileUIController.UpdateTileName(GetLandscapeName(GlobalVar.landscapeTyp
e));

    ResetWordState();
    switch (GlobalVar.difficulty)
    {
        case 0: InitEasyMode(); break;
        case 1: InitMediumMode(); break;
        case 2: InitHardMode(); break;
    }

    Debug.Log("🔄 Игра перезапущена после смерти игрока");
}

#endregion

//==================РАБОТА НАД СЛОВАРЕМ============//
public async void SaveWordsToJsonAsync()
{
    FileLoader.SaveWordsToJSON(wordDictionary);
}

public void CleanUI()
{
```

```
        collectWordUIController.CleanWordUI();
    }
}


using System.Collections.Generic;
using UnityEngine;
using UnityEngine.Pool;

namespace Scenes.Object.Nature.Ground.Scripts
{
    public class TilePool : MonoBehaviour
    {
        // The pool holds plain GameObjects (you can swap this for any
component type).
        private ObjectPool<GameObject> pool;

        [SerializeField] private bool prefillPool;
        [SerializeField] private int prefillCount = 5;

        void Awake()
        {
            // Create a pool with the four core callbacks.
            pool = new ObjectPool<GameObject>(
                createFunc: CreateItem,
                actionOnGet: OnGet,
                actionOnRelease: OnRelease,
                actionOnDestroy: OnDestroyItem,
                collectionCheck: true, // helps catch double-release mistakes
                defaultCapacity: prefillCount,
                maxSize: 10
            );
        }

        private GameObject _prefab;

        public void InitPool(GameObject prefab)
        {
            _prefab = prefab;
            if(prefillPool) PreFillPool();
        }

        private void PreFillPool()
        {
            var items = new List<GameObject>();
            for (int i = 0; i < prefillCount; i++)
            {
                var item = pool.Get();
                item.transform.position = new Vector3(-1000, -1000, -1000);
                items.Add(item);
            }
            foreach (var item in items)
            {
                pool.Release(item);
            }
        }

        public GameObject GetFromPool(Vector3 position, Quaternion rotation)
        {
            var item = pool.Get();
            item.transform.position = position;
            item.transform.rotation = rotation;
            return item;
        }

        public void ReturnToPool(GameObject item)
        {
            pool.Release(item);
        }


        // Creates a new pooled GameObject the first time (and whenever the pool
needs more).
        private GameObject CreateItem()
        {
            var instance = Instantiate(_prefab, transform, true);

            instance.SetActive(false);

            return instance;
        }

        // Called when an item is taken from the pool.
        private void OnGet(GameObject gameObject)
        {
            gameObject.SetActive(true);
            gameObject.GetComponent<ObjectGeneration>().Initialize();
        }

        // Called when an item is returned to the pool.
        private void OnRelease(GameObject gameObject)
        {
            gameObject.GetComponent<ObjectGeneration>().StartPoolReturn();
            gameObject.transform.position = new Vector3(0, -1000, 0);
        }

        // Called when the pool decides to destroy an item (e.g., above max size).
        private void OnDestroyItem(GameObject gameObject)
        {
            Destroy(gameObject);
        }
    }
}


using System;
using System.Collections;
using System.Collections.Generic;
using System.Linq;
using UnityEngine;
using Random = UnityEngine.Random;

public class RockBehaviour : MonoBehaviour
{
    [SerializeField] private Transform explosionCenter;
    [SerializeField] private List<GameObject> particles;
    [SerializeField] private Renderer mainRenderer;
    [SerializeField] private float explosionForce;
    [SerializeField] private float explosionTorgue;
    [SerializeField] private List<AudioClip> sounds;

    private List<Renderer> _particleRenders;
    private List<Rigidbody> _particleRbs;
    private AudioSource _audioSource;
    private bool _hasExploded;


    private void Awake()
    {
        Init();
    }

    private void Init()
    {
        _particleRenders = particles.Select(x =>
x.GetComponent<Renderer>()).ToList();
        _particleRbs = particles.Select(x =>
x.GetComponent<Rigidbody>()).ToList();
        _audioSource = GetComponent<AudioSource>();

        mainRenderer.enabled = true;
        _particleRenders.ForEach(x => x.enabled = false);;
        _particleRbs.ForEach(x => x.isKinematic = true);

        _hasExploded = false;
    }

    private void OnTriggerEnter(Collider other)
    {
        if (other.CompareTag("Player")) Explode();
    }

    private void Explode()
    {
        if(_hasExploded) return; //prevent multiple explosions (shouldn't happen')

        _hasExploded = true;

        mainRenderer.enabled = false;

        var randomSound = sounds[Random.Range(0, sounds.Count)];
        _audioSource.PlayOneShot(randomSound);

        _particleRenders.ForEach(x => x.enabled = true);
        _particleRbs.ForEach(x => x.isKinematic = false);

        //add random force to all rigidbodies spreading from the ground zero

        foreach (var rb in _particleRbs)
        {
            var direction = - (explosionCenter.position - rb.position);
            var forceVector = direction.normalized * Random.Range(0.75f, 1.25f) *
explosionForce;

            rb.AddForce(forceVector, ForceMode.Impulse);
            //add random spin to rb
            rb.AddTorque(Random.insideUnitSphere * Random.Range(0.5f, 1f) *
explosionTorgue);
```

```
            //reset to kinematic after a few seconds to prevent using memory
            StartCoroutine(SetKinematicRoutine());
        }
    }

    private IEnumerator SetKinematicRoutine()
    {
        yield return new WaitForSeconds(3f);

        _particleRbs.ForEach(x => x.isKinematic = true);
    }
}



using System.Collections;
using UnityEngine;

public class Archar : MonoBehaviour
{
    [Header("Refs")]
    [SerializeField] private Animator animator;
    [SerializeField] private Rigidbody rb;
    [SerializeField] private Transform target;

    [Header("Dash")]
    [SerializeField] private bool useImpulse = true;
    [SerializeField] private float delayBeforeDash = 0.12f;
    [SerializeField] private float dashImpulse = 10f;   // ××× Impulse
    [SerializeField] private float dashSpeed = 12f;   // ××× velocity-mode
    [SerializeField] private float dashDuration = 0.18f; // ××× velocity-mode
    [SerializeField] private float dashStopDelay = 0.18f; // ×××××× ××××××
×××××××

    [Header("Control")]
    [SerializeField] private bool faceTarget = true;
    [SerializeField] private float cooldown = 0.4f;
    [SerializeField] private bool zeroOutVelocityBeforeDash = true;

    [Header("Model orientation")]
    [Tooltip("×××× ×××××××××× ××××××× ×××××××× × -Z
×××××××××××××× transform.forward, ××××××× 180.")]
    [SerializeField] private float yawOffsetDeg = 180f; // 0 ××× 180
×××××××
    private GameObject player;
    private Quaternion YawOffset => Quaternion.Euler(0f, yawOffsetDeg, 0f);
    private bool isDashing, onCooldown, isDoRamAttack;
    private float dist;
    void Reset()
    {
        animator = GetComponentInChildren<Animator>();
        rb = GetComponent<Rigidbody>();
    }

    void Start()
    {
        player = GameObject.Find("Players");
        target = player.transform;
        // ××××××××××× ××××××× × ××××××××××××
××××××××× ×××××× ×× ×××××××
        //transform.rotation = Quaternion.Euler(0f, 180f, 0f);  //transform.rotation *
YawOffset;
    }

    void Update()
    {
        //transform.rotation = Quaternion.Euler(0f, 180f, 0f);
        dist = Vector3.Distance(player.transform.position, transform.position);

        if (Input.GetKeyDown(KeyCode.Space)) TryStartDash();
        if (Vector3.Distance(player.transform.position, transform.position) < 20 &&
!isDoRamAttack)
        {
            TryStartDash();
            isDoRamAttack = true;
        }

    }

    private void TryStartDash()
    {
        if (isDashing || onCooldown) return;
        StartCoroutine(DoRamAttack());
    }

    private IEnumerator DoRamAttack()
    {
        isDashing = true;
        // ××××××××××××× × ×××× × ×××××××
××××××××××××× ××××××××
        if (faceTarget && target)
        {
            Vector3 toTarget = target.position - transform.position;
            toTarget.y = 0f;
            if (toTarget.sqrMagnitude > 0.0001f)
            {
                var look = Quaternion.LookRotation(toTarget.normalized, Vector3.up)
* YawOffset;
                transform.rotation = look;
            }
        }

        if (animator) animator.SetTrigger("Fight");

        if (delayBeforeDash > 0f)
            yield return new WaitForSeconds(delayBeforeDash);

        // ×××××××××× ××××××××××××× ×××××× ×××
××××××××××××× ××××××××:
        // (×.×. ×××× ××××××× ××××××× ××××××)
        Vector3 dashDir = (Quaternion.Inverse(YawOffset) *
transform.forward).normalized;

        if (zeroOutVelocityBeforeDash)
            rb.linearVelocity = new Vector3(0f, rb.linearVelocity.y, 0f);

        if (useImpulse)
        {
            rb.AddForce(dashDir * dashImpulse, ForceMode.Impulse);
            yield return new WaitForSeconds(dashStopDelay);
            // ×××× ×× XZ (×××××××××× Y)
            rb.linearVelocity = new Vector3(0f, rb.linearVelocity.y, 0f);
        }
        else
        {
            float t = 0f;
            while (t < dashDuration)
            {
                Vector3 v = dashDir * dashSpeed;
                v.y = rb.linearVelocity.y;
                rb.linearVelocity = v;
                t += Time.deltaTime;
                yield return null;
            }
            rb.linearVelocity = new Vector3(0f, rb.linearVelocity.y, 0f);
        }

        isDashing = false;
        onCooldown = true;
        yield return new WaitForSeconds(cooldown);
        onCooldown = false;
    }
}



using UnityEngine;

public class Bercut : MonoBehaviour
{
    [SerializeField] private Animator animator;
    [SerializeField] private AudioSource audioSource;
    [SerializeField] private AudioClip[] clip;
    [Header(" ")]
    [SerializeField] private float attackDistance = 20f; //   Z
    [SerializeField] private float laneWidth = 4f;       //
PlayerController.lineDistance

    [Header(" ")]
    [SerializeField] private float diveSpeed = 10f; //   ( Z)
    [SerializeField] private float descendSpeed = 2f; //
    [SerializeField] private float stopHeight = 1.5f;   //
    [SerializeField] private float overshootZ = 4f;     //

    [SerializeField] private Transform player;

    private bool isDiving = false;
    private int myLane;          //   0/1/2 ( X)
    private float laneX;         //    X

    //
    private float targetZ;       //    Z (     + overshoot)
    private float targetY;       //    ( stopHeight)

    void Start()
    {
```

```
        player = GlobalVar.player.transform;

        //      X
        float x = transform.position.x;
        if (Mathf.Abs(x) < 1f) myLane = 1;
        else if (x > 0f) myLane = 2;
        else myLane = 0;

        laneX = LaneCenterX(myLane);
    }

    void Update()
    {
        if (GlobalVar.autoDestroy) { gameObject.SetActive(false); return; }

        if (!isDiving)
        {
            int playerLane = GlobalVar.playerController?.GetCurrentLine() ?? -1;

            // ,            attackDistance  Z
            float dz = Mathf.Abs(transform.position.z - player.position.z);
            if (playerLane == myLane && dz <= attackDistance)
            {
                StartDive();
            }
        }
        else
        {
            DiveForwardOnly();
        }
    }

    private void StartDive()
    {
        isDiving = true;
        //    Z  Y   (  )
        targetZ = player.position.z + overshootZ; //    ,
        targetY = stopHeight;
        int index = Random.Range(0, clip.Length);
        audioSource.clip = clip[index];
        animator.SetTrigger("Attack");
        audioSource.Play();
    }

    private void DiveForwardOnly()
    {
        Vector3 p = transform.position;

        //  X      (   )
        p.x = Mathf.MoveTowards(p.x, laneX, 100f * Time.deltaTime); //    ,

        //  Ш  Z    targetZ
        float dirZ = Mathf.Sign(targetZ - p.z); // +1   , -1
        p.z += dirZ * diveSpeed * Time.deltaTime;

        //
        p.y = Mathf.MoveTowards(p.y, targetY, descendSpeed * Time.deltaTime);

        transform.position = p;

        //
        Vector3 fwd = new Vector3(0f, 0f, dirZ);
        if (fwd.sqrMagnitude > 0.01f)
            transform.rotation = Quaternion.Slerp(transform.rotation,
Quaternion.LookRotation(fwd, Vector3.up), Time.deltaTime * 5f);

        //  ,    Z
        if ((dirZ > 0f && p.z >= targetZ) || (dirZ < 0f && p.z <= targetZ))
        {
            isDiving = false;
            //  : ,  /,   ..
            // gameObject.SetActive(false);
        }
    }

    private float LaneCenterX(int lane)
    {
        switch (lane)
        {
            case 0: return -laneWidth;
            case 1: return 0f;
            case 2: return laneWidth;
            default: return 0f;
        }
    }

    private void OnTriggerEnter(Collider other)
    {
        if (other.CompareTag("Player"))
        {
            // /
            gameObject.SetActive(false);
        }
    }

}



using UnityEngine;

public class Scorpion : MonoBehaviour
{
    [SerializeField]
    private Animator animator;
    [SerializeField]
    private AudioSource audioSource;
    private int lineToMove;
    private float dis;


    void Start()
    {
        float x = transform.position.x;
        if (Mathf.Abs(x) < 1f)
            lineToMove = 1;
        else if (x > 0)
            lineToMove = 2;
        else
            lineToMove = 0;
    }
    void Update()
    {
        if (GlobalVar.autoDestroy) gameObject.SetActive(false);

        dis = Vector3.Distance(gameObject.transform.position,
GlobalVar.player.transform.position);
        int playerLine = GlobalVar.playerController?.GetCurrentLine() ?? -1;
        if (GlobalVar.autoDestroy) Destroy(gameObject);
        if (dis < 18 && playerLine == lineToMove)
        {
            animator.SetTrigger("Attack");
        }
    }

    private void OnTriggerEnter(Collider other)
    {
        if (other.CompareTag("Player"))
        {
            gameObject.SetActive(false);
        }
    }

}



using UnityEngine;

public class Snake : MonoBehaviour
{
    [SerializeField]
    private Animator animator;
    [SerializeField]
    private AudioSource  audioSource;

    private int lineToMove;
    private bool canPlay = true;
    private float dis;

    // Start is called once before the first execution of Update after the
MonoBehaviour is created
    void Start()
    {
        transform.rotation = Quaternion.Euler(0f, 180f, 0f);
        float x = transform.position.x;
        if (Mathf.Abs(x) < 1f)
            lineToMove = 1;
        else if (x > 0)
            lineToMove = 2;
        else
            lineToMove = 0;
    }

    // Update is called once per frame
    void Update()
    {
        if (GlobalVar.autoDestroy) gameObject.SetActive(false);
```

```
        if(GlobalVar.player == null) return;

        dis = Vector3.Distance(gameObject.transform.position,
GlobalVar.player.transform.position);
        int playerLine = GlobalVar.playerController?.GetCurrentLine() ?? -1;
        if (GlobalVar.autoDestroy) Destroy(gameObject);

        if (dis < 13 && playerLine == lineToMove)
        {
            if (!audioSource.isPlaying && canPlay)
            {
                audioSource.Play();
            }
            canPlay = false;
            animator.SetTrigger("Attack");
        }
    }

    private void OnTriggerEnter(Collider other)
    {
        if (other.CompareTag("Player"))
        {
            gameObject.SetActive(false);
        }
    }
}



using UnityEngine;

public class Bee : MonoBehaviour
{
    [SerializeField]
    private Animator animator;
    [SerializeField]
    private AudioSource audioSource;

    [SerializeField]
    private float speedZ = 15f;         // скорость вперёд (по Z)
    [SerializeField]
    private float laneSwitchSpeed = 5f; // скорость смены дорожки (по X)
    [SerializeField]
    private float triggerDistance = 30f;

    private Transform player;
    private bool isChasing = false;
    // Start is called once before the first execution of Update after the
MonoBehaviour is created
    void Start()
    {
        transform.rotation = Quaternion.Euler(0f, 180f, 0f);
        //player = GlobalVar.player.transform;
        player = GlobalVar.player.transform;
        //// При старте ставим ту же X-позицию, что у игрока (дорожку)
        //Vector3 startPos = transform.position;
        //startPos.x = player.position.x;
        //transform.position = startPos;
    }

    // Update is called once per frame
    void Update()
    {
        if (GlobalVar.autoDestroy) gameObject.SetActive(false);

        if (player == null) return;

        float distanceToPlayer = Vector3.Distance(transform.position,
player.position);

        // Запускаем погоню, если игрок близко
        if (!isChasing && distanceToPlayer <= triggerDistance)
        {
            isChasing = true;
        }

        if (isChasing)
        {
            // Движение вперёд (по Z)
            Vector3 newPos = transform.position;
            newPos.z = Mathf.MoveTowards(transform.position.z, player.position.z,
speedZ * Time.deltaTime);

            // Движение в сторону игрока (по X — смена дорожки)
            newPos.x = Mathf.MoveTowards(transform.position.x, player.position.x,
laneSwitchSpeed * Time.deltaTime);

            transform.position = newPos;

            // Вращение в сторону игрока (опционально)
            Vector3 lookAtPos = new Vector3(player.position.x, transform.position.y,
player.position.z);
            transform.LookAt(lookAtPos);
        }
    }



    private void OnTriggerEnter(Collider other)
    {
        if (other.CompareTag("Player"))
        {
            gameObject.SetActive(false);
        }
    }

}



using TMPro;
using UnityEngine;
using UnityEngine.SocialPlatforms.Impl;
using UnityEngine.UI;

public class Score : MonoBehaviour
{
    [SerializeField] TMP_Text distanceText;
    [SerializeField] TMP_Text scoreText;
    [SerializeField] TMP_Text coinText;
    private Transform player;
    //PlayerController playerController;
    //int maxScore = 999999999;
    void Start()
    {
        player = transform;
        //playerController = GetComponent<PlayerController>();
    }

    // Update is called once per frame
    void Update()
    {
        distanceText.text = ((int)(player.position.z/2)-4).ToString();
        string formatted = GlobalVar.Asyq.ToString();
//GlobalVar.FormatScore(GlobalVar.Coins);
        scoreText.text = formatted;
    }


    //public string FormatScore(int score, int maxValue)
    //{
    //    int totalDigits = maxValue.ToString().Length;
    //    return score.ToString("D" + totalDigits);
    //}

}



using UnityEngine;
using UnityEngine.EventSystems;
using UnityEngine.UI;

public class PassDragToScrollRect : MonoBehaviour,
    IBeginDragHandler, IDragHandler, IEndDragHandler, IScrollHandler
{
    [SerializeField]
    private ScrollRect scroll;

    void Awake()
    {
        //scroll = GetComponentInParent<ScrollRect>();
    }

    public void OnBeginDrag(PointerEventData e)
    {
        if (scroll) scroll.OnBeginDrag(e);
    }

    public void OnDrag(PointerEventData e)
    {
        if (scroll) scroll.OnDrag(e);
    }

    public void OnEndDrag(PointerEventData e)
    {
        if (scroll) scroll.OnEndDrag(e);
```

```
    }

    //     (     )
    public void OnScroll(PointerEventData e)
    {
        if (scroll) scroll.OnScroll(e);
    }
}
```

```
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.Events;

namespace Scenes.Object.UI.Translater
{
    public class LocalizationManager : MonoBehaviour
    {
        [SerializeField]
        private TranslationDatabaseAsset translationDatabaseAsset;

        private Dictionary<string, TranslationEntry> translationDictionary;

        public UnityEvent<int> OnLanguageChanged;

        public static LocalizationManager Instance;
        private static readonly List<LocalizedTextElement> registeredElements =
new();

        private void Awake()
        {
            // populate the dictionary from the ScriptableObject asset
            if (translationDatabaseAsset != null)
            {
                translationDictionary = translationDatabaseAsset.ToDictionary();
            }
            else
            {
                Debug.LogWarning("LocalizationManager: No
TranslationDatabaseAsset assigned. Translations will fallback to keys.");
                translationDictionary = new Dictionary<string, TranslationEntry>();
            }

            if (Instance != null && Instance != this)
            {
                Destroy(gameObject);
                return;
            }
            Instance = this;
        }

        public static void Register(LocalizedTextElement element)
        {
            if (!registeredElements.Contains(element))
                registeredElements.Add(element);
        }

        public void UpdateLanguage(int lang)
        {
            foreach (var element in registeredElements)
            {
                element.UpdateLanguage(lang);
            }

            OnLanguageChanged?.Invoke(lang);
        }

        public string GetTranslation(string key)
        {
            if (string.IsNullOrEmpty(key))
                return string.Empty;
            if (translationDictionary != null &&
translationDictionary.TryGetValue(key, out var entry) && entry != null)
            {
                return entry.GetTranslation(GlobalVar.Lang);
            }
            // Fallback to key if missing translation
            return key;
        }

    }

    public static class StringLocalize
    {
        public static string Localize(this string key)
        {
            return LocalizationManager.Instance.GetTranslation(key);
        }
    }
}
```

```
using Scenes.Object.UI.Translater;
using TMPro;
using UnityEngine;

public class LocalizedTextElement : MonoBehaviour
{
    private TMP_Text targetText;
    [SerializeField] private string key; // Optional key for localization database
entries
    [SerializeField] private bool useKeyFromDatabase = false; // When true, use
translation from database by key
    [SerializeField] private string textKz;
    [SerializeField] private string textRu;

    public string Key
    {
        get => key;
        set => key = value;
    }

    public bool UseKeyFromDatabase
    {
        get => useKeyFromDatabase;
        set => useKeyFromDatabase = value;
    }

    public string TextKz => textKz;
    public string TextRu => textRu;

    private void Awake()
    {
        targetText = GetComponent<TMP_Text>();
    }

    private void OnEnable()
    {
        LocalizationManager.Register(this);
        UpdateLanguage(GlobalVar.Lang);
    }

    public void UpdateLanguage(int lang)
    {
        if (targetText == null) targetText = GetComponent<TMP_Text>();

        if (useKeyFromDatabase && !string.IsNullOrEmpty(key) &&
LocalizationManager.Instance != null)
        {
            targetText.text = LocalizationManager.Instance.GetTranslation(key);
            return;
        }

        targetText.text = (lang == 0) ? textKz : textRu;
    }
}
```

```
using System.Collections.Generic;
using System.Linq;
using UnityEngine;

namespace Scenes.Object.UI.Translater
{
    [CreateAssetMenu(fileName = "TranslationDatabase", menuName =
"Localization/Translation Database")]
    public class TranslationDatabaseAsset : ScriptableObject
    {
        [SerializeField]
        public List<TranslationEntry> entries = new();

        public Dictionary<string, TranslationEntry> ToDictionary()
        {
            if (entries == null)
                return new Dictionary<string, TranslationEntry>();
            // If there are duplicate keys, last one wins
            return entries
                .Where(e => e != null && !string.IsNullOrEmpty(e.Key))
                .GroupBy(e => e.Key)
                .ToDictionary(g => g.Key, g => g.Last());
        }

        public bool TryGet(string key, out TranslationEntry entry)
        {
```

```
            entry = null;
            if (string.IsNullOrEmpty(key) || entries == null)
                return false;
            // Linear search is fine for authoring; runtime uses dictionary built by
manager
            for (int i = 0; i < entries.Count; i++)
            {
                var e = entries[i];
                if (e != null && e.Key == key)
                {
                    entry = e;
                    return true;
                }
            }
            return false;
        }
    }
}


using System;
using UnityEngine;

namespace Scenes.Object.UI.Translater
{
    [Serializable]
    public class TranslationEntry
    {
        [SerializeField]
        private string key;
        [SerializeField]
        private string ruTranslation;
        [SerializeField]
        private string kzTranslation;

        public string Key => key;

        public string GetTranslation(int lang) => (lang == 0) ? kzTranslation :
ruTranslation;
    }
}


/// Credit SimonDarksideJ

using System.Collections.Generic;

namespace UnityEngine.UI.Extensions
{
    public static class ShaderLibrary
    {
        public static Dictionary<string, Shader> shaderInstances = new
Dictionary<string, Shader>();
        public static Shader[] preLoadedShaders;

        public static Shader GetShaderInstance(string shaderName)
        {
            if (shaderInstances.ContainsKey(shaderName))
            {
                return shaderInstances[shaderName];
            }

            var newInstance = Shader.Find(shaderName);
            if (newInstance != null)
            {
                shaderInstances.Add(shaderName, newInstance);
            }
            return newInstance;
        }
    }
}



/// Credit glennpow, Zarlang
/// Sourced from - http://forum.unity3d.com/threads/free-script-particle-systems-
in-ui-screen-space-overlay.406862/
/// Updated by Zarlang with a more robust implementation, including
TextureSheet animation support

namespace UnityEngine.UI.Extensions
{
#if UNITY_5_3_OR_NEWER
    [ExecuteInEditMode]
    [RequireComponent(typeof(CanvasRenderer), typeof(ParticleSystem))]
    [AddComponentMenu("UI/Effects/Extensions/UIParticleSystem")]
    public class UIParticleSystem : MaskableGraphic
    {
        [Tooltip("Having this enabled run the system in LateUpdate rather than in
Update making it faster but less precise (more clunky)")]
        public bool fixedTime = true;

        [Tooltip("Enables 3d rotation for the particles")]
        public bool use3dRotation = false;

        private Transform _transform;
        private ParticleSystem pSystem;
        private ParticleSystem.Particle[] particles;
        private UIVertex[] _quad = new UIVertex[4];
        private Vector4 imageUV = Vector4.zero;
        private ParticleSystem.TextureSheetAnimationModule
textureSheetAnimation;
        private int textureSheetAnimationFrames;
        private Vector2 textureSheetAnimationFrameSize;
        private ParticleSystemRenderer pRenderer;
        private bool isInitialised = false;

        private Material currentMaterial;

        private Texture currentTexture;

#if UNITY_5_5_OR_NEWER
        private ParticleSystem.MainModule mainModule;
#endif

        public override Texture mainTexture
        {
            get
            {
                return currentTexture;
            }
        }

        protected bool Initialize()
        {
            // initialize members
            if (_transform == null)
            {
                _transform = transform;
            }
            if (pSystem == null)
            {
                pSystem = GetComponent<ParticleSystem>();

                if (pSystem == null)
                {
                    return false;
                }

#if UNITY_5_5_OR_NEWER
                mainModule = pSystem.main;
                if (pSystem.main.maxParticles > 14000)
                {
                    mainModule.maxParticles = 14000;
                }
#else
                    if (pSystem.maxParticles > 14000)
                        pSystem.maxParticles = 14000;
#endif

                pRenderer = pSystem.GetComponent<ParticleSystemRenderer>();
                if (pRenderer != null)
                    pRenderer.enabled = false;

                if (material == null)
                {
                    var foundShader =
ShaderLibrary.GetShaderInstance("Scenes/Object/UI/UIPartical/Shaders");
                    //var foundShader = ShaderLibrary.GetShaderInstance("UI
Extensions/Particles/Additive");
                    if (foundShader)
                    {
                        material = new Material(foundShader);
                    }
                }

                currentMaterial = material;
                if (currentMaterial && currentMaterial.HasProperty("_MainTex"))
                {
                    currentTexture = currentMaterial.mainTexture;
                    if (currentTexture == null)
                        currentTexture = Texture2D.whiteTexture;
                }
                material = currentMaterial;
                // automatically set scaling
#if UNITY_5_5_OR_NEWER
                mainModule.scalingMode = ParticleSystemScalingMode.Hierarchy;
```

```
#else
                pSystem.scalingMode = ParticleSystemScalingMode.Hierarchy;
#endif

            particles = null;
        }
#if UNITY_5_5_OR_NEWER
        if (particles == null)
            particles = new ParticleSystem.Particle[pSystem.main.maxParticles];
#else
            if (particles == null)
                particles = new ParticleSystem.Particle[pSystem.maxParticles];
#endif

        imageUV = new Vector4(0, 0, 1, 1);

        // prepare texture sheet animation
        textureSheetAnimation = pSystem.textureSheetAnimation;
        textureSheetAnimationFrames = 0;
        textureSheetAnimationFrameSize = Vector2.zero;
        if (textureSheetAnimation.enabled)
        {
            textureSheetAnimationFrames = textureSheetAnimation.numTilesX *
textureSheetAnimation.numTilesY;
            textureSheetAnimationFrameSize = new Vector2(1f /
textureSheetAnimation.numTilesX, 1f / textureSheetAnimation.numTilesY);
        }

        return true;
    }

    protected override void Awake()
    {
        base.Awake();
        if (!Initialize())
            enabled = false;
    }


    protected override void OnPopulateMesh(VertexHelper vh)
    {
#if UNITY_EDITOR
        if (!Application.isPlaying)
        {
            if (!Initialize())
            {
                return;
            }
        }
#endif
        // prepare vertices
        vh.Clear();

        if (!gameObject.activeInHierarchy)
        {
            return;
        }

        if (!isInitialised && !pSystem.main.playOnAwake)
        {
            pSystem.Stop(false,
ParticleSystemStopBehavior.StopEmittingAndClear);
            isInitialised = true;
        }

        Vector2 temp = Vector2.zero;
        Vector2 corner1 = Vector2.zero;
        Vector2 corner2 = Vector2.zero;
        // iterate through current particles
        int count = pSystem.GetParticles(particles);

        for (int i = 0; i < count; ++i)
        {
            ParticleSystem.Particle particle = particles[i];

            // get particle properties
#if UNITY_5_5_OR_NEWER
            Vector2 position = (mainModule.simulationSpace ==
ParticleSystemSimulationSpace.Local ? particle.position :
_transform.InverseTransformPoint(particle.position));
#else
                Vector2 position = (pSystem.simulationSpace ==
ParticleSystemSimulationSpace.Local ? particle.position :
_transform.InverseTransformPoint(particle.position));
#endif
            float rotation = -particle.rotation * Mathf.Deg2Rad;
            float rotation90 = rotation + Mathf.PI / 2;
            Color32 color = particle.GetCurrentColor(pSystem);
            float size = particle.GetCurrentSize(pSystem) * 0.5f;

            // apply scale
#if UNITY_5_5_OR_NEWER
            if (mainModule.scalingMode == ParticleSystemScalingMode.Shape)
                position /= canvas.scaleFactor;
#else
                if (pSystem.scalingMode == ParticleSystemScalingMode.Shape)
                    position /= canvas.scaleFactor;
#endif

            // apply texture sheet animation
            Vector4 particleUV = imageUV;
            if (textureSheetAnimation.enabled)
            {
#if UNITY_5_5_OR_NEWER
                float frameProgress = 1 - (particle.remainingLifetime /
particle.startLifetime);

                if (textureSheetAnimation.frameOverTime.curveMin != null)
                {
                    frameProgress =
textureSheetAnimation.frameOverTime.curveMin.Evaluate(1 -
(particle.remainingLifetime / particle.startLifetime));
                }
                else if (textureSheetAnimation.frameOverTime.curve != null)
                {
                    frameProgress =
textureSheetAnimation.frameOverTime.curve.Evaluate(1 -
(particle.remainingLifetime / particle.startLifetime));
                }
                else if (textureSheetAnimation.frameOverTime.constant > 0)
                {
                    frameProgress =
textureSheetAnimation.frameOverTime.constant - (particle.remainingLifetime /
particle.startLifetime);
                }
#else
                float frameProgress = 1 - (particle.lifetime / particle.startLifetime);
#endif

                frameProgress = Mathf.Repeat(frameProgress *
textureSheetAnimation.cycleCount, 1);
                int frame = 0;

                switch (textureSheetAnimation.animation)
                {

                    case ParticleSystemAnimationType.WholeSheet:
                        frame = Mathf.FloorToInt(frameProgress *
textureSheetAnimationFrames);
                        break;

                    case ParticleSystemAnimationType.SingleRow:
                        frame = Mathf.FloorToInt(frameProgress *
textureSheetAnimation.numTilesX);

                        int row = textureSheetAnimation.rowIndex;
#if UNITY_2020 || UNITY_2019
                        if (textureSheetAnimation.rowMode ==
ParticleSystemAnimationRowMode.Random)
#else
                        if (textureSheetAnimation.rowMode ==
ParticleSystemAnimationRowMode.Random) //   (    , )
#endif
                        { // FIXME - is this handled internally by rowIndex?
                            row = Mathf.Abs((int)particle.randomSeed %
textureSheetAnimation.numTilesY);
                        }
                        frame += row * textureSheetAnimation.numTilesX;
                        break;

                }

                frame %= textureSheetAnimationFrames;

                particleUV.x = (frame % textureSheetAnimation.numTilesX) *
textureSheetAnimationFrameSize.x;
                particleUV.y = 1.0f - ((frame / textureSheetAnimation.numTilesX) +
1) * textureSheetAnimationFrameSize.y;
                particleUV.z = particleUV.x + textureSheetAnimationFrameSize.x;
                particleUV.w = particleUV.y + textureSheetAnimationFrameSize.y;
            }

            temp.x = particleUV.x;
            temp.y = particleUV.y;

            _quad[0] = UIVertex.simpleVert;
            _quad[0].color = color;
            _quad[0].uv0 = temp;
```

```
            temp.x = particleUV.x;
            temp.y = particleUV.w;
            _quad[1] = UIVertex.simpleVert;
            _quad[1].color = color;
            _quad[1].uv0 = temp;

            temp.x = particleUV.z;
            temp.y = particleUV.w;
            _quad[2] = UIVertex.simpleVert;
            _quad[2].color = color;
            _quad[2].uv0 = temp;

            temp.x = particleUV.z;
            temp.y = particleUV.y;
            _quad[3] = UIVertex.simpleVert;
            _quad[3].color = color;
            _quad[3].uv0 = temp;

            if (rotation == 0)
            {
                // no rotation
                corner1.x = position.x - size;
                corner1.y = position.y - size;
                corner2.x = position.x + size;
                corner2.y = position.y + size;

                temp.x = corner1.x;
                temp.y = corner1.y;
                _quad[0].position = temp;
                temp.x = corner1.x;
                temp.y = corner2.y;
                _quad[1].position = temp;
                temp.x = corner2.x;
                temp.y = corner2.y;
                _quad[2].position = temp;
                temp.x = corner2.x;
                temp.y = corner1.y;
                _quad[3].position = temp;
            }
            else
            {
                if (use3dRotation)
                {
                    // get particle properties
#if UNITY_5_5_OR_NEWER
                    Vector3 pos3d = (mainModule.simulationSpace ==
ParticleSystemSimulationSpace.Local ? particle.position :
_transform.InverseTransformPoint(particle.position));
#else
                    Vector3 pos3d = (pSystem.simulationSpace ==
ParticleSystemSimulationSpace.Local ? particle.position :
_transform.InverseTransformPoint(particle.position));
#endif

                    // apply scale
#if UNITY_5_5_OR_NEWER
                    if (mainModule.scalingMode ==
ParticleSystemScalingMode.Shape)
                        position /= canvas.scaleFactor;
#else
                    if (pSystem.scalingMode == ParticleSystemScalingMode.Shape)
                        position /= canvas.scaleFactor;
#endif

                    Vector3[] verts = new Vector3[4]
                    {
                        new Vector3(-size, -size, 0),
                        new Vector3(-size, size, 0),
                        new Vector3(size, size, 0),
                        new Vector3(size, -size, 0)
                    };

                    Quaternion particleRotation =
Quaternion.Euler(particle.rotation3D);

                    _quad[0].position = pos3d + particleRotation * verts[0];
                    _quad[1].position = pos3d + particleRotation * verts[1];
                    _quad[2].position = pos3d + particleRotation * verts[2];
                    _quad[3].position = pos3d + particleRotation * verts[3];
                }
                else
                {
                    // apply rotation
                    Vector2 right = new Vector2(Mathf.Cos(rotation),
Mathf.Sin(rotation)) * size;
                    Vector2 up = new Vector2(Mathf.Cos(rotation90),
Mathf.Sin(rotation90)) * size;
                    _quad[0].position = position - right - up;
                    _quad[1].position = position - right + up;
                    _quad[2].position = position + right + up;
                    _quad[3].position = position + right - up;
                }
            }

            vh.AddUIVertexQuad(_quad);
        }
    }

    private void Update()
    {
        if (!fixedTime && Application.isPlaying)
        {
            pSystem.Simulate(Time.unscaledDeltaTime, false, false, true);
            SetAllDirty();

            if ((currentMaterial != null && currentTexture !=
currentMaterial.mainTexture) ||
                (material != null && currentMaterial != null && material.shader !=
currentMaterial.shader))
            {
                pSystem = null;
                Initialize();
            }
        }
    }

    private void LateUpdate()
    {
        if (!Application.isPlaying)
        {
            SetAllDirty();
        }
        else
        {
            if (fixedTime)
            {
                pSystem.Simulate(Time.unscaledDeltaTime, false, false, true);
                SetAllDirty();
                if ((currentMaterial != null && currentTexture !=
currentMaterial.mainTexture) ||
                    (material != null && currentMaterial != null && material.shader !=
currentMaterial.shader))
                {
                    pSystem = null;
                    Initialize();
                }
            }
        }
        if (material == currentMaterial)
            return;
        pSystem = null;
        Initialize();
    }

    protected override void OnDestroy()
    {
        currentMaterial = null;
        currentTexture = null;
    }

    public void StartParticleEmission()
    {
        pSystem.time = 0;
        pSystem.Play();
    }

    public void StopParticleEmission()
    {
        pSystem.Stop(false,
ParticleSystemStopBehavior.StopEmittingAndClear);
    }

    public void PauseParticleEmission()
    {
        pSystem.Stop(false, ParticleSystemStopBehavior.StopEmitting);
    }
  }
#endif
            }


using System;
using Scenes.Object.UI.Translater;
using TMPro;
using UnityEngine;
```

```
public class WinRecord : UIPanel
{
    [SerializeField] TMP_Text wordCounter;

    public void ShowRecord(int count)
    {
        wordCounter.text = $"<color=#FF8C00>{count}</color> " +
"Words".Localize();
    }

}



using UnityEngine;
using UnityEngine.UI;
using UnityEngine.SceneManagement;
using System.Collections;
using Unity.VisualScripting;
public class AsyncSceneLoad : MonoBehaviour
{
    [SerializeField]
    private GameObject loadingScreen;
    [SerializeField]
    private GameObject mainMenu;
    [SerializeField]
    private Slider loadingSlider;
    // Start is called once before the first execution of Update after the
MonoBehaviour is created
    void Start()
    {

    }

    // Update is called once per frame
    void Update()
    {

    }

    public void LoadLevelBtn(string levelload)
    {
        mainMenu.SetActive(false);
        loadingScreen.SetActive(true);
        StartCoroutine(LoadLevelAsync(levelload));

    }


    IEnumerator LoadLevelAsync(string levelLoad)
    {
        AsyncOperation loadOperation =
SceneManager.LoadSceneAsync(levelLoad);
        while (!loadOperation.isDone)
        {
            float progressValue = Mathf.Clamp01(loadOperation.progress / 0.9f);
            loadingSlider.value = progressValue;
            yield return null;
        }
    }


}



using UnityEngine;
using UnityEngine.UI;

// Holds serialized references for the Dictionary Menu UI.
public class DictionaryMenuRefs : MonoBehaviour
{
    [field:SerializeField] public Button BtnBack { get; private set; }
}



using System;
using TMPro;
using UnityEngine;

public class GameSessionTimer : MonoBehaviour
{

    [Header("API")]
    [SerializeField] private ApiSender apiSender;
    [SerializeField] private string endpoint = "GameSession";
    [Header("")]
    [SerializeField] private bool countWhenPaused = false; //  ,
Time.timeScale=0
    [SerializeField] private float elapsed;        //   ,
    private string TimeString => Format(elapsed);
    private float minuteBucket;   //
    private bool running = true;

    void Update()
    {
        if (!running) return;
        float dt = countWhenPaused ? Time.unscaledDeltaTime : Time.deltaTime;

        if (dt <= 0f) return;

        elapsed += dt;
        minuteBucket += dt;
        //   60
        while (minuteBucket >= 60f)
        {
            minuteBucket -= 60f;
            elapsed = 0;
            StartCoroutine(apiSender.SendSessionTime(endpoint));
        }
    }

    //   (   ,  )
    public void StartTimer() => running = true;
    public void PauseTimer() => running = false;
    public void ResetTimer()
    {
        elapsed = 0f;
        minuteBucket = 0f;
    }

    //  :.
    private static string Format(float seconds) =>
        TimeSpan.FromSeconds(seconds).ToString(@"mm\:ss\.fff",
System.Globalization.CultureInfo.InvariantCulture);
}



public static class GlobalStatic
{
    public static float masterVolume = 1.0F;
    public static float soundVolume = 0.75f;
    public static float effectVolume = 0.7f;
}



using TMPro;
using UnityEngine;
using UnityEngine.UI;

public class LeaderRow : MonoBehaviour
{
    [SerializeField] TMP_Text rank;
    [SerializeField] TMP_Text login;
    [SerializeField] TMP_Text words;
    [SerializeField] TMP_Text session;
    [SerializeField] Image placeIcon;
    [SerializeField] private Sprite[] sprites;

    public void Set(int place, string user, int w, string sess)
    {

        if (place == 1)
        {
            placeIcon.gameObject.SetActive(true);
            placeIcon.sprite = sprites[0];
        }
        else if (place == 2)
        {
            placeIcon.gameObject.SetActive(true);
            placeIcon.sprite = sprites[1];
        }
        else if (place == 3)
        {
            placeIcon.gameObject.SetActive(true);
            placeIcon.sprite = sprites[2];
        }
        else if (place > 3)
        {
           rank.text = place.ToString();
        }

        login.text = user;
```

```
        words.text = w.ToString();
        session.text = sess;
    }
}



using UnityEngine;

public class Leaders : MonoBehaviour
{
    [SerializeField] public ApiSender apiSender;
    [SerializeField] LeaderRow rowBoard;
    [SerializeField] Transform Body;

    public void GetTopPlayers()
    {
        apiSender.TopPlayers(leaders =>
        {
            //
            for (int i = Body.childCount - 1; i >= 0; i--)
                Destroy(Body.GetChild(i).gameObject);

            if (leaders == null || leaders.Length == 0)
            {
                //   /  ""
                return;
            }
            //
            for (int i = 0; i < leaders.Length; i++)
            {
                var it = leaders[i];
                var row = Instantiate(rowBoard, Body);
                row.Set(i + 1, it.login, it.words, FormatSession(it.session)); //
            }
        });
    }

    private string FormatSession(string s)
    {
        // "00:00:29.0570000" -> "00:29.057"
        if (System.TimeSpan.TryParse(s, out var ts))
            return ts.ToString(@"mm\:ss\.fff",
System.Globalization.CultureInfo.InvariantCulture);
        return s;
    }




}



using TMPro;
using UnityEngine;
using UnityEngine.UI;
public class LoadPrefs : MonoBehaviour
{

    [Header(" ")]
    [SerializeField]
    private bool canUse = false;
    [SerializeField]
    private MenuController menuController;

    [Header(" ")]
    [SerializeField] private Slider masterSlider = null;
    [SerializeField] private Slider soundSlider = null;
    [SerializeField] private Slider sfxSlider = null;


    //[Header(" ")]
    //[SerializeField] private TMP_Text brightnessTextValue = null;
    //[SerializeField] private Slider brightnessSlider = null;

    //[Header("  ")]
    //[SerializeField] private TMP_Dropdown qualityDropdown;

    //[Header(" ")]
    //[SerializeField] private Toggle fullScreenToggle;

    //[Header("")]
    //[SerializeField] private TMP_Text sensetiveTextValue = null;
    //[SerializeField] private Slider sensetiveSlider = null;


    public void LoadPrefSettings()
    {
        if (canUse)
        {
            //
            if (PlayerPrefs.HasKey("masterVolum")) GlobalStatic.masterVolume =
PlayerPrefs.GetFloat("masterVolum");
            masterSlider.value = GlobalStatic.masterVolume;
            if (PlayerPrefs.HasKey("soundVolum")) GlobalStatic.soundVolume =
PlayerPrefs.GetFloat("soundVolum");
            soundSlider.value = GlobalStatic.soundVolume;
            if (PlayerPrefs.HasKey("effectVolum")) GlobalStatic.effectVolume =
PlayerPrefs.GetFloat("effectVolum");
            sfxSlider.value = GlobalStatic.effectVolume;

        }
    }


}



using System;
using TMPro;
using UnityEngine;
using UnityEngine.UI;

// Holds serialized references for the Main Menu UI so hierarchy changes don't
break code.
public class MainMenuRefs : MonoBehaviour
{
    [field:Header("Panels & Texts")]
    [field:SerializeField] public GameObject SettingsPanel { get; private set; }
    [field:SerializeField] public TMP_Text HeaderText { get; private set; }
    [field:SerializeField] public GameObject DifficultPanel { get; private set; }

    [field:Header("Buttons")]
    [field:SerializeField] public Button BtnStart { get; private set; }
    [field:SerializeField] public Button BtnEasy { get; private set; }
    [field:SerializeField] public Button BtnMedium { get; private set; }
    [field:SerializeField] public Button BtnHard { get; private set; }
    [field:SerializeField] public Button BtnScine { get; private set; }
    [field:SerializeField] public Button BtnDick { get; private set; }
    [field:SerializeField] public Button BtnQuiz { get; private set; }

    [field:Header("Sliders")]
    [field:SerializeField] public Slider MasterSlider { get; private set; }
    [field:SerializeField] public Slider MusicSlider { get; private set; }
    [field:SerializeField] public Slider SfxSlider { get; private set; }
    [field:SerializeField] public Slider VoiceSlider { get; private set; }

    [field:Header("Toggles")]
    [field:SerializeField] public Toggle ToggleKz { get; private set; }
    [field:SerializeField] public Toggle ToggleRu { get; private set; }
    [field:SerializeField] public Button BtnStart2 { get; private set; }


    private void OnEnable()
    {
        BtnStart.onClick.AddListener(DisableButtons);
        BtnStart2.onClick.AddListener(DisableButtons);
    }

    private void DisableButtons()
    {
        BtnStart.interactable = false;
        BtnStart2.interactable = false;
    }

    public void EnableButtons()
    {
        BtnStart.interactable = true;
        BtnStart2.interactable = true;
    }

    private void OnDisable()
    {
        BtnStart.onClick.RemoveListener(DisableButtons);
        BtnStart2.onClick.RemoveListener(DisableButtons);
    }
}



using System.Collections.Generic;
using Scenes.Object.Player;
using Scenes.Quiz.Prefabs.Scripts.UI;
using TMPro;
using UnityEngine;
using UnityEngine.Audio;
```

```
using UnityEngine.UI;
public class MenuController : MonoBehaviour
{
    [Header("Контроллер игрока")]
    [SerializeField] private PlayerController playerController;
    [SerializeField] private SessionTracker sessionTracker;
    [SerializeField] private PanelQueueManager panelQueueManager;

    [SerializeField] private StartGame  startGame;

    [Header("Окна")]
    [SerializeField] private WinRecord winnerPanel;
    [SerializeField] private UIPanel registerPanel;
    [SerializeField] private NewWordsUI newWordsPanel;
    [SerializeField] private UIPanel quizUnlockPanel;


    private AudioSource mainAudioSource;

    [Header("Источник звука")]
    [SerializeField] private AudioClip[] audioClip;

    Button btnStart;

    [Header("Канвас Меню")]
    [SerializeField] public GameObject mainMenu;
    [Header("Канвас Пауза")]
    [SerializeField] private GameObject pauseMenu;
    [Header("Канвас Продолжение")]
    [SerializeField] private GameObject respawnCanvas;

    [Header("Меню Пауза")]
    private Button btnContinues;
    private Button btnPauseClose;

    [Header("Разное")]
    [SerializeField] public Transform target; // Целевая точка (позиция +
поворот)

    private void Awake()
    {
        mainAudioSource = GetComponent<AudioSource>();
        pauseMenu = Instantiate(pauseMenu);
        respawnCanvas = Instantiate(respawnCanvas);
    }

    void Start()
    {
        if (!PlayerPrefs.HasKey("IdentId"))
        {
            panelQueueManager.Enqueue(registerPanel);
        }


        btnStart =
mainMenu.transform.Find("BackGround/MainPanel/btnStart")?.gameObject.Get
Component<Button>();

        playerController.respawnCanvas = respawnCanvas;

        btnStart.onClick.RemoveAllListeners();
        btnStart.onClick.AddListener(() =>
        {
            startGame.MoveToTarget(target); // 1
            StartGame();                    // 2
        });

        var btnStart2 = mainMenu.GetComponent<MainMenuRefs>().BtnStart2;

        btnStart2.onClick.RemoveAllListeners();
        btnStart2.onClick.AddListener(() =>
        {
            startGame.MoveToTarget(target); // 1
            StartGame();                    // 2
        });
    }
    public void StopGame()
    {
        ResumeGame();
    }
    public void ExitGame()
    {
        Application.Quit();
    }


    public void PauseGame()
    {
        Time.timeScale = 0f; // ⏸ Останавливает время
        pauseMenu.SetActive(true);
        //AudioListener.pause = true; // ⏸ Приостанавливает звук
        // Показать UI меню паузы, если есть
    }
    public void ResumeGame()
    {

        Time.timeScale = 1f;
        pauseMenu.SetActive(false);
        //AudioListener.pause = false;
    }

    private void StartGame()
    {
        sessionTracker.StartTimer();
        PlayMainTheam(MainTheme.Runner);
    }


    public void PlayMainTheam(MainTheme theme)
    {
        PlayMainTheam((int)theme);
    }

    private void PlayMainTheam(int index)
    {
        //Не менять клип если уже играется нужная музяка
        if (mainAudioSource.clip == audioClip[index]) return;

        mainAudioSource.clip = audioClip[index];
        mainAudioSource.Play();
    }

    public void ShowWinnerPanel(int reccord)
    {
        winnerPanel.ShowRecord(reccord);
        panelQueueManager.Enqueue(winnerPanel);
    }

    public void ShowNewWordPanel(List<WordData> words)
    {
        newWordsPanel.SetWords(words);
        panelQueueManager.Enqueue(newWordsPanel);;
    }

    public void ShowQuizUnlockPopup()
    {
        panelQueueManager.Enqueue(quizUnlockPanel);
    }
}
public enum MainTheme
{
    Menu = 0,
    Runner = 1,
    Quiz = 2
}


using UnityEngine;
using UnityEngine.Events;

public enum LayoutTarget { Phone, Tablet};

public class MenuLayoutSwitcher : MonoBehaviour
{
    private LayoutTarget currentLayout;

    public UnityEvent<LayoutTarget> OnLayoutChanged;

    public void SwitchLayout(LayoutTarget target)
    {
        if(target == currentLayout) return;

        currentLayout = target;
        OnLayoutChanged?.Invoke(target);
    }
}


using Scenes.UI;
using UnityEngine;

namespace Scenes.Object.UI.MainMenu.Scripts
{
    public class MenuSlideAnimator : MonoBehaviour
    {
        [SerializeField] private UITweener leftTweener;
```

```
        [SerializeField] private UITweener rightTweener;

        public void DoRightOpenAnimation()
        {
            rightTweener.DoForwardAnimation();
        }

        public void DoLeftOpenAnimation()
        {
            leftTweener.DoForwardAnimation();
        }

        public void DoRightCloseAnimation()
        {
            rightTweener.DoBackwardAnimation();
        }

        public void DoLeftCloseAnimation()
        {
            leftTweener.DoBackwardAnimation();
        }
    }
}


using System.Collections;
using TMPro;
using UnityEngine;

public class PlayerRegistration : MonoBehaviour
{
    [SerializeField] private ApiSender apiSender;
    [SerializeField] TMP_InputField inputField;

    public void Registration()
    {
        StartCoroutine(RegistrationFlow());
    }

    private IEnumerator RegistrationFlow()
    {
        bool ok = false;
        yield return StartCoroutine(apiSender.CreateUser(inputField, r => ok = r));

        //if (ok)
            Destroy(gameObject);            //
        //else
        //    Debug.LogError("  ");
    }

}



using Scenes.UI;
using UnityEngine;
using UnityEngine.UI;

// Holds serialized references for the Quiz Menu UI.
public class QuizMenuRefs : MonoBehaviour
{
    [field:SerializeField] public Button BtnBack { get; private set; }
}



using System.Collections;
using UnityEngine;
using UnityEngine.UI;

public class ScinRender : MonoBehaviour
{
    public Camera renderCamera;
    public RenderTexture renderTexture;
    public RawImage[] imageSlots; // 9   UI
    public GameObject[] characterModels; // 9

    // Start is called once before the first execution of Update after the
MonoBehaviour is created
    void Start()
    {


        renderTexture.graphicsFormat =
UnityEngine.Experimental.Rendering.GraphicsFormat.R8G8B8A8_SRGB;

        StartCoroutine(RenderAllCharacters());
    }

    // Update is called once per frame
    void Update()
    {

    }


    IEnumerator RenderAllCharacters()
    {
        for (int i = 0; i < characterModels.Length; i++)
        {
            //
            for (int j = 0; j < characterModels.Length; j++)
                characterModels[j].SetActive(j == i);

            yield return new WaitForEndOfFrame();

            renderCamera.targetTexture = renderTexture;
            renderCamera.Render();

            RenderTexture.active = renderTexture;
            Texture2D tex = new Texture2D(renderTexture.width,
renderTexture.height, TextureFormat.RGBA32, false);
            tex.ReadPixels(new Rect(0, 0, renderTexture.width,
renderTexture.height), 0, 0);
            tex.Apply();

            RenderTexture.active = null;
            //renderCamera.targetTexture = null;

            imageSlots[i].texture = tex;

            yield return null;
        }

        //  3D
        foreach (var model in characterModels)
            model.SetActive(false);



    }
}



using UnityEngine;
using UnityEngine.UI;

// Holds serialized references for the Skin Menu UI.
public class SkinMenuRefs : MonoBehaviour
{
    [field:SerializeField] public Button BtnClose { get; private set; }
}



using System;
using System.Collections.Generic;
using Scenes.Object.UI.Translater;
using TMPro;
using UnityEngine;
using UnityEngine.UI;

public class TutorialPanel : MonoBehaviour
{
    [SerializeField]
    private Button skipButton;
    [SerializeField]
    private TMP_Text headerText;

    [SerializeField]
    private Color defaultColor;

    [SerializeField]
    private Color fadeColor;

    [SerializeField]
    private List<GameObject> panels;
    [SerializeField]
    private List<Image> stepImages;
    [SerializeField]
    private List<string> stepNames;

    private int currentStep = 0;

    private void Awake()
    {
        skipButton.onClick.AddListener(SkipStep);
```

```
    }

    private void SkipStep()
    {
        currentStep++;
        LoadStep(currentStep);
    }

    public void LoadStep(int step)
    {
        currentStep = step;
        headerText.text = stepNames[step].Localize();


        for (var i = 0; i < panels.Count; i++)
        {
            var panel = panels[i];
            if (i == currentStep)
            {
                panel.SetActive(true);
                continue;
            }
            panel.SetActive(false);
        }

        for (var i = 0; i < stepImages.Count; i++)
        {
            var image = stepImages[i];
            if (i == currentStep)
            {
                image.color = defaultColor;
                continue;
            }
            image.color = fadeColor;
        }
    }

    // Start is called once before the first execution of Update after the
MonoBehaviour is created
    void Start()
    {
        foreach (Image image in stepImages)
        {
            image.color = fadeColor;
        }

        foreach (GameObject panel in panels)
        {
            panel.SetActive(false);
        }
        LoadStep(currentStep);
    }
}



using UnityEngine;
using UnityEngine.EventSystems;

public class DragRotate : MonoBehaviour, IBeginDragHandler, IDragHandler,
IEndDragHandler
{
    public Transform modelToRotate;
    public float speed = 0.3f;
    public float inertia = 7f;

    float vx;

    public void OnBeginDrag(PointerEventData e) { }
    public void OnEndDrag(PointerEventData e) { }

    public void OnDrag(PointerEventData e)
    {
        vx = -e.delta.x * speed;
        modelToRotate.Rotate(Vector3.up, vx, Space.World);
    }

    void Update()
    {
        if (Mathf.Abs(vx) > 0.0001f)
        {
            modelToRotate.Rotate(Vector3.up, vx * Time.deltaTime, Space.World);
            vx = Mathf.Lerp(vx, 0f, Time.deltaTime * inertia);
        }
    }
}


using System;
using System.Collections.Generic;
using UnityEngine;

[Serializable]
public class Skin
{
    public string id;          // ►►►►►►►►►►► ID (►►►►►►►►► "skin1")
    public string displayName;
    public int price;          // ►►►►►
    public GameObject SkinObject; // ►►► ►►►►►
    public bool isPurchased = true;   // ►►►►►►► ►►
}

[Serializable]
public class SkinSaveData
{
    public int activeSkinIndex;
    public List<Skin> skins;
}



//{
//    "activeSkinIndex":0,
//        "skins":[
//        {
//            "id":"0",
//            "price":0,
//            "isPurchased":true
//        },
//    { "id":"1","price":20,"isPurchased":false},
//{ "id":"2","price":30,"isPurchased":false},
//{ "id":"3","price":40,"isPurchased":false},
//{ "id":"4","price":50,"isPurchased":false},
//{ "id":"5","price":60,"isPurchased":false},
//{ "id":"6","price":70,"isPurchased":false},
//{ "id":"7","price":80,"isPurchased":false},
//{ "id":"8","price":90,"isPurchased":false},
//{ "id":"8","price":90,"isPurchased":false}
//    ]
//}


using TMPro;
using UnityEngine;
using UnityEngine.EventSystems;
using UnityEngine.UI;

public class SkinBlock : MonoBehaviour
{
    [SerializeField] private int skinBlockIndex;
    [SerializeField] private Button btnBlock;
    [SerializeField] private GameObject selectedOverlay;
    [SerializeField] private TMP_Text txtskineStatus;
    [SerializeField] private GameObject imgAsyqPrice;

    void Start()
    {
        StatusSkine();
    }


    private void Update()
    {
        if(GlobalVar.data == null) return;

        if (GlobalVar.currScine == skinBlockIndex)
        {
            selectedOverlay.SetActive(true);
        }
        else
        {
            selectedOverlay.SetActive(false);
        }



        if (skinBlockIndex != 0 && imgAsyqPrice.activeSelf &&
GlobalVar.data.skins[skinBlockIndex].isPurchased)
        {
            txtskineStatus.gameObject.SetActive(true);
            imgAsyqPrice.SetActive(false);
        }

    }


    public void StatusSkine()
    {
```

```
        if(GlobalVar.data == null) return;

        if (GlobalVar.currScine == skinBlockIndex)
            selectedOverlay.SetActive(true);

        if (skinBlockIndex != 0 &&
GlobalVar.data.skins[skinBlockIndex].isPurchased)
        {
            txtskineStatus.gameObject.SetActive(true);
            imgAsyqPrice.SetActive(false);
        }
    }

}



using System;
using System.Collections.Generic;
using TMPro;
using UnityEngine;
using UnityEngine.Audio;
using UnityEngine.UI;
using Scenes.GameScript;
using Scenes.Object.UI.Translater;

public class SkinPreview : MonoBehaviour
{
    [SerializeField] private Transform modelToRotate;
    [SerializeField] private RawImage rawImageUI;
    [SerializeField] private float rotationSpeed = 150f;
    [SerializeField] private float inertiaDampening = 5f;
    [Header("???.????? ??????")]
    [SerializeField] private GameObject[] partsBoy ;
    [Header("?????")]
    [SerializeField] private GameObject[] scinsBoy;
    [Header("??????????")]
    [SerializeField] private Animator animator;
    [SerializeField] private PlayerController playerController;

    private int asyqCurrCount;
    //-----------???????? ? ???????????????? ????????????
??????--------------

    [Header("??????? ???? ???????")]
    [SerializeField]
    private List<Skin> skins = new List<Skin>();

    [SerializeField] private TMP_Text btnText;

    [SerializeField] private TMP_Text displayNameText;
    //[SerializeField]
    //private SkinBlock skinBlock;

    [Header("?????")]
    [SerializeField]
    private AudioSource audioSource;
    [SerializeField]
    private AudioClip[] clips;

    //-----------------------------------------------------------------

    private bool isDragging = false;
    private float currentVelocity = 0f;
    private int currScine = 0;
    private int currSelectScine = 0;
    private int selScine = 0;

    private void OnEnable()
    {
        LoadData();
    }

    private void Start()
    {
        if (PlayerPrefs.HasKey("Asyq"))
        {
            asyqCurrCount = PlayerPrefs.GetInt("Asyq");
        }
        ChangeScine(currScine);
        currSelectScine = currScine;
    }

    void Update()
    {
        if (Input.GetMouseButtonUp(0))
        {
            isDragging = false;
        }

        if (isDragging)
        {
            float mouseX = Input.GetAxis("Mouse X");
            currentVelocity = -mouseX * rotationSpeed;
        }

        if (Mathf.Abs(currentVelocity) > 0.01f)
        {
            modelToRotate.Rotate(Vector3.up, currentVelocity * Time.deltaTime,
Space.World);
            currentVelocity = Mathf.Lerp(currentVelocity, 0f, Time.deltaTime *
inertiaDampening);
        }
    }

    public void TryBuySkin()
    {
        if (selScine < 0 || selScine >= skins.Count) return;

        Skin skin = skins[selScine];
        if (skin.isPurchased)
        {
            currScine = selScine;
            GlobalVar.currScine = currScine;
            audioSource.clip = clips[0];

            SaveData();
            SkinLoader.NotifySkinSelected(currScine);

            playerController.ApplySkin(selScine);
            // Notify gameplay scene (if loaded) about the applied skin
            //return;
        }


        if (asyqCurrCount >= skin.price && !skin.isPurchased)
        {
            asyqCurrCount -= skin.price;
            PlayerPrefs.SetInt("Asyq", asyqCurrCount);
            GlobalVar.Asyq = asyqCurrCount;
            skin.isPurchased = true;
            btnText.text = "Apply".Localize();;
            audioSource.clip = clips[1];
            SaveData();
            LoadData();
            SkinLoader.NotifySkinSelected(selScine);

            playerController.ApplySkin(selScine);
        }

        audioSource.Play();

    }
    // ?????????????? ???????
    private void SaveData()
    {
        // Delegated to centralized SkinLoader
        SkinLoader.Save(skins, currScine);
    }


    //==========?????? ??????===========//
    public void ChangeScine(int index)
    {
        skins[selScine].SkinObject.SetActive(false);
        skins[index].SkinObject.SetActive(true);
        selScine = index;
        switch (index)
        {
            case 0:
                partsBoy[0].SetActive(true);
                partsBoy[1].SetActive(true);

                animator.SetTrigger("BoyStay");
                break;
            case 1:
                partsBoy[0].SetActive(true);
                partsBoy[1].SetActive(false);
                //animator.SetTrigger("StandartStay");
                animator.SetTrigger("CosmoStay");
                break;
            case 2:
                partsBoy[0].SetActive(true);
                partsBoy[1].SetActive(true);
                //animator.SetTrigger("StandartStay");
                animator.SetTrigger("BatyrStay");
                break;
            case 3:
```

```
            partsBoy[0].SetActive(true);
            partsBoy[1].SetActive(true);
            //animator.SetTrigger("StandartStay");
            animator.SetTrigger("ScientStay");
            break;
        case 4:
            partsBoy[0].SetActive(true);
            partsBoy[1].SetActive(false);
            animator.SetTrigger("CyberStay");
            break;
        case 5:
            partsBoy[0].SetActive(true);
            partsBoy[1].SetActive(true);
            animator.SetTrigger("NinjaStay");
            break;
        case 6:
            partsBoy[0].SetActive(false);
            partsBoy[1].SetActive(false);
            animator.SetTrigger("RobotStay");
            break;
        case 7:
            partsBoy[0].SetActive(false);
            partsBoy[1].SetActive(false);
            animator.SetTrigger("RockStay");
            break;
        case 8:
            partsBoy[0].SetActive(false);
            partsBoy[1].SetActive(false);
            animator.SetTrigger("FireStay");
            break;
        case 9:
            partsBoy[0].SetActive(false);
            partsBoy[1].SetActive(true);
            animator.SetTrigger("MagicStay");
            break;
        }


    displayNameText.text = skins[selScine].displayName;

    btnText.text = skins[selScine].isPurchased ? "Apply".Localize() :
"Buy".Localize();

    }
    //********** ********** *******
    private void LoadData()
    {
        // Delegated to centralized SkinLoader
        currScine = SkinLoader.LoadInto(skins);
    }

}



using System.Collections.Generic;
using System.IO;
using System.Linq;
using TMPro;
using UnityEditor;
using UnityEngine;
using UnityEngine.SceneManagement;
using Scenes.Object.UI.Translater;

namespace Scenes.Object.UI.Translater.Editor
{
    [CustomEditor(typeof(LocalizationManager))]
    public class LocalizationManagerEditor : UnityEditor.Editor
    {
        public override void OnInspectorGUI()
        {
            DrawDefaultInspector();

            EditorGUILayout.Space(8);
            EditorGUILayout.LabelField("Localization Tools",
EditorStyles.boldLabel);

            var manager = (LocalizationManager)target;
            using (new EditorGUI.DisabledScope(false))
            {
                if (GUILayout.Button("Collect Texts From Scene (Assign Keys and
Update DB)", GUILayout.Height(32)))
                {
                    CollectAndStore(manager);
                }
            }
        }

        private void CollectAndStore(LocalizationManager manager)
        {
            if (manager == null)
                return;

            // Ensure database asset exists
            if (manager.GetType().GetField("translationDatabaseAsset",
System.Reflection.BindingFlags.Instance |
System.Reflection.BindingFlags.NonPublic) == null)
            {
                EditorUtility.DisplayDialog("Localization", "Manager does not expose
translationDatabaseAsset field.", "OK");
                return;
            }

            var dbAsset = GetDatabase(manager);
            if (dbAsset == null)
            {
                if (EditorUtility.DisplayDialog("Create Database?",
                        "No TranslationDatabaseAsset assigned. Create a new one?",
                        "Create", "Cancel"))
                {
                    dbAsset = CreateDatabaseAsset();
                    if (dbAsset != null)
                    {
                        Undo.RecordObject(manager, "Assign Translation Database
Asset");
                        var soMgr = new SerializedObject(manager);

soMgr.FindProperty("translationDatabaseAsset").objectReferenceValue =
dbAsset;
                        soMgr.ApplyModifiedProperties();
                        EditorUtility.SetDirty(manager);
                    }
                }
                else return;
            }

            if (dbAsset == null) return;

            // Find all LocalizedTextElement in the active scene(s)
            var elements =
Resources.FindObjectsOfTypeAll<LocalizedTextElement>()
                .Where(e => e != null && e.gameObject.scene.IsValid() &&
(e.hideFlags & HideFlags.HideInHierarchy) == 0)
                .ToArray();

            if (elements.Length == 0)
            {
                EditorUtility.DisplayDialog("Localization", "No LocalizedTextElement
found in loaded scenes.", "OK");
                return;
            }

            int assignedKeys = 0;
            int updatedEntries = 0;
            int addedEntries = 0;

            // Prepare serialized access to database
            var so = new SerializedObject(dbAsset);
            var entriesProp = so.FindProperty("entries");

            // Build index of existing keys for quick lookup
            var keyToIndex = new Dictionary<string, int>();
            for (int i = 0; i < entriesProp.arraySize; i++)
            {
                var elemProp = entriesProp.GetArrayElementAtIndex(i);
                var keyProp = elemProp.FindPropertyRelative("key");
                if (!string.IsNullOrEmpty(keyProp.stringValue) &&
!keyToIndex.ContainsKey(keyProp.stringValue))
                    keyToIndex.Add(keyProp.stringValue, i);
            }

            // Track used keys to ensure newly generated are unique even within
this run
            var usedKeys = new HashSet<string>(keyToIndex.Keys);

            Undo.IncrementCurrentGroup();
            int undoGroup = Undo.GetCurrentGroup();

            foreach (var el in elements)
            {
                if (el == null) continue;
                string key = el.Key;
                if (string.IsNullOrEmpty(key))
                {
                    key = GenerateKey(el.gameObject, usedKeys);
                    Undo.RecordObject(el, "Assign Localization Key");
                    el.Key = key;
                    EditorUtility.SetDirty(el);
```

```
                assignedKeys++;
            }
            usedKeys.Add(key);

            // Determine text values
            string kz = el.TextKz;
            string ru = el.TextRu;
            if (string.IsNullOrEmpty(kz) && string.IsNullOrEmpty(ru))
            {
                // Fallback to current TMP text if both are empty
                var tmp = el.GetComponent<TMP_Text>();
                if (tmp != null)
                {
                    kz = tmp.text;
                    ru = tmp.text;
                }
            }

            // Upsert into database
            if (!keyToIndex.TryGetValue(key, out int idx))
            {
                idx = entriesProp.arraySize;
                entriesProp.InsertArrayElementAtIndex(idx);
                keyToIndex[key] = idx;
                addedEntries++;
            }
            else
            {
                updatedEntries++;
            }

            var entryProp = entriesProp.GetArrayElementAtIndex(idx);
            entryProp.FindPropertyRelative("key").stringValue = key;
            if (ru != null)
                entryProp.FindPropertyRelative("ruTranslation").stringValue = ru;
            if (kz != null)
                entryProp.FindPropertyRelative("kzTranslation").stringValue = kz;
        }

        so.ApplyModifiedProperties();
        EditorUtility.SetDirty(dbAsset);
        AssetDatabase.SaveAssets();

        Undo.CollapseUndoOperations(undoGroup);

        EditorUtility.DisplayDialog(
            "Localization",
            $"Processed {elements.Length} elements.\n" +
            $"Keys assigned: {assignedKeys}\n" +
            $"DB entries added: {addedEntries}\n" +
            $"DB entries updated: {updatedEntries}",
            "OK");
    }

    private static TranslationDatabaseAsset
GetDatabase(LocalizationManager manager)
    {
        var soMgr = new SerializedObject(manager);
        var prop = soMgr.FindProperty("translationDatabaseAsset");
        return prop != null ? prop.objectReferenceValue as
TranslationDatabaseAsset : null;
    }

    private static TranslationDatabaseAsset CreateDatabaseAsset()
    {
        var path = EditorUtility.SaveFilePanelInProject(
            "Create Translation Database",
            "TranslationDatabase",
            "asset",
            "Select location for the new TranslationDatabase asset.");
        if (string.IsNullOrEmpty(path)) return null;

        var asset =
ScriptableObject.CreateInstance<TranslationDatabaseAsset>();
        AssetDatabase.CreateAsset(asset, path);
        AssetDatabase.SaveAssets();
        AssetDatabase.Refresh();
        return asset;
    }

    private static string GenerateKey(GameObject go, HashSet<string> used)
    {
        string sceneName = go.scene.IsValid() ? go.scene.name : "Scene";
        string path = GetHierarchyPath(go);
        string baseKey = $"ui/{sceneName}/{path}".ToLowerInvariant();
        baseKey = Sanitize(baseKey);
        string key = baseKey;
        int i = 1;
        while (used.Contains(key))
        {
            key = baseKey + "_" + i++;
        }
        return key;
    }

    private static string GetHierarchyPath(GameObject go)
    {
        var stack = new System.Collections.Generic.Stack<string>();
        var current = go.transform;
        while (current != null)
        {
            stack.Push(current.name);
            current = current.parent;
        }
        return string.Join("/", stack);
    }

    private static string Sanitize(string s)
    {
        if (string.IsNullOrEmpty(s)) return "key";
        var chars = s.Select(c => char.IsLetterOrDigit(c) ?
char.ToLowerInvariant(c) : '_').ToArray();
        var result = new string(chars);
        while (result.Contains("__")) result = result.Replace("__", "_");
        return result.Trim('_');
    }
  }
}



using System;
using System.Net.Http;
using System.Text;
using System.Threading.Tasks;
using TMPro;
using UnityEditor;
using UnityEngine;
using UnityEngine.SceneManagement;

[CustomEditor(typeof(LocalizedTextElement))]
public class LocalizedTextElementEditor : UnityEditor.Editor
{
    private SerializedProperty _textKzProp;
    private SerializedProperty _textRuProp;
    private SerializedProperty _keyProp;
    private SerializedProperty _useKeyProp;

    private void OnEnable()
    {
        _textKzProp = serializedObject.FindProperty("textKz");
        _textRuProp = serializedObject.FindProperty("textRu");
        _keyProp = serializedObject.FindProperty("key");
        _useKeyProp = serializedObject.FindProperty("useKeyFromDatabase");
    }

    public override void OnInspectorGUI()
    {
        serializedObject.Update();

        var component = (LocalizedTextElement)target;
        var tmp = component.GetComponent<TMP_Text>();
        if (tmp == null)
        {
            EditorGUILayout.HelpBox("No TMP_Text component found on this
GameObject. 'From TMP' buttons will be disabled.", MessageType.Info);
        }

        EditorGUILayout.PropertyField(_useKeyProp, new GUIContent("Use Key
From Database"));

        EditorGUILayout.BeginHorizontal();
        EditorGUILayout.PropertyField(_keyProp, new GUIContent("Key"));
        if (GUILayout.Button(new GUIContent("Load By RU", "Load key from DB
by RU translation"), GUILayout.Width(110)))
        {
            TryLoadKeyFromDatabaseByRu();
        }
        EditorGUILayout.EndHorizontal();

        using (new EditorGUI.DisabledScope(_useKeyProp.boolValue))
        {
            DrawLocalizedField(_textKzProp, label:"Text KZ", targetLang:"kk",
sourceProp:_textRuProp, tmp, enableFromTmp: tmp != null);
            DrawLocalizedField(_textRuProp, label:"Text RU", targetLang:"ru",
sourceProp:_textKzProp, tmp, enableFromTmp: tmp != null);
        }
```

```
        EditorGUILayout.Space(6);
        if (GUILayout.Button("Apply Current Language To TMP"))
        {
            // Convenience: preview currently selected language to TMP in Editor
            if (tmp != null)
            {
                var lang = GetCurrentLang();
                if (_useKeyProp.boolValue &&
!string.IsNullOrEmpty(_keyProp.stringValue))
                {
                    // Try to fetch via manager if present in editor

                    //find manager using find object of type
                    var manager =
FindObjectOfType<Scenes.Object.UI.Translater.LocalizationManager>();

                    if (manager != null)
                    {
                        tmp.text = manager.GetTranslation(_keyProp.stringValue);
                    }
                    else
                    {
                        // Fallback just shows the key when no manager is present
                        tmp.text = _keyProp.stringValue;
                    }
                }
                else
                {
                    tmp.text = lang == 0 ? _textKzProp.stringValue :
_textRuProp.stringValue;
                }
                EditorUtility.SetDirty(tmp);
            }
        }

        serializedObject.ApplyModifiedProperties();
    }

    private int GetCurrentLang()
    {
        // Try to fetch GlobalVar.Lang via reflection (editor-time). Fallback to RU
(1)
        var globalVarType = Type.GetType("GlobalVar");
        if (globalVarType != null)
        {
            var field = globalVarType.GetField("Lang");
            if (field != null)
            {
                try
                {
                    return (int)field.GetValue(null);
                }
                catch { }
            }
        }
        return 1;
    }

    private void DrawLocalizedField(SerializedProperty prop, string label, string
targetLang, SerializedProperty sourceProp, TMP_Text tmp, bool
enableFromTmp)
    {
        EditorGUILayout.BeginHorizontal();
        EditorGUILayout.PropertyField(prop, new GUIContent(label));

        using (new EditorGUI.DisabledScope(!enableFromTmp))
        {
            if (GUILayout.Button(new GUIContent("⤓", "Pull from TMP_Text"),
GUILayout.Width(26)))
            {
                Undo.RecordObject(serializedObject.targetObject, "Pull Localized
Text From TMP");
                prop.stringValue = tmp != null ? tmp.text : prop.stringValue;
                EditorUtility.SetDirty(serializedObject.targetObject);
            }
        }

        if (GUILayout.Button(new GUIContent("🌐", "Translate using Google
Translate"), GUILayout.Width(26)))
        {
            // Determine source text: prefer other field, fallback to current, then TMP
            var sourceText = !string.IsNullOrWhiteSpace(sourceProp.stringValue)
                ? sourceProp.stringValue
                : (!string.IsNullOrWhiteSpace(prop.stringValue) ? prop.stringValue :
(tmp != null ? tmp.text : string.Empty));

            TranslateAndAssign(sourceText, prop, targetLang);
        }
        EditorGUILayout.EndHorizontal();
    }

    private async void TranslateAndAssign(string sourceText, SerializedProperty
targetProp, string targetLang)
    {
        if (string.IsNullOrWhiteSpace(sourceText))
        {
            EditorUtility.DisplayDialog("Translate", "No source text available to
translate.", "OK");
            return;
        }

        try
        {
            var translated = await GoogleTranslate(sourceText, targetLang);
            if (!string.IsNullOrEmpty(translated))
            {
                Undo.RecordObject(serializedObject.targetObject, "Translate
Localized Text");
                // "Аударма Перевод offline f7be0b30c542fa2b03d3c167cd5da9d4
offline_launch_doc.md ru ru ru"
                //clear the tanslated string

                string[] split = translated.Split(' ');
                translated = split[0];

                targetProp.stringValue = translated;
                serializedObject.ApplyModifiedProperties();
                EditorUtility.SetDirty(serializedObject.targetObject);
                Repaint();
            }
            else
            {
                EditorUtility.DisplayDialog("Translate", "Failed to translate text.",
"OK");
            }
        }
        catch (Exception ex)
        {
            Debug.LogError($"[LocalizedTextElementEditor] Translation error:
{ex}");
            EditorUtility.DisplayDialog("Translate", "Translation error. See Console
for details.", "OK");
        }
    }

    // Simple, unofficial Google Translate endpoint (no API key). For editor tooling
only.
    private static async Task<string> GoogleTranslate(string text, string
targetLang)
    {
        if (string.IsNullOrEmpty(text)) return text;

        // Use 'auto' for source language detection
        var url =
$"https://translate.googleapis.com/translate_a/single?client=gtx&sl=auto&tl={tar
getLang}&dt=t&q={Uri.EscapeDataString(text)}";

        using (var client = new HttpClient())
        {
            client.DefaultRequestHeaders.Add("User-Agent", "UnityEditor-
LocalizedTextElement");
            var resp = await client.GetAsync(url);
            resp.EnsureSuccessStatusCode();
            var body = await resp.Content.ReadAsStringAsync();
            return ParseGoogleBody(body);
        }
    }

    // Very naive parser for the translate.googleapis.com response
    private static string ParseGoogleBody(string body)
    {
        // Expected structure: [[["Translated phrase","Original",...], [..]], ...]
        // We'll extract all first elements of inner arrays and concatenate
        if (string.IsNullOrEmpty(body)) return string.Empty;
        var sb = new StringBuilder();
        int i = 0;
        while (i < body.Length)
        {
            // look for opening quote
            if (body[i] == '\"')
            {
                int j = i + 1;
                var seg = new StringBuilder();
                bool escaped = false;
                while (j < body.Length)
                {
                    var c = body[j++];
```

```
                if (escaped)
                {
                    switch (c)
                    {
                        case 'n': seg.Append('\n'); break;
                        case 'r': seg.Append('\r'); break;
                        case 't': seg.Append('\t'); break;
                        case '"': seg.Append('"'); break;
                        case '\\': seg.Append('\\'); break;
                        default: seg.Append(c); break;
                    }
                    escaped = false;
                    continue;
                }
                if (c == '\\') { escaped = true; continue; }
                if (c == '\"') break;
                seg.Append(c);
            }
            // Append first string then skip to next `],` which ends the segment
            if (seg.Length > 0)
            {
                if (sb.Length > 0) sb.Append(' ');
                sb.Append(seg);
            }
            i = j;
        }
        else i++;
    }
    return sb.ToString().Trim();
}
private void TryLoadKeyFromDatabaseByRu()
{
    var ruText = _textRuProp != null ? _textRuProp.stringValue : string.Empty;
    if (string.IsNullOrWhiteSpace(ruText))
    {
        EditorUtility.DisplayDialog("Load Key", "RU text is empty. Please fill
'Text RU' first.", "OK");
        return;
    }

    var db = GetTranslationDatabaseFromManager();
    if (db == null)
    {
        EditorUtility.DisplayDialog("Load Key", "No LocalizationManager in
scene or no TranslationDatabase assigned.", "OK");
        return;
    }

    // Search via SerializedObject to access private fields of TranslationEntry
    var so = new SerializedObject(db);
    var entriesProp = so.FindProperty("entries");
    if (entriesProp == null || entriesProp.arraySize == 0)
    {
        EditorUtility.DisplayDialog("Load Key", "Translation database is empty.",
"OK");
        return;
    }

    string matchKey = null;
    int matches = 0;
    string needle = ruText.Trim();

    for (int i = 0; i < entriesProp.arraySize; i++)
    {
        var entryProp = entriesProp.GetArrayElementAtIndex(i);
        var ruProp = entryProp.FindPropertyRelative("ruTranslation");
        if (ruProp != null && string.Equals(ruProp.stringValue?.Trim(), needle,
System.StringComparison.Ordinal))
        {
            var keyProp = entryProp.FindPropertyRelative("key");
            if (!string.IsNullOrEmpty(keyProp.stringValue))
            {
                matchKey = keyProp.stringValue;
                matches++;
            }
        }
    }

    if (matches == 0)
    {
        EditorUtility.DisplayDialog("Load Key", "No entries with matching RU
translation found.", "OK");
        return;
    }
    if (matches > 1)
    {
        EditorUtility.DisplayDialog("Load Key", $"Found {matches} entries with
the same RU text. The first match will be used: {matchKey}", "OK");
    }
    Undo.RecordObject(serializedObject.targetObject, "Assign Localization
Key From DB");
    _keyProp.stringValue = matchKey;
    // Also enable using the DB key for convenience
    _useKeyProp.boolValue = true;
    serializedObject.ApplyModifiedProperties();
    EditorUtility.SetDirty(serializedObject.targetObject);
    Repaint();
}

private Scenes.Object.UI.Translater.TranslationDatabaseAsset
GetTranslationDatabaseFromManager()
{
    var manager = Scenes.Object.UI.Translater.LocalizationManager.Instance;
    if (manager == null) return null;
    var soMgr = new SerializedObject(manager);
    var dbProp = soMgr.FindProperty("translationDatabaseAsset");
    return dbProp != null ? dbProp.objectReferenceValue as
Scenes.Object.UI.Translater.TranslationDatabaseAsset : null;
}
}


using System;
using System.Net.Http;
using System.Text;
using System.Threading.Tasks;
using TMPro;
using UnityEditor;
using UnityEngine;

public static class TMPTextLocalizedContext
{
    [MenuItem("CONTEXT/TMP_Text/Localize")]
    private static void Localize(MenuCommand command)
    {
        var tmp = command.context as TMP_Text;
        if (tmp == null) return;

        var go = tmp.gameObject;

        Undo.IncrementCurrentGroup();
        int group = Undo.GetCurrentGroup();

        var lte = go.GetComponent<LocalizedTextElement>();
        if (lte == null)
        {
            lte = Undo.AddComponent<LocalizedTextElement>(go);
        }

        // Set serialized fields via SerializedObject (they're private [SerializeField])
        var so = new SerializedObject(lte);
        var textRuProp = so.FindProperty("textRu");
        var textKzProp = so.FindProperty("textKz");

        string sourceRu = tmp.text ?? string.Empty;
        if (textRuProp != null)
        {
            textRuProp.stringValue = sourceRu;
        }
        so.ApplyModifiedPropertiesWithoutUndo();
        EditorUtility.SetDirty(lte);

        // Start async translation RU -> KZ
        TranslateAndApplyRUtoKZ(sourceRu, so, textKzProp, lte, tmp).Forget();

        Undo.CollapseUndoOperations(group);

        // Feedback
        ShowSceneViewNotification(tmp, "Localized: Added component and
started RU→KZ translation...");
    }

    private static void ShowSceneViewNotification(TMP_Text tmp, string msg)
    {
        if (SceneView.lastActiveSceneView != null)
            SceneView.lastActiveSceneView.ShowNotification(new
GUIContent(msg));
        else
            EditorUtility.DisplayDialog("Localized", msg, "OK");
    }

    private static async Task TranslateAndApplyRUtoKZ(string ruText,
SerializedObject so, SerializedProperty textKzProp, LocalizedTextElement lte,
TMP_Text tmp)
    {
        if (string.IsNullOrWhiteSpace(ruText))
        {
```

```
            ShowSceneViewNotification(tmp, "Localized: No source text to
translate.");
            return;
        }

        try
        {
            var kz = await GoogleTranslate(ruText, "ru", "kk");
            //clear the string

            string[] split = kz.Split(' ');
            kz = split[0];

            if (!string.IsNullOrEmpty(kz))
            {
                so.Update();
                if (textKzProp != null)
                {
                    textKzProp.stringValue = kz;
                }
                so.ApplyModifiedProperties();
                EditorUtility.SetDirty(lte);
                ShowSceneViewNotification(tmp, "Localized: Translation completed
(RU→KZ).");
            }
            else
            {
                ShowSceneViewNotification(tmp, "Localized: Translation failed.");
            }
        }
        catch (Exception ex)
        {
            Debug.LogError("[TMPTextLocalizedContext] Translation error: " + ex);
            ShowSceneViewNotification(tmp, "Localized: Translation error. See
Console.");
        }
    }

    // Unofficial Google Translate endpoint; for editor tooling only.
    private static async Task<string> GoogleTranslate(string text, string
sourceLang, string targetLang)
    {
        if (string.IsNullOrEmpty(text)) return text;

        var url =
$"https://translate.googleapis.com/translate_a/single?client=gtx&sl={sourceLan
g}&tl={targetLang}&dt=t&q={Uri.EscapeDataString(text)}";
        using (var client = new HttpClient())
        {
            client.DefaultRequestHeaders.Add("User-Agent", "UnityEditor-
TMPTextLocalizedContext");
            var resp = await client.GetAsync(url);
            resp.EnsureSuccessStatusCode();
            var body = await resp.Content.ReadAsStringAsync();
            return ParseGoogleBody(body);
        }
    }

    // Naive JSON-ish parser matching translate.googleapis.com shape
    private static string ParseGoogleBody(string body)
    {
        if (string.IsNullOrEmpty(body)) return string.Empty;
        var sb = new StringBuilder();
        int i = 0;
        while (i < body.Length)
        {
            if (body[i] == '"')
            {
                int j = i + 1;
                var seg = new StringBuilder();
                bool escaped = false;
                while (j < body.Length)
                {
                    var c = body[j++];
                    if (escaped)
                    {
                        switch (c)
                        {
                            case 'n': seg.Append('\n'); break;
                            case 'r': seg.Append('\r'); break;
                            case 't': seg.Append('\t'); break;
                            case '"': seg.Append('"'); break;
                            case '\\': seg.Append('\\'); break;
                            default: seg.Append(c); break;
                        }
                        escaped = false;
                        continue;
                    }
                    if (c == '\\') { escaped = true; continue; }
                    if (c == '"') break;
                    seg.Append(c);
                }
                if (seg.Length > 0)
                {
                    if (sb.Length > 0) sb.Append(' ');
                    sb.Append(seg);
                }
                i = j;
            }
            else i++;
        }
        return sb.ToString().Trim();
    }
}

// Helper to intentionally ignore Task without warnings in editor scripts
internal static class TaskExtensions_TMPTextLocalizedContext
{
    public static void Forget(this Task task) { }
}
```

```
using System;
using System.Net.Http;
using System.Text;
using System.Threading.Tasks;
using System.IO;
using UnityEditor;
using UnityEngine;
using Scenes.Object.UI.Translater;

[CustomEditor(typeof(TranslationDatabaseAsset))]
public class TranslationDatabaseAssetEditor : UnityEditor.Editor
{
    private SerializedProperty _entriesProp;
    private Vector2 _scroll;

    private void OnEnable()
    {
        _entriesProp = serializedObject.FindProperty("entries");
    }

    public override void OnInspectorGUI()
    {
        serializedObject.Update();

        DrawToolbar();
        EditorGUILayout.Space(4);

        // Table Header
        using (new EditorGUILayout.HorizontalScope(EditorStyles.helpBox))
        {
            GUILayout.Label("#", GUILayout.Width(24));
            GUILayout.Label("Key", GUILayout.MinWidth(120));
            GUILayout.Label("RU", GUILayout.MinWidth(160));
            GUILayout.Label("KZ", GUILayout.MinWidth(160));
            GUILayout.Label("Actions", GUILayout.Width(160));
        }

        _scroll = EditorGUILayout.BeginScrollView(_scroll,
GUILayout.MaxHeight(420));
        if (_entriesProp != null)
        {
            for (int i = 0; i < _entriesProp.arraySize; i++)
            {
                var element = _entriesProp.GetArrayElementAtIndex(i);
                var keyProp = element.FindPropertyRelative("key");
                var ruProp = element.FindPropertyRelative("ruTranslation");
                var kzProp = element.FindPropertyRelative("kzTranslation");

                using (new EditorGUILayout.HorizontalScope())
                {
                    GUILayout.Label(i.ToString(), GUILayout.Width(24));

                    EditorGUILayout.PropertyField(keyProp, GUIContent.none,
GUILayout.MinWidth(120));
                    EditorGUILayout.PropertyField(ruProp, GUIContent.none,
GUILayout.MinWidth(160));
                    EditorGUILayout.PropertyField(kzProp, GUIContent.none,
GUILayout.MinWidth(160));

                    using (new
EditorGUILayout.HorizontalScope(GUILayout.Width(160)))
                    {
                        if (GUILayout.Button(new GUIContent("RU ← KZ", "Translate
RU from KZ using Google Translate"), GUILayout.Width(76)))
                        {
```

```
                    var source = kzProp.stringValue;
                    TranslateAndAssign(source, ruProp, targetLang: "ru");
                }
                if (GUILayout.Button(new GUIContent("KZ ← RU", "Translate KZ
from RU using Google Translate"), GUILayout.Width(76)))
                {
                    var source = ruProp.stringValue;
                    TranslateAndAssign(source, kzProp, targetLang: "kk");
                }
            }

            if (GUILayout.Button("✕", GUILayout.Width(22)))
            {
                Undo.RecordObject(serializedObject.targetObject, "Remove
Translation Entry");
                _entriesProp.DeleteArrayElementAtIndex(i);
                EditorUtility.SetDirty(serializedObject.targetObject);
                break; // layout changed
            }
        }

        // Thin separator
        EditorGUILayout.LabelField(GUIContent.none,
GUI.skin.horizontalSlider);
    }
}
EditorGUILayout.EndScrollView();

EditorGUILayout.Space(6);
using (new EditorGUILayout.HorizontalScope())
{
    if (GUILayout.Button("Add Entry"))
    {
        Undo.RecordObject(serializedObject.targetObject, "Add Translation
Entry");
        int index = _entriesProp.arraySize;
        _entriesProp.InsertArrayElementAtIndex(index);
        var el = _entriesProp.GetArrayElementAtIndex(index);
        el.FindPropertyRelative("key").stringValue = string.Empty;
        el.FindPropertyRelative("ruTranslation").stringValue = string.Empty;
        el.FindPropertyRelative("kzTranslation").stringValue = string.Empty;
        EditorUtility.SetDirty(serializedObject.targetObject);
    }
    if (GUILayout.Button("Sort by Key"))
    {
        SortByKey();
    }
}

serializedObject.ApplyModifiedProperties();
}

private void DrawToolbar()
{
    using (new EditorGUILayout.HorizontalScope())
    {
        if (GUILayout.Button(new GUIContent("Export Excel", "Export all entries
to an Excel (.xls) file (Key, RU, KZ)"), GUILayout.Width(100)))
        {
            ExportExcel();
        }
        if (GUILayout.Button(new GUIContent("Import Excel", "Import entries
from an Excel (.xls) file exported by this tool (Key, RU, KZ)"),
GUILayout.Width(100)))
        {
            ImportExcel();
        }
        GUILayout.FlexibleSpace();
        if (GUILayout.Button(new GUIContent("Translate Empty RU from KZ",
"For all entries where RU is empty and KZ is not, fill RU using Google
Translate")))
        {
            BulkTranslate(emptyTarget: "ruTranslation", source: "kzTranslation",
targetLang: "ru");
        }
        if (GUILayout.Button(new GUIContent("Translate Empty KZ from RU",
"For all entries where KZ is empty and RU is not, fill KZ using Google
Translate")))
        {
            BulkTranslate(emptyTarget: "kzTranslation", source: "ruTranslation",
targetLang: "kk");
        }
    }
}

private void SortByKey()
{
    // Because entries are private fields we sort via serialized array read/write
    if (_entriesProp == null) return;
    var count = _entriesProp.arraySize;
    var list = new System.Collections.Generic.List<(string key, string ru, string
kz)>();
    for (int i = 0; i < count; i++)
    {
        var el = _entriesProp.GetArrayElementAtIndex(i);
        list.Add((
            el.FindPropertyRelative("key").stringValue,
            el.FindPropertyRelative("ruTranslation").stringValue,
            el.FindPropertyRelative("kzTranslation").stringValue
        ));
    }
    list.Sort((a, b) => string.Compare(a.key, b.key,
StringComparison.Ordinal));

    Undo.RecordObject(serializedObject.targetObject, "Sort Translation
Entries");
    _entriesProp.arraySize = 0;
    for (int i = 0; i < list.Count; i++)
    {
        _entriesProp.InsertArrayElementAtIndex(i);
        var el = _entriesProp.GetArrayElementAtIndex(i);
        el.FindPropertyRelative("key").stringValue = list[i].key;
        el.FindPropertyRelative("ruTranslation").stringValue = list[i].ru;
        el.FindPropertyRelative("kzTranslation").stringValue = list[i].kz;
    }
    serializedObject.ApplyModifiedProperties();
    EditorUtility.SetDirty(serializedObject.targetObject);
}

private async void BulkTranslate(string emptyTarget, string source, string
targetLang)
{
    if (_entriesProp == null) return;
    try
    {
        for (int i = 0; i < _entriesProp.arraySize; i++)
        {
            var el = _entriesProp.GetArrayElementAtIndex(i);
            var targetProp = el.FindPropertyRelative(emptyTarget);
            var sourceProp = el.FindPropertyRelative(source);
            if (string.IsNullOrWhiteSpace(targetProp.stringValue) &&
!string.IsNullOrWhiteSpace(sourceProp.stringValue))
            {
                var translated = await GoogleTranslate(sourceProp.stringValue,
targetLang);
                translated = CleanupTranslated(translated);
                targetProp.stringValue = translated;
            }
        }
        serializedObject.ApplyModifiedProperties();
        EditorUtility.SetDirty(serializedObject.targetObject);
        Repaint();
    }
    catch (Exception ex)
    {
        Debug.LogError($"[TranslationDatabaseAssetEditor] Bulk translate
error: {ex}");
        EditorUtility.DisplayDialog("Translate", "Bulk translation error. See
Console for details.", "OK");
    }
}

private async void TranslateAndAssign(string sourceText, SerializedProperty
targetProp, string targetLang)
{
    if (string.IsNullOrWhiteSpace(sourceText))
    {
        EditorUtility.DisplayDialog("Translate", "No source text to translate.",
"OK");
        return;
    }
    try
    {
        var translated = await GoogleTranslate(sourceText, targetLang);
        translated = CleanupTranslated(translated);

        Undo.RecordObject(serializedObject.targetObject, "Translate Entry");
        targetProp.stringValue = translated;
        serializedObject.ApplyModifiedProperties();
        EditorUtility.SetDirty(serializedObject.targetObject);
        Repaint();
    }
    catch (Exception ex)
    {
        Debug.LogError($"[TranslationDatabaseAssetEditor] Translation error:
{ex}");
        EditorUtility.DisplayDialog("Translate", "Translation error. See Console
for details.", "OK");
```

```
        }
    }

    private static string CleanupTranslated(string translated)
    {
        if (string.IsNullOrEmpty(translated)) return translated;
        // In case unwanted suffixes are returned (rare), keep it simple and trim
whitespace.
        // The previous editor had a bug splitting by space and taking only the first
word. Avoid that here.
        return translated.Trim();
    }

    private static async Task<string> GoogleTranslate(string text, string
targetLang)
    {
        if (string.IsNullOrEmpty(text)) return text;
        var url =
$"https://translate.googleapis.com/translate_a/single?client=gtx&sl=auto&tl={tar
getLang}&dt=t&q={Uri.EscapeDataString(text)}";
        using (var client = new HttpClient())
        {
            client.DefaultRequestHeaders.Add("User-Agent", "UnityEditor-
TranslationDB");
            var resp = await client.GetAsync(url);
            resp.EnsureSuccessStatusCode();
            var body = await resp.Content.ReadAsStringAsync();
            return ParseGoogleBody(body);
        }
    }

    private static string ParseGoogleBody(string body)
    {
        if (string.IsNullOrEmpty(body)) return string.Empty;
        var sb = new StringBuilder();
        int i = 0;
        bool firstStringInTuple = true;
        int quoteCountInTuple = 0;
        while (i < body.Length)
        {
            if (body[i] == '"')
            {
                int j = i + 1;
                var seg = new StringBuilder();
                bool escaped = false;
                while (j < body.Length)
                {
                    var c = body[j++];
                    if (escaped)
                    {
                        switch (c)
                        {
                            case 'n': seg.Append('\n'); break;
                            case 'r': seg.Append('\r'); break;
                            case 't': seg.Append('\t'); break;
                            case '"': seg.Append('"'); break;
                            case '\\': seg.Append('\\'); break;
                            default: seg.Append(c); break;
                        }
                        escaped = false;
                        continue;
                    }
                    if (c == '\\') { escaped = true; continue; }
                    if (c == '"') break;
                    seg.Append(c);
                }
                // The API returns repeating tuples like
["translated","original",null,null,...]
                // We take every first string in such tuples which correspond to
translated parts.
                if (firstStringInTuple && seg.Length > 0)
                {
                    if (sb.Length > 0) sb.Append(' ');
                    sb.Append(seg);
                }
                quoteCountInTuple++;
                if (quoteCountInTuple >= 2)
                {
                    // Reset at end of pair (translated, original)
                    firstStringInTuple = true;
                    quoteCountInTuple = 0;
                }
                else
                {
                    firstStringInTuple = false;
                }
                i = j;
            }
            else i++;
        }
        return sb.ToString().Trim();
    }

    private void ExportCsv()
    {
        if (_entriesProp == null) return;
        var path = EditorUtility.SaveFilePanel("Export Translations CSV",
Application.dataPath, "translations.csv", "csv");
        if (string.IsNullOrEmpty(path)) return;
        var sb = new StringBuilder();
        sb.AppendLine("Key,RU,KZ");
        for (int i = 0; i < _entriesProp.arraySize; i++)
        {
            var el = _entriesProp.GetArrayElementAtIndex(i);
            var key = el.FindPropertyRelative("key").stringValue ?? string.Empty;
            var ru = el.FindPropertyRelative("ruTranslation").stringValue ??
string.Empty;
            var kz = el.FindPropertyRelative("kzTranslation").stringValue ??
string.Empty;

sb.Append(EscapeCsv(key)).Append(',').Append(EscapeCsv(ru)).Append(',').Ap
pend(EscapeCsv(kz)).Append('\n');
        }
        try
        {
            File.WriteAllText(path, sb.ToString(), new UTF8Encoding(true));
            EditorUtility.DisplayDialog("Export CSV", $"Exported
{_entriesProp.arraySize} entries to:\n{path}", "OK");
        }
        catch (Exception ex)
        {
            Debug.LogError($"[TranslationDatabaseAssetEditor] Export CSV error:
{ex}");
            EditorUtility.DisplayDialog("Export CSV", "Failed to write file. See
Console for details.", "OK");
        }
    }

    private void ExportExcel()
    {
        if (_entriesProp == null) return;
        var path = EditorUtility.SaveFilePanel("Export Translations Excel",
Application.dataPath, "translations.xls", "xls");
        if (string.IsNullOrEmpty(path)) return;
        var sb = new StringBuilder();
        sb.Append("<html><head><meta charset=\"utf-8\"></head><body>");
        sb.Append("<table border='1' cellspacing='0' cellpadding='3'>");

sb.Append("<thead><tr><th>Key</th><th>RU</th><th>KZ</th></tr></thead><t
body>");
        for (int i = 0; i < _entriesProp.arraySize; i++)
        {
            var el = _entriesProp.GetArrayElementAtIndex(i);
            var key = el.FindPropertyRelative("key").stringValue ?? string.Empty;
            var ru = el.FindPropertyRelative("ruTranslation").stringValue ??
string.Empty;
            var kz = el.FindPropertyRelative("kzTranslation").stringValue ??
string.Empty;
            // Replace newlines with <br> for readability in Excel and escape HTML
            string Cell(string s)
            {
                if (string.IsNullOrEmpty(s)) return string.Empty;
                return EscapeHtml(s).Replace("\r\n", "<br>").Replace("\n", "<br>");
            }
            sb.Append("<tr>");
            sb.Append("<td>").Append(Cell(key)).Append("</td>");
            sb.Append("<td>").Append(Cell(ru)).Append("</td>");
            sb.Append("<td>").Append(Cell(kz)).Append("</td>");
            sb.Append("</tr>");
        }
        sb.Append("</tbody></table></body></html>");
        try
        {
            File.WriteAllText(path, sb.ToString(), new UTF8Encoding(true));
            EditorUtility.DisplayDialog("Export Excel", $"Exported
{_entriesProp.arraySize} entries to:\n{path}", "OK");
        }
        catch (Exception ex)
        {
            Debug.LogError($"[TranslationDatabaseAssetEditor] Export Excel error:
{ex}");
            EditorUtility.DisplayDialog("Export Excel", "Failed to write file. See
Console for details.", "OK");
        }
    }

    private static string EscapeHtml(string s)
    {
```

```
        if (string.IsNullOrEmpty(s)) return string.Empty;
        var sb = new StringBuilder(s.Length);
        foreach (var ch in s)
        {
            switch (ch)
            {
                case '<': sb.Append("&lt;"); break;
                case '>': sb.Append("&gt;"); break;
                case '&': sb.Append("&amp;"); break;
                case '"': sb.Append("&quot;"); break;
                case '\'': sb.Append("&#39;"); break;
                default: sb.Append(ch); break;
            }
        }
        return sb.ToString();
    }

    private void ImportExcel()
    {
        if (_entriesProp == null) return;
        var path = EditorUtility.OpenFilePanel("Import Translations Excel",
Application.dataPath, "xls");
        if (string.IsNullOrEmpty(path)) return;
        string html;
        try
        {
            html = File.ReadAllText(path, Encoding.UTF8);
        }
        catch (Exception ex)
        {
            Debug.LogError($"[TranslationDatabaseAssetEditor] Import Excel read
error: {ex}");
            EditorUtility.DisplayDialog("Import Excel", "Failed to read file. See
Console for details.", "OK");
            return;
        }
        if (string.IsNullOrEmpty(html))
        {
            EditorUtility.DisplayDialog("Import Excel", "File is empty.", "OK");
            return;
        }

        // Parse rows and cells from the simple HTML table we generate in
ExportExcel
        var rows = new
System.Collections.Generic.List<System.Collections.Generic.List<string>>();
        string lower = html.ToLowerInvariant();
        int pos = 0;
        while (true)
        {
            int trOpen = lower.IndexOf("<tr", pos, StringComparison.Ordinal);
            if (trOpen < 0) break;
            int trOpenEnd = lower.IndexOf('>', trOpen);
            if (trOpenEnd < 0) break;
            int trClose = lower.IndexOf("</tr>", trOpenEnd + 1,
StringComparison.Ordinal);
            if (trClose < 0) break;
            string rowHtml = html.Substring(trOpenEnd + 1, trClose - (trOpenEnd +
1));
            string rowLower = lower.Substring(trOpenEnd + 1, trClose - (trOpenEnd
+ 1));

            var cells = new System.Collections.Generic.List<string>();
            int rp = 0;
            while (rp < rowLower.Length)
            {
                // search for td or th
                int td = rowLower.IndexOf("<td", rp, StringComparison.Ordinal);
                int th = rowLower.IndexOf("<th", rp, StringComparison.Ordinal);
                int start = -1;
                bool isTh = false;
                if (td == -1 && th == -1) break;
                if (td == -1 || (th != -1 && th < td)) { start = th; isTh = true; }
                else { start = td; isTh = false; }
                int openEnd = rowLower.IndexOf('>', start);
                if (openEnd == -1) break;
                string closeTag = isTh ? "</th>" : "</td>";
                int close = rowLower.IndexOf(closeTag, openEnd + 1,
StringComparison.Ordinal);
                if (close == -1) break;
                // Extract inner HTML between the opening and closing cell tags using
row-relative indices
                int cellStart = openEnd + 1;   // start of inner content within the row
                int cellLen = close - cellStart;
                if (cellStart < 0) cellStart = 0;
                if (cellLen < 0) cellLen = 0;
                if (cellStart + cellLen > rowHtml.Length)
                {
                    cellLen = Math.Max(0, rowHtml.Length - cellStart);
                }
                string cellHtml = rowHtml.Substring(cellStart, cellLen);

                // Normalize <br> to new lines (handle common variants)
                string normalized = cellHtml
                    .Replace("<br>", "\n").Replace("<br/>", "\n").Replace("<br />", "\n")
                    .Replace("<BR>", "\n").Replace("<BR/>", "\n").Replace("<BR />",
"\n");
                // Strip remaining tags
                string plain = StripHtmlTags(normalized);
                // Unescape HTML entities
                plain = UnescapeHtml(plain);
                cells.Add(plain);
                rp = close + closeTag.Length;
            }
            rows.Add(cells);
            pos = trClose + 5;
        }

        if (rows.Count == 0)
        {
            EditorUtility.DisplayDialog("Import Excel", "No rows found in the file.",
"OK");
            return;
        }

        int startRow = 0;
        if (rows[0].Count >= 3 &&
            string.Equals(rows[0][0], "Key", StringComparison.OrdinalIgnoreCase)
&&
            string.Equals(rows[0][1], "RU", StringComparison.OrdinalIgnoreCase)
&&
            string.Equals(rows[0][2], "KZ", StringComparison.OrdinalIgnoreCase))
        {
            startRow = 1;
        }

        var map = new System.Collections.Generic.Dictionary<string,
int>(StringComparer.Ordinal);
        for (int i = 0; i < _entriesProp.arraySize; i++)
        {
            var el = _entriesProp.GetArrayElementAtIndex(i);
            var key = el.FindPropertyRelative("key").stringValue ?? string.Empty;
            if (!map.ContainsKey(key)) map[key] = i;
        }

        int updated = 0, added = 0, skipped = 0;
        Undo.RecordObject(serializedObject.targetObject, "Import Translation
Excel");
        for (int r = startRow; r < rows.Count; r++)
        {
            var cols = rows[r];
            if (cols == null || cols.Count == 0) { skipped++; continue; }
            var key = (cols.Count > 0 ? cols[0] : string.Empty)?.Trim();
            if (string.IsNullOrEmpty(key)) { skipped++; continue; }
            var ru = cols.Count > 1 ? cols[1] : string.Empty;
            var kz = cols.Count > 2 ? cols[2] : string.Empty;

            if (map.TryGetValue(key, out var idx))
            {
                var el = _entriesProp.GetArrayElementAtIndex(idx);
                el.FindPropertyRelative("ruTranslation").stringValue = ru;
                el.FindPropertyRelative("kzTranslation").stringValue = kz;
                updated++;
            }
            else
            {
                int index = _entriesProp.arraySize;
                _entriesProp.InsertArrayElementAtIndex(index);
                var el = _entriesProp.GetArrayElementAtIndex(index);
                el.FindPropertyRelative("key").stringValue = key;
                el.FindPropertyRelative("ruTranslation").stringValue = ru;
                el.FindPropertyRelative("kzTranslation").stringValue = kz;
                map[key] = index;
                added++;
            }
        }
        serializedObject.ApplyModifiedProperties();
        EditorUtility.SetDirty(serializedObject.targetObject);
        Repaint();
        EditorUtility.DisplayDialog("Import Excel", $"Import finished. Updated:
{updated}, Added: {added}, Skipped: {skipped}.", "OK");
    }

    private static string StripHtmlTags(string s)
    {
        if (string.IsNullOrEmpty(s)) return string.Empty;
        var sb = new StringBuilder(s.Length);
        bool inside = false;
```

```
        for (int i = 0; i < s.Length; i++)
        {
            char c = s[i];
            if (c == '<') { inside = true; continue; }
            if (c == '>') { inside = false; continue; }
            if (!inside) sb.Append(c);
        }
        return sb.ToString();
    }

    private static string UnescapeHtml(string s)
    {
        if (string.IsNullOrEmpty(s)) return string.Empty;
        var sb = new StringBuilder(s.Length);
        for (int i = 0; i < s.Length; i++)
        {
            char c = s[i];
            if (c == '&')
            {
                int semi = s.IndexOf(';', i + 1);
                if (semi > i + 1)
                {
                    string entity = s.Substring(i + 1, semi - i - 1);
                    string repl = null;
                    switch (entity)
                    {
                        case "lt": repl = "<"; break;
                        case "gt": repl = ">"; break;
                        case "amp": repl = "&"; break;
                        case "quot": repl = "\""; break;
                        case "apos": repl = "'"; break;
                        case "#39": repl = "'"; break;
                        default:
                            if (entity.StartsWith("#x",
StringComparison.OrdinalIgnoreCase))
                            {
                                var hex = entity.Substring(2);
                                try { repl = char.ConvertFromUtf32(Convert.ToInt32(hex,
16)); } catch { repl = null; }
                            }
                            else if (entity.StartsWith("#"))
                            {
                                var dec = entity.Substring(1);
                                try { repl = char.ConvertFromUtf32(Convert.ToInt32(dec,
10)); } catch { repl = null; }
                            }
                            break;
                    }
                    if (repl != null)
                    {
                        sb.Append(repl);
                        i = semi; // skip to semicolon
                        continue;
                    }
                }
            }
            sb.Append(c);
        }
        return sb.ToString();
    }

    private void ImportCsv()
    {
        if (_entriesProp == null) return;
        var path = EditorUtility.OpenFilePanel("Import Translations CSV",
Application.dataPath, "csv");
        if (string.IsNullOrEmpty(path)) return;
        string[] lines;
        try
        {
            lines = File.ReadAllLines(path, Encoding.UTF8);
        }
        catch (Exception ex)
        {
            Debug.LogError($"[TranslationDatabaseAssetEditor] Import CSV read
error: {ex}");
            EditorUtility.DisplayDialog("Import CSV", "Failed to read file. See
Console for details.", "OK");
            return;
        }
        if (lines.Length == 0)
        {
            EditorUtility.DisplayDialog("Import CSV", "File is empty.", "OK");
            return;
        }
        int start = 0;
        // Detect header
        var header = ParseCsvLine(lines[0]);
        if (header.Count >= 3 &&
            string.Equals(header[0], "Key", StringComparison.OrdinalIgnoreCase)
&&
            string.Equals(header[1], "RU", StringComparison.OrdinalIgnoreCase)
&&
            string.Equals(header[2], "KZ", StringComparison.OrdinalIgnoreCase))
        {
            start = 1;
        }

        // Build existing key -> index map
        var map = new System.Collections.Generic.Dictionary<string,
int>(StringComparer.Ordinal);
        for (int i = 0; i < _entriesProp.arraySize; i++)
        {
            var el = _entriesProp.GetArrayElementAtIndex(i);
            var key = el.FindPropertyRelative("key").stringValue ?? string.Empty;
            if (!map.ContainsKey(key)) map[key] = i; // ignore duplicates; first wins
        }

        int updated = 0, added = 0, skipped = 0;
        Undo.RecordObject(serializedObject.targetObject, "Import Translation
CSV");
        for (int i = start; i < lines.Length; i++)
        {
            if (string.IsNullOrWhiteSpace(lines[i])) continue;
            var cols = ParseCsvLine(lines[i]);
            if (cols.Count == 0) { skipped++; continue; }
            var key = cols[0]?.Trim();
            if (string.IsNullOrEmpty(key)) { skipped++; continue; }
            var ru = cols.Count > 1 ? cols[1] : string.Empty;
            var kz = cols.Count > 2 ? cols[2] : string.Empty;

            if (map.TryGetValue(key, out var idx))
            {
                var el = _entriesProp.GetArrayElementAtIndex(idx);
                el.FindPropertyRelative("ruTranslation").stringValue = ru;
                el.FindPropertyRelative("kzTranslation").stringValue = kz;
                updated++;
            }
            else
            {
                int index = _entriesProp.arraySize;
                _entriesProp.InsertArrayElementAtIndex(index);
                var el = _entriesProp.GetArrayElementAtIndex(index);
                el.FindPropertyRelative("key").stringValue = key;
                el.FindPropertyRelative("ruTranslation").stringValue = ru;
                el.FindPropertyRelative("kzTranslation").stringValue = kz;
                map[key] = index;
                added++;
            }
        }
        serializedObject.ApplyModifiedProperties();
        EditorUtility.SetDirty(serializedObject.targetObject);
        Repaint();
        EditorUtility.DisplayDialog("Import CSV", $"Import finished. Updated:
{updated}, Added: {added}, Skipped: {skipped}.", "OK");
    }

    private static string EscapeCsv(string field)
    {
        if (field == null) return string.Empty;
        bool mustQuote = field.Contains(",") || field.Contains("\n") ||
field.Contains("\r") || field.Contains("\"");
        if (!mustQuote) return field;
        var escaped = field.Replace("\"", "\"\"");
        return "\"" + escaped + "\"";
    }

    private static System.Collections.Generic.List<string> ParseCsvLine(string
line)
    {
        var result = new System.Collections.Generic.List<string>();
        if (line == null) return result;
        var sb = new StringBuilder();
        bool inQuotes = false;
        for (int i = 0; i < line.Length; i++)
        {
            char c = line[i];
            if (inQuotes)
            {
                if (c == '")
                {
                    // Check for escaped quote
                    if (i + 1 < line.Length && line[i + 1] == '")
                    {
                        sb.Append('");
                        i++; // skip second quote
                    }
                    else
```

```
                {
                    inQuotes = false;
                }
            }
            else
            {
                sb.Append(c);
            }
        }
        else
        {
            if (c == ',')
            {
                result.Add(sb.ToString());
                sb.Length = 0;
            }
            else if (c == '"')
            {
                inQuotes = true;
            }
            else
            {
                sb.Append(c);
            }
        }
    }
    result.Add(sb.ToString());
    return result;
  }
}
```

```
using UnityEngine;
using UnityEngine.UI;
using TMPro;
using System.Collections.Generic;
using System.Linq;

public class DictionaryUI : MonoBehaviour
{
    public Transform contentParent;
    public GameObject wordItemPrefab;
    public Button nextPageButton;
    public Button prevPageButton;

    private List<WordData> collectedWords;
    private int currentPage = 0;
    private int itemsPerPage = 10;



    public void Init(List<WordData> allWords)
    {
        collectedWords = allWords.Where(w => w.IsCollected).ToList();
        currentPage = 0;
        ShowPage();
    }

    public void ShowPage()
    {
        foreach (Transform child in contentParent)
            Destroy(child.gameObject);

        var pageWords = collectedWords
            .Skip(currentPage * itemsPerPage)
            .Take(itemsPerPage);

        foreach (var word in pageWords)
        {
            var go = Instantiate(wordItemPrefab, contentParent);
            var texts = go.GetComponentsInChildren<TextMeshProUGUI>();
            texts[0].text = word.RussianWord;
            texts[1].text = word.KazakhWord;
        }

        prevPageButton.interactable = currentPage > 0;
        nextPageButton.interactable = (currentPage + 1) * itemsPerPage <
collectedWords.Count;
    }

    public void NextPage()
    {
        currentPage++;
        ShowPage();
    }

    public void PrevPage()
    {
        currentPage--;
        ShowPage();
    }
}
```

```
using UnityEngine;
using UnityEngine.Audio;

public class Letter3D : MonoBehaviour
{
    //[SerializeField] AudioSource audioSource;
    //[SerializeField] GameObject txtLetter;
    private char myLetter;
    private TileGeneration tileGen;
    //private MeshRenderer  mesh;

    private void Start()
    {
        //mesh=  GetComponent<MeshRenderer>();
    }

    public void Setup(char letter, TileGeneration manager)
    {
        myLetter = letter;
        tileGen = manager;
    }


    private void Update()
    {
        //
        if (GlobalVar.isWordFinished)
        {
            //Destroy(gameObject);
            gameObject.SetActive(false);
        }
        if (GlobalVar.autoDestroy) gameObject.SetActive(false);
    }

    private void OnTriggerEnter(Collider other)
    {
        if (other.CompareTag("Player"))
        {
            tileGen.OnLetterCollected(myLetter); //
            //audioSource.Play();
            //mesh.enabled = false;

            //Destroy(gameObject);
            gameObject.SetActive(false);
        }
    }



}
```

```
using System.Collections.Generic;

[System.Serializable]
public class ThemeData
{
    public int Id;
    public string RussianWord;
    public string KazakhWord;
}

[System.Serializable]
public class ThemeListWrapper
{
    public List<ThemeData> theame;
}
```

```
using System.Collections.Generic;
using UnityEngine;

namespace Scenes.Object.Words.Script
{
    [CreateAssetMenu(fileName = "ThemeDatabase", menuName =
"Words/ThemeDatabase")]
```

```
    public class ThemeDatabase : ScriptableObject
    {
        public List<ThemeData> Themes;
    }
}


using DG.Tweening;
using Scenes.UI.Scripts;
using TMPro;
using UnityEngine;
using UnityEngine.UI;

public class UILetter : MonoBehaviour
{
    public TextMeshProUGUI textMesh;
    public Image background; // ???? ?????????????? ????
    [SerializeField] private Transform visuals;

    public void Init(char letter)
    {
        if (letter == ' ')
        {
            textMesh.text = "";
            textMesh.alpha = 0f;
            if (background != null) background.enabled = false;
        }
        else
        {
            textMesh.text = letter.ToString();
            textMesh.alpha = 0.6f;
            if (background != null) background.enabled = true;
        }

        InstantHide();
    }

    public void UpdateLetterAlpha(float alpha)
    {
        textMesh.alpha = alpha;
    }

    public void Reveal(char letter)
    {
        textMesh.text = letter.ToString();
        textMesh.alpha = 1f;

        PlayJumpAnimation();
    }

    private void PlayJumpAnimation()
    {
        var sequence = DOTween.Sequence();
        sequence.Append(visuals.DOLocalJump(visuals.localPosition + new
Vector3(0, 0, 5), 20, 1, 0.3f));
        sequence.Play();
    }

    private void InstantHide()
    {
        //set alpha to 0
        textMesh.alpha = 0f;
        var col = new Color(background.color.r, background.color.g,
background.color.b, 0f);
        background.color = col;
    }

    public void PlayHideAnimation()
    {
        var sequence = DOTween.Sequence();
        textMesh.alpha = 1f;
        sequence.Append(visuals.DOLocalJump(visuals.localPosition + new
Vector3(0, 0, 5), 20, 1, 0.3f));
        sequence.Append(background.DOFade(0, 0.3f));
        sequence.Join(textMesh.DOFade(0, 0.3f));

        sequence.Play();
    }

    public void PlayShowAnimation(float targetAlpha =  0.6f)
    {
        var sequence = DOTween.Sequence();
        sequence.Append(background.DOFade(1, 0.3f));
        sequence.Join(textMesh.DOFade(targetAlpha, 0.3f));

        sequence.Play();
    }
}
```

```
using System;
using System.Collections.Generic;
using System.Linq;
using Scenes.Object.Words.Script;
using UnityEngine;

// Adapter to combine static and dynamic data into the existing WordData
structure
public static class WordDataAdapter
{
    public static WordData Combine(WordStaticEntry stat, WordDynamicData
dyn)
    {
        if (stat == null)
        {
            Debug.LogError("WordDataAdapter.Combine called with null static
entry");
            return null;
        }

        var wd = new WordData
        {
            RussianWord = stat.RussianWord,
            KazakhWord = stat.KazakhWord,
            AudioPath = stat.AudioPath,
            Sentence = stat.Sentence,
            CategoryId = stat.CategoryId,
            IsCollected = dyn?.IsCollected ?? false,
            HasWordStar = dyn?.HasWordStar ?? false,
            WordStarCount = dyn?.WordStarCount ?? 0,
            HasAudioStar = dyn?.HasAudioStar ?? false,
            AudioStarCount = dyn?.AudioStarCount ?? 0,
            HasSentenceStar = dyn?.HasSentenceStar ?? false,
            SentenceStarCount = dyn?.SentenceStarCount ?? 0
        };
        return wd;
    }

    public static List<WordData> CombineAll(IEnumerable<WordStaticEntry>
statList, IEnumerable<WordDynamicData> dynList)
    {
        if (statList == null) return new List<WordData>();

        var dynDict = (dynList ?? Enumerable.Empty<WordDynamicData>())
            .Where(d => d != null && !string.IsNullOrWhiteSpace(d.Key))
            .GroupBy(d => d.Key, StringComparer.OrdinalIgnoreCase)
            .ToDictionary(g => g.Key, g => g.First(),
StringComparer.OrdinalIgnoreCase);

        var result = new List<WordData>();
        foreach (var s in statList)
        {
            if (s == null) continue;
            dynDict.TryGetValue(GetKeyForStatic(s), out var d);
            result.Add(Combine(s, d));
        }
        return result;
    }

    public static string GetKeyForStatic(WordStaticEntry s) => s?.KazakhWord
?? string.Empty;
    public static string GetKeyForWord(WordData w) => w?.KazakhWord ??
string.Empty;

    public static WordDynamicData ExtractDynamic(WordData w)
    {
        if (w == null) return null;
        return new WordDynamicData
        {
            Key = GetKeyForWord(w),
            IsCollected = w.IsCollected,
            HasWordStar = w.HasWordStar,
            WordStarCount = w.WordStarCount,
            HasAudioStar = w.HasAudioStar,
            AudioStarCount = w.AudioStarCount,
            HasSentenceStar = w.HasSentenceStar,
            SentenceStarCount = w.SentenceStarCount
        };
    }

    public static WordStaticEntry ExtractStatic(WordData w)
    {
        if (w == null) return null;
        return new WordStaticEntry
        {
            RussianWord = w.RussianWord,
            KazakhWord = w.KazakhWord,
```

```
            AudioPath = w.AudioPath,
            Sentence = w.Sentence,
            CategoryId = w.CategoryId
        };
    }
}



using System;
using System.Collections.Generic;
using System.IO;
using UnityEngine;

// Dynamic (runtime / player-progress) data for words
[Serializable]
public class WordDynamicData
{
    // Key used to match to static entry. By default use KazakhWord as a stable
key.
    public string Key;

    public bool IsCollected;

    public bool HasWordStar;
    public int WordStarCount;

    public bool HasAudioStar;
    public int AudioStarCount;

    public bool HasSentenceStar;
    public int SentenceStarCount;
}

public static class WordDynamicDataStore
{
    private const string FileName = "words_dynamic.json";

    [Serializable]
    private class Wrapper
    {
        public List<WordDynamicData> Items = new List<WordDynamicData>();
    }

    public static string GetFilePath()
    {
        return Path.Combine(Application.persistentDataPath, FileName);
    }

    public static void Save(List<WordDynamicData> items)
    {
        try
        {
            var wrapper = new Wrapper { Items = items ?? new
List<WordDynamicData>() };
            var json = JsonUtility.ToJson(wrapper, true);
            File.WriteAllText(GetFilePath(), json);
#if UNITY_EDITOR
            Debug.Log($"WordDynamicData saved to: {GetFilePath()}\nCount:
{wrapper.Items.Count}");
#endif
        }
        catch (Exception e)
        {
            Debug.LogError($"Failed to save WordDynamicData: {e}");
        }
    }

    public static List<WordDynamicData> Load()
    {
        try
        {
            var path = GetFilePath();
            if (!File.Exists(path)) return new List<WordDynamicData>();
            var json = File.ReadAllText(path);
            var wrapper = JsonUtility.FromJson<Wrapper>(json);
            return wrapper?.Items ?? new List<WordDynamicData>();
        }
        catch (Exception e)
        {
            Debug.LogError($"Failed to load WordDynamicData: {e}");
            return new List<WordDynamicData>();
        }
    }

    public static void Upsert(WordDynamicData item)
    {
        if (item == null || string.IsNullOrWhiteSpace(item.Key))
        {
            Debug.LogWarning("Attempted to upsert null or keyless
WordDynamicData");
            return;
        }

        var list = Load();
        var found = false;
        for (int i = 0; i < list.Count; i++)
        {
            if (string.Equals(list[i].Key, item.Key,
StringComparison.OrdinalIgnoreCase))
            {
                list[i] = item;
                found = true;
                break;
            }
        }
        if (!found) list.Add(item);
        Save(list);
    }
}


using System;
using System.Collections.Generic;
using Quiz.Scripts;
using UnityEngine;

namespace Scenes.Object.Words.Script
{
    // Static (authoring-time) data for words
    [CreateAssetMenu(menuName = "Words/Word Static Database", fileName =
"WordStaticDatabase")]
    public class WordStaticDatabase : ScriptableObject
    {
        public List<WordStaticEntry> Entries = new List<WordStaticEntry>();
    }

    [Serializable]
    public class WordStaticEntry
    {
        public string RussianWord;
        public string KazakhWord;
        public string AudioPath;
        public SentenceData Sentence;
        public int CategoryId; // 1 = еда, 2 = животные, 3 = предметы (example)
    }
}


using System;
using System.Collections.Generic;
using Quiz.Scripts;
using UnityEngine;

//public class WordWorld
//{
//[System.Serializable]
//public class WordEntry
//{
//    public string RussianWord;  // ????? ?? ????????
//    public string KazakhWord;   // ????? ?? ???????????
//}

//[System.Serializable]
//public class WordGroup
//{
//    public string GroupName;          // ????????? "??????",
"???????"
//    public List<WordEntry> Words;
//}

//public class WordDatabase : MonoBehaviour
//{
//    public List<WordGroup> Groups;
//}



[Serializable]
public class WordData : IComparable<WordData>
{
    //Потому что C# не был сделан чтобы сортировать казахские слова
    public const string KazakhAlphabet =
"аәбвгғдеёжзийкқлмнңоөпрстуұүфхһцчшщъыіьэюя";

    public string RussianWord;
    public string KazakhWord;
```

```
    public string AudioPath;
    public SentenceData Sentence;

    public int CategoryId; // 1 = ******, 2 = ******, 3 = ********

    public bool IsCollected;

    public bool HasWordStar;
    public int WordStarCount;

    public bool HasAudioStar;
    public int AudioStarCount;

    public bool HasSentenceStar;
    public int SentenceStarCount;

    public int CompareTo(WordData other)
    {
        if (ReferenceEquals(this, other)) return 0;
        if (other is null) return 1;

        var k1 = KazakhWord.ToLower();
        var k2 = other.KazakhWord.ToLower();

        int minLength = Mathf.Min(k1.Length, k2.Length);

        for (int i = 0; i < minLength; i++)
        {
            int index1 = KazakhAlphabet.IndexOf(k1[i]);
            int index2 = KazakhAlphabet.IndexOf(k2[i]);

            if (index1 != index2)
                return index1.CompareTo(index2);
        }

        return k1.Length.CompareTo(k2.Length);
    }
}

[System.Serializable]
public class WordListWrapper
{
    public List<WordData> words;
}

//}



using UnityEngine;

public class ScrollWord : MonoBehaviour
{
    [SerializeField] private RectTransform leftCap;
    [SerializeField] private RectTransform rightCap;
    [SerializeField] private RectTransform center;
    [SerializeField] private float targetWidth = 800f;
    [SerializeField] private float duration = 1.5f;

    private float startWidth;

    void OnEnable()
    {
        startWidth = center.sizeDelta.x;
        //StartCoroutine(Unroll());
    }

    private System.Collections.IEnumerator Unroll()
    {
        float t = 0f;
        while (t < duration)
        {
            t += Time.deltaTime;
            float k = Mathf.Clamp01(t / duration);
            float newWidth = Mathf.Lerp(startWidth, targetWidth, k);
            //
            center.sizeDelta = new Vector2(newWidth, center.sizeDelta.y);
            //
            rightCap.anchoredPosition = new Vector2(leftCap.anchoredPosition.x +
newWidth, rightCap.anchoredPosition.y);
            yield return null;
        }
    }
}



using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using Scenes.Object.Words.Script;
using UnityEditor;
using UnityEngine;

[CustomEditor(typeof(ThemeDatabase))]
public class ThemeDatabaseEditor : UnityEditor.Editor
{
    public override void OnInspectorGUI()
    {
        DrawDefaultInspector();
        GUILayout.Space(10);
        if (GUILayout.Button("Load From JSON..."))
        {
            LoadFromJson((ThemeDatabase)target);
        }
    }

    private void LoadFromJson(ThemeDatabase db)
    {
        var path = EditorUtility.OpenFilePanel("Select Themes JSON",
Application.dataPath, "json");
        if (string.IsNullOrEmpty(path)) return;

        try
        {
            var json = File.ReadAllText(path);
            var themes = ParseThemes(json);

            if (themes == null)
            {
                EditorUtility.DisplayDialog("Load From JSON", "Could not parse the
JSON into ThemeData list.", "OK");
                return;
            }

            Undo.RecordObject(db, "Load Themes From JSON");
            db.Themes = themes;
            EditorUtility.SetDirty(db);
            AssetDatabase.SaveAssets();
            Repaint();
            Debug.Log($"Loaded {db.Themes?.Count ?? 0} themes into {db.name}
from {path}");
        }
        catch (Exception e)
        {
            Debug.LogError($"Failed to load JSON: {e}");
            EditorUtility.DisplayDialog("Load From JSON", $"Error reading or
parsing file:\n{e.Message}", "OK");
        }
    }

    private List<ThemeData> ParseThemes(string json)
    {
        if (string.IsNullOrWhiteSpace(json)) return null;
        var trimmed = json.Trim();

        // Case 1: top-level array
        if (trimmed.StartsWith("["))
        {
            var arr = JsonHelper.FromJson<ThemeData>(trimmed);
            return arr?.ToList() ?? new List<ThemeData>();
        }

        // Case 2: deserialize as ScriptableObject-like with Themes field
        var tempDb = ScriptableObject.CreateInstance<ThemeDatabase>();
        try
        {
            JsonUtility.FromJsonOverwrite(trimmed, tempDb);
            if (tempDb.Themes != null && tempDb.Themes.Count > 0)
                return new List<ThemeData>(tempDb.Themes);
        }
        catch { /* ignore */ }
        finally
        {
            ScriptableObject.DestroyImmediate(tempDb);
        }

        // Case 3: generic wrappers
        try
        {
            var wrapA = JsonUtility.FromJson<ThemesWrapper>(trimmed);
            if (wrapA?.Themes != null && wrapA.Themes.Count > 0)
                return wrapA.Themes;
        }
        catch { }

        try
```

```
        {
            var wrapB = JsonUtility.FromJson<themesWrapper>(trimmed);
            if (wrapB?.themes != null && wrapB.themes.Count > 0)
                return wrapB.themes;
        }
        catch { }

        // Support legacy typo: "theame"
        try
        {
            var wrapC = JsonUtility.FromJson<TheameWrapper>(trimmed);
            if (wrapC?.theame != null && wrapC.theame.Count > 0)
                return wrapC.theame;
        }
        catch { }

        return null;
    }

    [Serializable]
    private class ThemesWrapper
    {
        public List<ThemeData> Themes;
    }

    [Serializable]
    private class themesWrapper
    {
        public List<ThemeData> themes;
    }

    [Serializable]
    private class TheameWrapper
    {
        public List<ThemeData> theame;
    }
}



using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using Scenes.Object.Words.Script;
using UnityEditor;
using UnityEngine;

[CustomEditor(typeof(WordStaticDatabase))]
public class WordStaticDatabaseEditor : UnityEditor.Editor
{
    public override void OnInspectorGUI()
    {
        DrawDefaultInspector();
        GUILayout.Space(10);
        if (GUILayout.Button("Load From JSON..."))
        {
            LoadFromJson((WordStaticDatabase)target);
        }
    }

    private void LoadFromJson(WordStaticDatabase db)
    {
        var path = EditorUtility.OpenFilePanel("Select Word Static JSON",
Application.dataPath, "json");
        if (string.IsNullOrEmpty(path)) return;

        try
        {
            var json = File.ReadAllText(path);
            var entries = ParseEntries(json);

            if (entries == null)
            {
                EditorUtility.DisplayDialog("Load From JSON", "Could not parse the
JSON into WordStaticEntry list.", "OK");
                return;
            }

            Undo.RecordObject(db, "Load Word Static From JSON");
            db.Entries = entries;
            EditorUtility.SetDirty(db);
            AssetDatabase.SaveAssets();
            Repaint();
            Debug.Log($"Loaded {db.Entries?.Count ?? 0} entries into {db.name}
from {path}");
        }
        catch (Exception e)
        {
            Debug.LogError($"Failed to load JSON: {e}");
            EditorUtility.DisplayDialog("Load From JSON", $"Error reading or
parsing file:\n{e.Message}", "OK");
        }
    }

    private List<WordStaticEntry> ParseEntries(string json)
    {
        if (string.IsNullOrWhiteSpace(json)) return null;
        var trimmed = json.Trim();

        // Case 1: top-level array
        if (trimmed.StartsWith("["))
        {
            var arr = JsonHelper.FromJson<WordStaticEntry>(trimmed);
            return arr?.ToList() ?? new List<WordStaticEntry>();
        }

        // Case 2: deserialize as ScriptableObject-like with Entries field
        var tempDb = ScriptableObject.CreateInstance<WordStaticDatabase>();
        try
        {
            JsonUtility.FromJsonOverwrite(trimmed, tempDb);
            if (tempDb.Entries != null && tempDb.Entries.Count > 0)
                return new List<WordStaticEntry>(tempDb.Entries);
        }
        catch { /* ignore */ }
        finally
        {
            ScriptableObject.DestroyImmediate(tempDb);
        }

        // Case 3: generic wrappers
        try
        {
            var wrapA = JsonUtility.FromJson<EntriesWrapper>(trimmed);
            if (wrapA?.Entries != null && wrapA.Entries.Count > 0)
                return wrapA.Entries;
        } catch {}

        try
        {
            var wrapB = JsonUtility.FromJson<WordsWrapper>(trimmed);
            if (wrapB?.words != null && wrapB.words.Count > 0)
                return wrapB.words;
        } catch {}

        return null;
    }

    [Serializable]
    private class EntriesWrapper
    {
        public List<WordStaticEntry> Entries;
    }

    [Serializable]
    private class WordsWrapper
    {
        public List<WordStaticEntry> words;
    }
}

internal static class JsonHelper
{
    [Serializable]
    private class Wrapper<T>
    {
        public T[] Items;
    }

    public static T[] FromJson<T>(string json)
    {
        if (string.IsNullOrWhiteSpace(json)) return Array.Empty<T>();
        var wrapped = "{\"Items\":" + json + "}";
        var w = JsonUtility.FromJson<Wrapper<T>>(wrapped);
        return w?.Items ?? Array.Empty<T>();
    }
}



using System;
using DG.Tweening;
using Scenes.UI;
using Scenes.UI.Scripts;
using UnityEngine;
using UnityEngine.UI;
```

```
namespace Scenes.Quiz.Prefabs.Scripts
{
    public class AsyqCollect : MonoBehaviour
    {
        [SerializeField] private UITweenerBase smallAsyqTween;
        [SerializeField] private UITweenerBase bigAsyqTween;
        [SerializeField] private AsyqLabel asyqLabel;
        [SerializeField] private Image bigAsyqImage;
        [SerializeField] private Image smallAsyqImage;

        private void OnEnable()
        {
            smallAsyqTween.ResetToStart();
            bigAsyqTween.ResetToStart();
            smallAsyqTween.onAnimationEnd.AddListener(OnSmallAsyqCollect);
            bigAsyqTween.onAnimationEnd.AddListener(OnBigAsyqCollect);
            bigAsyqImage.DOFade(1f, 0f);
            smallAsyqImage.DOFade(1f, 0f);
        }

        private void OnDisable()
        {
smallAsyqTween.onAnimationEnd.RemoveListener(OnSmallAsyqCollect);
            bigAsyqTween.onAnimationEnd.RemoveListener(OnBigAsyqCollect);
        }

        private void OnBigAsyqCollect()
        {
            bigAsyqImage.DOFade(0f, 0.1f);
            asyqLabel.UpdateLabel();
        }

        private void OnSmallAsyqCollect()
        {
            smallAsyqImage.DOFade(0f, 0.1f);
            asyqLabel.UpdateLabel();
        }

        public void PlaySmallAsyqCollect()
        {
            if(!isActiveAndEnabled) return;
            smallAsyqTween.DoForwardAnimation();
        }

        public void PlayBigAsyqCollect()
        {
            if(!isActiveAndEnabled) return;
            bigAsyqTween.DoForwardAnimation();
        }
    }
}



using System;
using TMPro;
using UnityEngine;

namespace Scenes.Quiz.Prefabs.Scripts
{
    public class AsyqLabel : MonoBehaviour
    {
        [SerializeField] private TextMeshProUGUI asyqCountText;
        [SerializeField] private bool alwaysUpdate;
        private void OnEnable()
        {
            UpdateLabel();
        }

        private void Update()
        {
            if(alwaysUpdate) UpdateLabel();
        }

        public void UpdateLabel()
        {
            asyqCountText.text = $"{GlobalVar.Asyq}";
        }
    }
}



using System.Collections;
using System.Collections.Generic;
using DG.Tweening;
using Quiz.Scripts;
using UnityEngine;
using UnityEngine.UI;

namespace Scenes.Quiz.Prefabs.Scripts
{
    public class AudioSelectController : MultipleChoiceControllerBase
    {
        [SerializeField]
        private Button playButton;
        [SerializeField]
        private AudioSource audioSource;
        [SerializeField] private Slider progressSlider;

        private AudioClip _audioClip;

        public override void Init(List<WordData> wordPool)
        {
            base.Init(wordPool);
            if (playButton != null)
            {
                playButton.onClick.RemoveAllListeners();
                playButton.onClick.AddListener(PlayClip);
                playButton.interactable = true;
            }
        }

        protected override void OnQuestionPicked(WordData picked)
        {
            StartCoroutine(FileLoader.GetWordAudioClipStreaming(picked, clip =>
_audioClip = clip));
        }


        private void PlayClip()
        {
            if (_audioClip == null)
            {
                Debug.Log("Clip not loaded");
                return;
            }
            playButton.interactable = false;
            var clipLength = _audioClip.length;
            if (progressSlider != null)
            {
                progressSlider.value = 0f;
                progressSlider.maxValue = 1f;
                progressSlider.DOValue(1f, clipLength).SetEase(Ease.Linear);
            }
            audioSource.PlayOneShot(_audioClip);
            StartCoroutine(ResetPlayButtonAfterDelay(clipLength + 0.1f));
        }

        private IEnumerator ResetPlayButtonAfterDelay(float delay)
        {
            yield return new WaitForSeconds(delay);
            if (playButton != null)
            {
                progressSlider.value = 0f;
                playButton.interactable = true;
            }
        }

        protected override bool HasStar(WordData word) => word.HasAudioStar;
        protected override void SetHasStar(WordData word, bool value) =>
word.HasAudioStar = value;
        protected override int GetStarCount(WordData word) =>
word.AudioStarCount;
        protected override void SetStarCount(WordData word, int value) =>
word.AudioStarCount = value;
    }
}


using System;
using DG.Tweening;
using Quiz.Scripts;
using Scenes.UI;
using Scenes.UI.Scripts;
using UnityEngine;
using UnityEngine.Events;
using UnityEngine.UI;

namespace Scenes.Quiz.Prefabs.Scripts
{
    public class CompleteScreenController : MonoBehaviour
    {
        [SerializeField]
        private Image backgroundImage;

        [SerializeField]
```

```
        private float fadeTime;

        [SerializeField]
        private Button continueButton;

        [SerializeField]
        private UITweenerBase appearTweener;
        [SerializeField]
        private UITweenerBase superTweener;

        [SerializeField]
        private DifficultySelectController difficultySelect;

        public UnityEvent OnContinueClick;

        private void OnEnable()
        {
            continueButton.onClick.AddListener(OnContinue);
            backgroundImage.color = new Color(0, 0, 0, 0);
        }

        private void OnDisable()
        {
            continueButton.onClick.RemoveListener(OnContinue);
            backgroundImage.color = new Color(0, 0, 0, 0);
        }

        public void PlayAnimation(bool hasBonus = false)
        {
            backgroundImage.DOFade(0.8f, fadeTime);
            if(!hasBonus) appearTweener.DoForwardAnimation();
            else superTweener.DoForwardAnimation();
        }

        private void OnContinue()
        {
            difficultySelect.gameObject.SetActive(true);
            OnContinueClick?.Invoke();
            gameObject.SetActive(false);
        }
    }
}


using System;
using System.Collections.Generic;
using System.Linq;
using Quiz.Scripts.UI;
using Scenes.Object.UI.Translater;
using Scenes.Quiz.Prefabs.Scripts;
using TMPro;
using UnityEngine;
using UnityEngine.UI;
using Random = UnityEngine.Random;

namespace Quiz.Scripts
{
    public class DifficultySelectController : MonoBehaviour
    {
        [SerializeField] private WordSelectController wordSelectController;
        [SerializeField] private AudioSelectController audioSelectController;
        [SerializeField] private SentenceBuildController  sentenceBuildController;

        [SerializeField] private Button wordButton;
        [SerializeField] private Button audioButton;
        [SerializeField] private Button sentenceButton;

        [SerializeField] private Slider wordSlider;
        [SerializeField] private Slider audioSlider;
        [SerializeField] private Slider sentenceSlider;

        [Space]
        [SerializeField] private TextMeshProUGUI wordUnlockText;
        [SerializeField] private TextMeshProUGUI audioUnlockText;
        [SerializeField] private TextMeshProUGUI sentenceUnlockText;
        [SerializeField] private QuizNavbarFade navbarFade;
        [SerializeField] private WordGameConfig wordGameConfig;

        private int _totalWordCount;
        private void OnEnable()
        {
            Init();
            navbarFade.StartFadeOut();
        }

        private List<WordData> GetWordPool()
        {
            var words = FileLoader.LoadWordsFromJSON();
            // Single pass filter of collected words
            words = words.Where(w => w.IsCollected).ToList();
            _totalWordCount = words.Count;

            var wordPool = new List<WordData>();

            // If there are not enough words to unlock, keep behavior consistent with
Init()
            // but avoid early returns as per requirement. We'll just return the filtered
list at the end.
            // Build buckets in a single pass through the words list.
            var threeStarWords = new List<WordData>();
            var hasStarWords = new List<WordData>();
            var noStarWords = new List<WordData>();

            foreach (var word in words)
            {
                if (word.HasWordStar)
                {
                    threeStarWords.Add(word);
                }
                else if (word.WordStarCount > 0)
                {
                    hasStarWords.Add(word);
                }
                else
                {
                    noStarWords.Add(word);
                }
            }

            // If not unlocked yet, just use the filtered list and skip weighted selection
(no early return)
            if (words.Count < wordGameConfig.wordGameUnlockCount)
            {
                wordPool = words;
            }
            else
            {
                // Weighted selection without goto, adjusting weights dynamically
based on available buckets
                int targetSize = Mathf.Min(wordGameConfig.wordPoolSize,
words.Count);
                for (int i = 0; i < targetSize; i++)
                {
                    float w3 = threeStarWords.Count > 0 ?
wordGameConfig.threeStarPriority : 0f;
                    float w2 = hasStarWords.Count > 0 ?
wordGameConfig.hasStarPriority : 0f;
                    float w1 = noStarWords.Count > 0 ?
wordGameConfig.noStarPriority : 0f;
                    float total = w3 + w2 + w1;

                    if (total <= 0f)
                    {
                        // No words left in any bucket
                        break;
                    }

                    float r = Random.Range(0f, total);
                    if (r < w3)
                    {
                        var word = threeStarWords[Random.Range(0,
threeStarWords.Count)];
                        wordPool.Add(word);
                        threeStarWords.Remove(word);
                        continue;
                    }

                    r -= w3;
                    if (r < w2)
                    {
                        var word = hasStarWords[Random.Range(0,
hasStarWords.Count)];
                        wordPool.Add(word);
                        hasStarWords.Remove(word);
                        continue;
                    }

                    // Must be from noStarWords (w1 > 0 by construction here)
                    var w = noStarWords[Random.Range(0, noStarWords.Count)];
                    wordPool.Add(w);
                    noStarWords.Remove(w);
                }
            }

            return wordPool;
        }
```

```
    private List<WordData> GetSentencePool()
    {
        var words = FileLoader.LoadWordsWithSentencesFromJSON();
        words = words.Where(w =>
!string.IsNullOrEmpty(w.Sentence.CorrectWord)).ToList();

        var wordPool = new List<WordData>();
        var threeStarWords = new List<WordData>();
        var hasStarWords = new List<WordData>();
        var noStarWords = new List<WordData>();

        foreach (var word in words)
        {
            if (word.HasSentenceStar)
            {
                threeStarWords.Add(word);
            }
            else if (word.SentenceStarCount > 0)
            {
                hasStarWords.Add(word);
            }
            else
            {
                noStarWords.Add(word);
            }
        }

        var totalWeight = wordGameConfig.threeStarPriority +
wordGameConfig.hasStarPriority + wordGameConfig.noStarPriority;

        for (int i = 0; i < wordGameConfig.wordPoolSize; i++)
        {
            var random = Random.Range(0, totalWeight);
            if (random < wordGameConfig.threeStarPriority &&
threeStarWords.Count > 0)
            {
                var word = threeStarWords[Random.Range(0,
threeStarWords.Count)];
                wordPool.Add(word);
                threeStarWords.Remove(word);
                continue;
            }

            if (random < wordGameConfig.threeStarPriority +
wordGameConfig.hasStarPriority && hasStarWords.Count > 0)
            {
                var word = hasStarWords[Random.Range(0,
hasStarWords.Count)];
                wordPool.Add(word);
                hasStarWords.Remove(word);
                continue;
            }

            if (random < totalWeight)
            {
                if(noStarWords.Count == 0) continue;
                var word = noStarWords[Random.Range(0, noStarWords.Count)];
                wordPool.Add(word);
                noStarWords.Remove(word);
            }
        }

        return wordPool;
    }

    public void Init()
    {

        var pool = GetWordPool();

        //Don't even bother if we don't have enough words to play
        if(_totalWordCount < wordGameConfig.wordGameUnlockCount) return;

        var sentencePool = GetSentencePool();

        wordButton.interactable = _totalWordCount >=
wordGameConfig.wordGameUnlockCount;
        audioButton.interactable = _totalWordCount >=
wordGameConfig.audioGameUnlockCount;
        sentenceButton.interactable = _totalWordCount >=
wordGameConfig.sentenceGameUnlockCount;

        wordSlider.maxValue = wordGameConfig.wordGameUnlockCount;
        audioSlider.maxValue = wordGameConfig.audioGameUnlockCount;
        sentenceSlider.maxValue =
wordGameConfig.sentenceGameUnlockCount;

        wordSlider.value = Mathf.Min(_totalWordCount,
wordGameConfig.wordGameUnlockCount);
        audioSlider.value = Mathf.Min(_totalWordCount,
wordGameConfig.audioGameUnlockCount);
        sentenceSlider.value = Mathf.Min(_totalWordCount,
wordGameConfig.sentenceGameUnlockCount);

        wordUnlockText.text =
$"{Mathf.Min(wordGameConfig.wordGameUnlockCount,_totalWordCount)}/{wor
dGameConfig.wordGameUnlockCount}";
        audioUnlockText.text =
$"{Mathf.Min(wordGameConfig.audioGameUnlockCount,_totalWordCount)}/{wo
rdGameConfig.audioGameUnlockCount}";
        sentenceUnlockText.text =
$"{Mathf.Min(wordGameConfig.sentenceGameUnlockCount,_totalWordCount)}/
{wordGameConfig.sentenceGameUnlockCount}";


        //asyqCountText.text = $"{GlobalVar.Asyq}";

        wordButton.onClick.RemoveAllListeners();
        audioButton.onClick.RemoveAllListeners();
        sentenceButton.onClick.RemoveAllListeners();

        wordButton.onClick.AddListener(() => OnWordSelected(pool));
        audioButton.onClick.AddListener(() => OnAudioSelected(pool));
        sentenceButton.onClick.AddListener(() =>
OnSentenceSelected(sentencePool));
    }

    private void OnWordSelected(List<WordData> pool)
    {
        navbarFade.StartFadeIn();
        wordSelectController.gameObject.SetActive(true);
        gameObject.SetActive(false);
        wordSelectController.Init(pool);
    }

    private void OnAudioSelected(List<WordData> pool)
    {
        navbarFade.StartFadeIn();
        audioSelectController.gameObject.SetActive(true);
        gameObject.SetActive(false);
        audioSelectController.Init(pool);
    }

    private void OnSentenceSelected(List<WordData> pool)
    {
        navbarFade.StartFadeIn();
        sentenceBuildController.gameObject.SetActive(true);
        gameObject.SetActive(false);
        sentenceBuildController.Init(pool);
    }
  }
}


using System;
using System.Collections;
using System.Collections.Generic;
using System.IO;
using UnityEngine;
using UnityEngine.Networking;

namespace Quiz.Scripts
{
#if !DONT_USE_STREAMING
  public static class FileLoader
  {
    private const string FORMAT = ".mp3";

    public static List<WordData> LoadWordsFromJSON()
    {


        string path = Path.Combine(Application.streamingAssetsPath,
"words.json");
        if (!File.Exists(path))
        {
            Debug.LogError("Файл слов не найден: " + path);
            return null;
        }
        string json = File.ReadAllText(path);
        WordListWrapper wrapper =
JsonUtility.FromJson<WordListWrapper>(json);
        return wrapper.words;
    }

    public static void SaveWordsToJSON(List<WordData> words)
    {
```

```
        string path = Path.Combine(Application.streamingAssetsPath,
"words.json");
        if (!File.Exists(path))
        {
            Debug.LogError("Файл слов не найден: " + path);
            return;
        }
        string json = File.ReadAllText(path);
        var wrapper = JsonUtility.FromJson<WordListWrapper>(json);
        foreach (var word in words)
        {
            //check if wrapper has the word with the same RussianWord and if so,
replace it
            var wordIndex = wrapper.words.FindIndex(w => w.RussianWord ==
word.RussianWord);
            if (wordIndex != -1)
            {
                wrapper.words[wordIndex] = word;
            }
            else
            {
                wrapper.words.Add(word);
            }
        }
        json = JsonUtility.ToJson(wrapper, true);
        File.WriteAllText(path, json);
    }

    public static void SaveWordToJSON(WordData word)
    {
        string path = Path.Combine(Application.streamingAssetsPath,
"words.json");
        if (!File.Exists(path))
        {
            Debug.LogError("Файл слов не найден: " + path);
            return;
        }
        string json = File.ReadAllText(path);
        var wrapper = JsonUtility.FromJson<WordListWrapper>(json);

        //check if wrapper has the word with the same RussianWord and if so,
replace it
        var wordIndex = wrapper.words.FindIndex(w => w.RussianWord ==
word.RussianWord);
        if (wordIndex != -1)
        {
            wrapper.words[wordIndex] = word;
        }
        else
        {
            wrapper.words.Add(word);
        }

        json = JsonUtility.ToJson(wrapper, true);
        File.WriteAllText(path, json);
    }


    //Загружаем темы
    public static List<ThemeData> LoadThemesFromJson()
    {
        string path = Path.Combine(Application.streamingAssetsPath,
"theame.json");
        if (!File.Exists(path))
        {
            Debug.LogError("Файл тем не найден: " + path);
            return null;
        }

        string json = File.ReadAllText(path);
        ThemeListWrapper wrapper =
JsonUtility.FromJson<ThemeListWrapper>(json);

        return wrapper.theame;
    }

    //Костыль чтобы загружать аудио из стриминг асетов
    public static IEnumerator GetWordAudioClipStreaming(WordData
wordData, Action<AudioClip> callback, Action<string> errorCallback = null)
    {
        if (string.IsNullOrEmpty(wordData.AudioPath))
        {
            var msg = $"Audio path for {wordData.KazakhWord} is not defined";
            Debug.LogWarning(msg);
            errorCallback?.Invoke(msg);
            yield break;
        }

        var clipName = wordData.AudioPath;
        var modulId = wordData.CategoryId;
#if UNITY_ANDROID && !UNITY_EDITOR
        var clipPath =
$"jar:file:///{Application.dataPath}!/assets/Sound/Word/Modul_{modulId}/{clipNa
me}{FORMAT}";
#else
        var clipPath =
$"{Application.streamingAssetsPath}/Sound/Word/Modul_{modulId}/{clipName}{
FORMAT}";
#endif

        using (var www = UnityWebRequestMultimedia.GetAudioClip(clipPath,
AudioType.MPEG))
        {
            yield return www.SendWebRequest();
            if (www.result == UnityWebRequest.Result.Success)
            {
                var clip = DownloadHandlerAudioClip.GetContent(www);
                callback?.Invoke(clip);
                yield break;
            }
            var err = $"Error loading audio clip: path='{clipPath}',
error='{www.error}'";
            Debug.LogError(err);
            errorCallback?.Invoke(err);
        }
    }

    public static List<WordData> LoadWordsWithSentencesFromJSON()
    {
        string path = Path.Combine(Application.streamingAssetsPath,
"words_with_sentences.json");
        if (!File.Exists(path))
        {
            Debug.LogError("Файл слов не найден: " + path);
            return null;
        }
        string json = File.ReadAllText(path);
        WordListWrapper wrapper =
JsonUtility.FromJson<WordListWrapper>(json);
        return wrapper.words;
    }

    // Editor-friendly validation that checks whether audio files referenced by
words exist on disk.
    // This does not perform actual streaming request but validates paths that
GetWordAudioClipStreaming would use in Editor/Standalone.
    public static List<string> ValidateWordAudioFilesExist(List<WordData>
words, out int okCount, out int skippedNoPath)
    {
        var errors = new List<string>();
        okCount = 0;
        skippedNoPath = 0;
        if (words == null) return errors;

        foreach (var word in words)
        {
            if (string.IsNullOrEmpty(word.AudioPath))
            {
                skippedNoPath++;
                continue;
            }
            var clipName = word.AudioPath;
            var modulId = word.CategoryId;
            var clipPath = Path.Combine(Application.streamingAssetsPath,
$"Sound/Word/Modul_{modulId}/{clipName}{FORMAT}");
            if (File.Exists(clipPath))
            {
                okCount++;
            }
            else
            {
                var msg = $"Missing file: '{clipPath}' | Word='{word.KazakhWord}'
({word.RussianWord}) | CategoryId={word.CategoryId}";
                errors.Add(msg);
            }
        }
        return errors;
    }
  }
#endif
}


using System;
using System.Collections;
using System.Collections.Generic;
using System.IO;
using System.Linq;
```

```
using Scenes.Object.Words.Script;
using UnityEngine;
using UnityEngine.Networking;

namespace Quiz.Scripts
{
#if DONT_USE_STREAMING

    public static class FileLoader
    {
        private const string FORMAT = ".mp3";
        private const string WORD_STATIC_PATH = "WordStaticDatabase";
        private const string THEME_STARIC_PATH = "ThemeDatabase";

        public static List<WordData> LoadWordsFromJSON()
        {
            var staticDatabase =
Resources.Load<WordStaticDatabase>(WORD_STATIC_PATH).Entries;

            var dynamicData = WordDynamicDataStore.Load();

            var words = WordDataAdapter.CombineAll(staticDatabase,
dynamicData);

            return words;
        }
        public static void SaveWordsToJSON(List<WordData> words)
        {
            var dynamicData =
words.Select(WordDataAdapter.ExtractDynamic).ToList();

            WordDynamicDataStore.Save(dynamicData);
        }

        public static void SaveWordToJSON(WordData word)
        {
            var dynamicData = WordDataAdapter.ExtractDynamic(word);

            WordDynamicDataStore.Upsert(dynamicData);
        }


        //Загружаем темы
        public static List<ThemeData> LoadThemesFromJson()
        {
            var themes =
Resources.Load<ThemeDatabase>(THEME_STARIC_PATH).Themes;

            return themes;
        }

        //Костыль чтобы загружать аудио из стриминг асетов
        public static IEnumerator GetWordAudioClipStreaming(WordData
wordData, Action<AudioClip> callback, Action<string> errorCallback = null)
        {
            if (string.IsNullOrEmpty(wordData.AudioPath))
            {
                var msg = $"Audio path for {wordData.KazakhWord} is not defined";
                Debug.LogWarning(msg);
                errorCallback?.Invoke(msg);
                yield break;
            }

            var clipPath = Path.Combine($"Sound/Word", wordData.AudioPath);

            var clip = Resources.Load<AudioClip>(clipPath);

            callback?.Invoke(clip);
        }

        public static List<WordData> LoadWordsWithSentencesFromJSON()
        {
            return LoadWordsFromJSON();
        }

        // Editor-friendly validation that checks whether audio files referenced by
words exist on disk.
        // This does not perform actual streaming request but validates paths that
GetWordAudioClipStreaming would use in Editor/Standalone.
        public static List<string> ValidateWordAudioFilesExist(List<WordData>
words, out int okCount, out int skippedNoPath)
        {
            var errors = new List<string>();
            okCount = 0;
            skippedNoPath = 0;
            if (words == null) return errors;

            foreach (var word in words)
            {
                if (string.IsNullOrEmpty(word.AudioPath))
                {
                    skippedNoPath++;
                    continue;
                }
                var clipName = word.AudioPath;
                var moduleId = word.CategoryId;
                var clipPath = Path.Combine(Application.streamingAssetsPath,
$"Sound/Word/Modul_{moduleId}/{clipName}{FORMAT}");
                if (File.Exists(clipPath))
                {
                    okCount++;
                }
                else
                {
                    var msg = $"Missing file: '{clipPath}' | Word='{word.KazakhWord}'
({word.RussianWord}) | CategoryId={word.CategoryId}";
                    errors.Add(msg);
                }
            }
            return errors;
        }
    }

#endif
}


using System;
using System.Collections;
using System.Collections.Generic;
using System.IO;
using UnityEngine;
using UnityEngine.Networking;

namespace Quiz.Scripts
{
    public static class FileLoaderWebRequest
    {
        private const string FORMAT = ".mp3";

        public static IEnumerator LoadWordsFromJSON(Action<List<WordData>>
callback)
        {
            string path = Path.Combine(Application.streamingAssetsPath,
"words.json");
            string url = string.Empty;
            #if UNITY_ANDROID && !UNITY_EDITOR
                url = "jar:file://" + Application.dataPath + "!/assets/words.json";
            #else
                url = "file://" + path;
            #endif

            UnityWebRequest www = UnityWebRequest.Get(url);
            yield return www.SendWebRequest();

            if (www.result != UnityWebRequest.Result.Success)
            {
                Debug.LogError("Файл слов не найден: " + path);
                callback?.Invoke(null);
            }
            else
            {
                string json = www.downloadHandler.text;
                WordListWrapper wrapper =
JsonUtility.FromJson<WordListWrapper>(json);
                callback?.Invoke(wrapper.words);
            }
        }

        public static IEnumerator SaveWordsToJSON(List<WordData> words,
Action onComplete)
        {
            string path = Path.Combine(Application.streamingAssetsPath,
"words.json");
            string url = string.Empty;
            #if UNITY_ANDROID && !UNITY_EDITOR
                url = "jar:file://" + Application.dataPath + "!/assets/words.json";
            #else
                url = "file://" + path;
            #endif

            UnityWebRequest www = UnityWebRequest.Get(url);
            yield return www.SendWebRequest();

            if (www.result != UnityWebRequest.Result.Success)
            {
                Debug.LogError("Файл слов не найден: " + path);
            }
```

```
        else
        {
            string json = www.downloadHandler.text;
            var wrapper = JsonUtility.FromJson<WordListWrapper>(json);
            foreach (var word in words)
            {
                var wordIndex = wrapper.words.FindIndex(w => w.RussianWord ==
word.RussianWord);
                if (wordIndex != -1)
                {
                    wrapper.words[wordIndex] = word;
                }
                else
                {
                    wrapper.words.Add(word);
                }
            }
            json = JsonUtility.ToJson(wrapper, true);

            byte[] bodyRaw = System.Text.Encoding.UTF8.GetBytes(json);
            UnityWebRequest wwwUpload = new UnityWebRequest(url,
"POST");
            wwwUpload.uploadHandler = new UploadHandlerRaw(bodyRaw);
            wwwUpload.downloadHandler = new DownloadHandlerBuffer();
            yield return wwwUpload.SendWebRequest();

            if(wwwUpload.result != UnityWebRequest.Result.Success)
            {
                Debug.LogError("Failed to save words: " + wwwUpload.error);
            }
            onComplete?.Invoke();
        }
    }

    public static IEnumerator SaveWordToJSON(WordData word, Action
onComplete)
    {
        string path = Path.Combine(Application.streamingAssetsPath,
"words.json");
        string url = string.Empty;
        #if UNITY_ANDROID && !UNITY_EDITOR
            url = "jar:file://" + Application.dataPath + "!/assets/words.json";
        #else
            url = "file://" + path;
        #endif

        UnityWebRequest www = UnityWebRequest.Get(url);
        yield return www.SendWebRequest();

        if (www.result != UnityWebRequest.Result.Success)
        {
            Debug.LogError("Файл слов не найден: " + path);
        }
        else
        {
            string json = www.downloadHandler.text;
            var wrapper = JsonUtility.FromJson<WordListWrapper>(json);

            var wordIndex = wrapper.words.FindIndex(w => w.RussianWord ==
word.RussianWord);
            if (wordIndex != -1)
            {
                wrapper.words[wordIndex] = word;
            }
            else
            {
                wrapper.words.Add(word);
            }

            json = JsonUtility.ToJson(wrapper, true);

            byte[] bodyRaw = System.Text.Encoding.UTF8.GetBytes(json);
            UnityWebRequest wwwUpload = new UnityWebRequest(url,
"POST");
            wwwUpload.uploadHandler = new UploadHandlerRaw(bodyRaw);
            wwwUpload.downloadHandler = new DownloadHandlerBuffer();
            yield return wwwUpload.SendWebRequest();

            if(wwwUpload.result != UnityWebRequest.Result.Success)
            {
                Debug.LogError("Failed to save word: " + wwwUpload.error);
            }
            onComplete?.Invoke();
        }
    }

    public static IEnumerator
LoadThemesFromJson(Action<List<ThemeData>> callback)
    {
        string path = Path.Combine(Application.streamingAssetsPath,
"theame.json");
        string url = string.Empty;
        #if UNITY_ANDROID && !UNITY_EDITOR
            url = "jar:file://" + Application.dataPath + "!/assets/theame.json";
        #else
            url = "file://" + path;
        #endif

        UnityWebRequest www = UnityWebRequest.Get(url);
        yield return www.SendWebRequest();

        if (www.result != UnityWebRequest.Result.Success)
        {
            Debug.LogError("Файл тем не найден: " + path);
            callback?.Invoke(null);
        }
        else
        {
            string json = www.downloadHandler.text;
            ThemeListWrapper wrapper =
JsonUtility.FromJson<ThemeListWrapper>(json);
            callback?.Invoke(wrapper.theame);
        }
    }

    public static AudioClip GetWordAudioClip(WordData wordData)
    {
        if (string.IsNullOrEmpty(wordData.AudioPath))
        {
            Debug.Log("Audio path is not defined");
            return null;
        }

        var clipPath = Path.Combine("Audio", wordData.AudioPath);

        Debug.Log(clipPath);

        var clip = Resources.Load<AudioClip>(clipPath);

        return clip;
    }

    public static IEnumerator GetWordAudioClipStreaming(WordData
wordData, Action<AudioClip> callback, Action<string> errorCallback = null)
    {
        if (string.IsNullOrEmpty(wordData.AudioPath))
        {
            var msg = $"Audio path for {wordData.KazakhWord} is not defined";
            Debug.LogWarning(msg);
            errorCallback?.Invoke(msg);
            yield break;
        }

        var clipName = wordData.AudioPath;
        var modulId = wordData.CategoryId;
#if UNITY_ANDROID && !UNITY_EDITOR
        var clipPath =
$"jar:file://{Application.dataPath}!/assets/Sound/Word/Modul_{modulId}/{clipNa
me}{FORMAT}";
#else
        var clipPath =
$"{Application.streamingAssetsPath}/Sound/Word/Modul_{modulId}/{clipName}{
FORMAT}";
#endif

        using (var www = UnityWebRequestMultimedia.GetAudioClip(clipPath,
AudioType.MPEG))
        {
            yield return www.SendWebRequest();
            if (www.result == UnityWebRequest.Result.Success)
            {
                var clip = DownloadHandlerAudioClip.GetContent(www);
                callback?.Invoke(clip);
                yield break;
            }
            var err = $"Error loading audio clip: path='{clipPath}',
error='{www.error}'";
            Debug.LogError(err);
            errorCallback?.Invoke(err);
        }
    }

    public static IEnumerator
LoadWordsWithSentencesFromJSON(Action<List<WordData>> callback)
    {
        string path = Path.Combine(Application.streamingAssetsPath,
"words_with_sentences.json");
        string url = string.Empty;
```

```
        #if UNITY_ANDROID && !UNITY_EDITOR
            url = "jar:file://" + Application.dataPath +
"!/assets/words_with_sentences.json";
        #else
            url = "file://" + path;
        #endif

        UnityWebRequest www = UnityWebRequest.Get(url);
        yield return www.SendWebRequest();

        if (www.result != UnityWebRequest.Result.Success)
        {
            Debug.LogError("Файл слов не найден: " + path);
            callback?.Invoke(null);
        }
        else
        {
            string json = www.downloadHandler.text;
            WordListWrapper wrapper =
JsonUtility.FromJson<WordListWrapper>(json);
            callback?.Invoke(wrapper.words);
        }
    }

    public static IEnumerator ValidateWordAudioFilesExist(List<WordData>
words, Action<List<string>, int, int> callback)
    {
        var errors = new List<string>();
        int okCount = 0;
        int skippedNoPath = 0;

        if (words == null)
        {
            callback?.Invoke(errors, okCount, skippedNoPath);
            yield break;
        }

        foreach (var word in words)
        {
            if (string.IsNullOrEmpty(word.AudioPath))
            {
                skippedNoPath++;
                continue;
            }

            var clipName = word.AudioPath;
            var modulId = word.CategoryId;
            string url = string.Empty;
            #if UNITY_ANDROID && !UNITY_EDITOR
                url =
$"jar:file://{Application.dataPath}!/assets/Sound/Word/Modul_{modulId}/{clipNa
me}{FORMAT}";
            #else
                url =
$"file://{Application.streamingAssetsPath}/Sound/Word/Modul_{modulId}/{clipNa
me}{FORMAT}";
            #endif

            UnityWebRequest www = UnityWebRequest.Head(url);
            yield return www.SendWebRequest();

            if (www.result == UnityWebRequest.Result.Success)
            {
                okCount++;
            }
            else
            {
                var msg = $"Missing file: '{url}' | Word='{word.KazakhWord}'
({word.RussianWord}) | CategoryId={word.CategoryId}";
                errors.Add(msg);
            }
        }

        callback?.Invoke(errors, okCount, skippedNoPath);
    }
  }
}


using System.Collections.Generic;

namespace Quiz.Scripts
{
  public static class ListUtil
  {
    public static void Shuffle<T>(this List<T> list)
    {
        var rng = new System.Random();
        int n = list.Count;
        while (n > 1)
        {
            n--;
            int k = rng.Next(n + 1);
            (list[k], list[n]) = (list[n], list[k]);
        }
    }
  }
}


using System.Collections;
using DG.Tweening;
using UnityEngine;
using UnityEngine.UI;

namespace Scenes.Quiz.Prefabs.Scripts
{
  public class QuizNavbarFade : MonoBehaviour
  {
    [SerializeField]
    private float fadeTime;
    [SerializeField]
    private Image navbarImage;

    private Coroutine _fadeOutCoroutine;
    private Coroutine _fadeInCoroutine;

    public void StartFadeIn()
    {
        if(_fadeOutCoroutine != null) StopCoroutine(_fadeOutCoroutine);
        _fadeInCoroutine = StartCoroutine(FadeInCoroutine());
    }

    private IEnumerator FadeOutCoroutine()
    {
        while (navbarImage.color.a > 0f)
        {
            var alpha = navbarImage.color.a - Time.deltaTime / fadeTime;
            navbarImage.color = new Color(navbarImage.color.r,
navbarImage.color.g, navbarImage.color.b, alpha);
            yield return null;
        }
        navbarImage.gameObject.SetActive(false);
    }

    private IEnumerator FadeInCoroutine()
    {
        navbarImage.gameObject.SetActive(true);
        while (navbarImage.color.a < 0.8f)
        {
            var alpha = navbarImage.color.a + Time.deltaTime / fadeTime;
            navbarImage.color = new Color(navbarImage.color.r,
navbarImage.color.g, navbarImage.color.b, alpha);
            yield return null;
        }
    }

    public void StartFadeOut()
    {
        if(_fadeInCoroutine != null) StopCoroutine(_fadeInCoroutine);
        _fadeOutCoroutine = StartCoroutine(FadeOutCoroutine());
    }
  }
}


using Quiz.Scripts;

namespace Scenes.Quiz.Prefabs.Scripts
{
  public static class RandomThemePicker
  {
    public static int GetRandomThemeId()
    {
        var themes = FileLoader.LoadThemesFromJson();

        var rand = UnityEngine.Random.Range(0, themes.Count);
        return themes[rand].Id;
    }
  }
}


using Quiz.Scripts;
using UnityEngine;

public class ResetProgress : MonoBehaviour
{
```

```
    public void ResetPogress()
    {
        var words = FileLoader.LoadWordsFromJSON();
    }
}



using System.Collections.Generic;
using Quiz.Scripts.UI;
using TMPro;
using UnityEngine;

namespace Scenes.Quiz.Prefabs.Scripts
{
    public class SentenceBuildController : MultipleChoiceControllerBase
    {
        [SerializeField]
        private TextMeshProUGUI questionText;

        public override void Init(List<WordData> wordPool)
        {
            base.Init(wordPool);
        }

        protected override void OnQuestionPicked(WordData picked)
        {
            // Set the question text to the sentence with the missing word
            if (questionText != null && picked != null && picked.Sentence != null)
            {
                questionText.text = picked.Sentence.SentenceWithoutWord;
            }
        }

        protected override void ApplyCorrectEffects(WordItemUI word)
        {
            questionText.text = currentWord.Sentence.SentenceWithCorrectWord;
            base.ApplyCorrectEffects(word);
        }


        protected override void SetUpQuestion()
        {
            // Use base flow for common pre-checks and hint setup, but replace
choice building
            if (currentCorrectCount >= wordGameConfig.sessionGoal)
            {
                StartBonus();
            }

            if (bonusStarted && bonusFailed)
            {
                OnSessionCompleted(false);
                return;
            }

            if (bonusStarted && currentCorrectCount >=
wordGameConfig.sessionGoal + wordGameConfig.bonusWordCount)
            {
                OnSessionCompleted(true);
                return;
            }

            if (!wordQueueManager.HasWords())
            {
                Debug.Log("All words are used");
                return;
            }
            hintCoroutine = HintCoroutine();
            StartCoroutine(hintCoroutine);
            fiftyFiftyUsedThisQuestion = false;
            correctIndex = -1;

            currentWord = wordQueueManager.GetNextWord();
            OnQuestionPicked(currentWord);

            if (currentWord == null || currentWord.Sentence == null)
            {
                base.SetUpQuestion();
                return;
            }

            var sentence = currentWord.Sentence;
            var correctWord = sentence.CorrectWord;

            var displayWords = new List<string>();
            displayWords.Add(correctWord);
            foreach (var w in sentence.WrongWords) displayWords.Add(w);
            displayWords.Sort((_, __) => UnityEngine.Random.Range(0, 2) == 0 ? -
1 : 1);

            for (int i = 0; i < wordItems.Count && i < 4; i++)
            {
                if (displayWords[i] == correctWord)
                {
                    correctIndex = i;
                    wordItems[i].Init(displayWords[i], OnCorrectClicked);
                }
                else
                {
                    wordItems[i].Init(displayWords[i], OnWrongClicked);
                }
            }

            if (fiftyFiftyButton != null) fiftyFiftyButton.interactable = true;
            if (skipButton != null) skipButton.interactable = true;
        }

        // Map star logic to Sentence-specific fields
        protected override bool HasStar(WordData word) =>
word.HasSentenceStar;
        protected override void SetHasStar(WordData word, bool value) =>
word.HasSentenceStar = value;
        protected override int GetStarCount(WordData word) =>
word.SentenceStarCount;
        protected override void SetStarCount(WordData word, int value) =>
word.SentenceStarCount = value;
    }
}


using System;
using System.Collections.Generic;

namespace Quiz.Scripts
{
    [Serializable]
    public class SentenceData
    {
        public string Sentence;
        public string CorrectWord;
        public List<string> WrongWords;

        public string SentenceWithoutWord => Sentence.Replace("<->", "...");
        public string SentenceWithCorrectWord => Sentence.Replace("<->",
CorrectWord);
    }

    [Serializable]
    public class SentenceDataWrapper
    {
        public List<SentenceData> Sentences;
    }
}


using System.Collections.Generic;
using Quiz.Scripts.UI;
using UnityEngine;
using UnityEngine.Serialization;

namespace Quiz.Scripts
{
    public class ThemeSelect : MonoBehaviour
    {
        [SerializeField] private GameObject topicUIPrefab;
        [SerializeField] private Transform topicUIContainer;
        [SerializeField] private DifficultySelectController difficultySelectController;

        private List<ThemeData> allThemes;
        // Start is called once before the first execution of Update after the
MonoBehaviour is created
        void Start()
        {
            var topics = FileLoader.LoadThemesFromJson();
            allThemes = topics;
            InitThemes();
        }

        private void InitThemes()
        {
            foreach (var theme in allThemes)
            {
                var inst = Instantiate(topicUIPrefab, topicUIContainer);
                var uiItem = inst.GetComponent<TopicItemUI>();
                uiItem.Init(theme, OnTopicSelected);
            }
```

```
        }

        private void OnTopicSelected(TopicItemUI itemUI)
        {

            difficultySelectController.gameObject.SetActive(true);
            gameObject.SetActive(false);
        }
    }
}



using UnityEngine;

namespace Quiz.Scripts
{
    [CreateAssetMenu(fileName = "WordGameConfig", menuName =
"WordGameConfig", order = 0)]
    public class WordGameConfig : ScriptableObject
    {
        public int wordPoolSize;
        public int wordQueueSize;
        public int minWordsBeforeRepeat;
        public int sessionGoal;
        public int bonusWordCount;
        [Space]
        public float threeStarPriority;
        public float hasStarPriority;
        public float noStarPriority;
        [Space]
        public int wordGameUnlockCount;
        public int audioGameUnlockCount;
        public int sentenceGameUnlockCount;
        [Space]
        public int fiftyFiftyCost;
        public int skipCost;
        public float hintDelay;

    }
}


using System.Collections.Generic;
using System.Linq;
using UnityEngine;

namespace Quiz.Scripts
{
    public class WordQueueManager
    {
        private readonly List<WordData> _wordPool;
        private readonly int _queueSize;
        private readonly int _minWordsBeforeRepeat;

        // Track last index to avoid returning the same word twice in a row
        private int _lastIndex = -1;

        public List<WordData> PoolCopy => _wordPool;

        public WordQueueManager(List<WordData> wordPool, int queueSize, int
minWordsBeforeRepeat)
        {
            _wordPool = new List<WordData>(wordPool);
        }

        public bool HasWords()
        {
            return _wordPool.Count > 0;
        }

        public WordData GetNextWord()
        {
            if (_wordPool == null || _wordPool.Count == 0)
            {
                return null;
            }

            var count = _wordPool.Count;


            int index;
            if (_lastIndex < 0)
            {
                index = Random.Range(0, count);
            }
            else
            {
                // особый кейс чтобы не повторять предидущее слово
                index = Random.Range(0, count - 1);
                if (index >= _lastIndex)
                {
                    index++;
                }
            }

            _lastIndex = index;
            return _wordPool[index];
        }

        public void RequeueWordOnWrong(WordData word)
        {
            return;
        }

        public void RequeueWordOnCorrect(WordData word)
        {
            return;
        }
    }
}


using System.Collections;
using FIMSpace;
using Quiz.Scripts;
using Quiz.Scripts.UI;
using TMPro;
using UnityEngine;

namespace Scenes.Quiz.Prefabs.Scripts
{
    public class WordSelectController : MultipleChoiceControllerBase
    {
        protected override bool HasStar(WordData word) => word.HasWordStar;
        protected override void SetHasStar(WordData word, bool value) =>
word.HasWordStar = value;
        protected override int GetStarCount(WordData word) =>
word.WordStarCount;
        protected override void SetStarCount(WordData word, int value) =>
word.WordStarCount = value;
        [Header("Системные")]
        [SerializeField] private TextMeshProUGUI headerText;

        protected override void OnQuestionPicked(WordData picked)
        {
            if (headerText != null)
            {
                headerText.text = picked.KazakhWord.CapitalizeFirstLetter();
            }
        }

        protected override void ApplySkip(WordItemUI correctItem)
        {
            // In word mode, skip should not add star but still proceed
            correctItem.DoCorrectAnimation();
            foreach (var word in wordItems)
            {
                word.DisableButton();
            }
            StartCoroutine(RestartAfterSkip());
        }

        private IEnumerator RestartAfterSkip()
        {
            yield return new WaitForSeconds(2f);
            FileLoader.SaveWordToJSON(currentWord);
            SetUpQuestion();
        }
    }
}


using System.Collections;
using System.Collections.Generic;
using Quiz.Scripts;
using Quiz.Scripts.UI;
using Scenes.Quiz.Prefabs.Scripts.UI;
using Scenes.UI;
using UnityEngine;
using UnityEngine.UI;

namespace Scenes.Quiz.Prefabs.Scripts
{
    /// <summary>
    /// Base controller encapsulating shared multiple-choice quiz logic
    /// between WordSelectController and AudioSelectController.
    /// </summary>
```

```
public abstract class MultipleChoiceControllerBase : MonoBehaviour
{
    [Header("Common UI")]
    [SerializeField] protected List<WordItemUI> wordItems;
    [SerializeField] protected ProgressBarUI progressBar;
    [SerializeField] protected AsyqCollect asyqCollect;
    [SerializeField] protected Button backButton;
    [SerializeField] protected DifficultySelectController difficultySelect;
    [SerializeField] protected CompleteScreenController
completeScreenController;

    [Header("Hints")]
    [SerializeField] protected List<AdvancedUITweener> hintTweens;
    [SerializeField] protected Button fiftyFiftyButton;
    [SerializeField] protected Button skipButton;

    [Header("Config")]
    [SerializeField] protected WordGameConfig wordGameConfig;

    protected WordQueueManager wordQueueManager;
    protected WordData currentWord;
    protected int correctIndex = -1;
    protected bool fiftyFiftyUsedThisQuestion;
    protected IEnumerator hintCoroutine;

    protected int currentCorrectCount;
    protected bool bonusStarted;
    protected bool bonusFailed;

    public virtual void Init(List<WordData> wordPool)
    {
        wordQueueManager = new WordQueueManager(wordPool,
wordGameConfig.wordQueueSize, wordGameConfig.minWordsBeforeRepeat);
        var progressMax = wordGameConfig.sessionGoal +
wordGameConfig.bonusWordCount;
        bonusStarted = false;
        progressBar.Init(0, progressMax);
        currentCorrectCount = 0;

        if (backButton != null)
        {
            backButton.onClick.RemoveAllListeners();
            backButton.onClick.AddListener(OnBackClicked);
        }

        if (fiftyFiftyButton != null)
        {
            fiftyFiftyButton.onClick.RemoveAllListeners();
            fiftyFiftyButton.onClick.AddListener(OnFiftyFiftyClicked);
        }
        if (skipButton != null)
        {
            skipButton.onClick.RemoveAllListeners();
            skipButton.onClick.AddListener(OnSkipClicked);
        }

        SetUpQuestion();
    }

    protected IEnumerator HintCoroutine()
    {
        while (true)
        {
            yield return new WaitForSeconds(wordGameConfig.hintDelay);
            foreach (var tween in hintTweens)
            {
                tween.DoAnimation();
            }
        }
    }

    protected virtual void OnBackClicked()
    {
        if (hintCoroutine != null) StopCoroutine(hintCoroutine);
        if (difficultySelect != null) difficultySelect.gameObject.SetActive(true);
        gameObject.SetActive(false);
    }

    protected virtual void OnSessionCompleted(bool hasBonus = false)
    {
        if (hintCoroutine != null) StopCoroutine(hintCoroutine);
        bonusStarted = false;
        bonusFailed = false;
        completeScreenController.gameObject.SetActive(true);
        completeScreenController.OnContinueClick.RemoveAllListeners();
        completeScreenController.OnContinueClick.AddListener(() => {
gameObject.SetActive(false);});

        completeScreenController.PlayAnimation(hasBonus);
        //gameObject.SetActive(false);
    }

    protected virtual void StartBonus()
    {
        bonusStarted = true;
        progressBar.DoBonusAnimation();
    }

    protected virtual void SetUpQuestion()
    {
        if (!bonusStarted && currentCorrectCount >=
wordGameConfig.sessionGoal)
        {
            StartBonus();
        }

        if (bonusStarted && bonusFailed)
        {
            OnSessionCompleted(false);
            return;
        }

        if (bonusStarted && currentCorrectCount >=
wordGameConfig.sessionGoal + wordGameConfig.bonusWordCount)
        {
            OnSessionCompleted(true);
            return;
        }

        if (!wordQueueManager.HasWords())
        {
            Debug.Log("All words are used");
            return;
        }

        hintCoroutine = HintCoroutine();
        StartCoroutine(hintCoroutine);
        fiftyFiftyUsedThisQuestion = false;
        correctIndex = -1;

        PrepareNewCurrentWord();
        var correctWord = currentWord;

        OnQuestionPicked(correctWord);

        var displayWords = BuildDefaultDisplayWords(correctWord);
        WireDefaultChoices(displayWords, correctWord);

        EnableHintButtons();

        AfterChoicesWired();
    }

    protected virtual void AfterChoicesWired() { }

    // Extraction: preparation of current word
    protected virtual void PrepareNewCurrentWord()
    {
        var poolCopy = new List<WordData>(wordQueueManager.PoolCopy);
        currentWord = wordQueueManager.GetNextWord();
    }

    // Extraction: build 3 random wrong words + correct
    protected virtual List<WordData> BuildDefaultDisplayWords(WordData
correctWord)
    {
        var poolCopy = new List<WordData>(wordQueueManager.PoolCopy);
        poolCopy.Remove(correctWord);
        var displayWords = new List<WordData>();
        for (int i = 0; i < 3 && poolCopy.Count > 0; i++)
        {
            var randomWord = poolCopy[Random.Range(0, poolCopy.Count)];
            displayWords.Add(randomWord);
            poolCopy.Remove(randomWord);
        }
        displayWords.Add(correctWord);
        // shuffle list (simple random comparator)
        displayWords.Sort((_, __) => Random.Range(0, 2) == 0 ? -1 : 1);
        return displayWords;
    }

    // Extraction: wire default UI choices for WordData list using RussianWord
labels
    protected virtual void WireDefaultChoices(List<WordData> displayWords,
WordData correctWord)
    {
```

```
        for (int i = 0; i < wordItems.Count && i < 4 && i < displayWords.Count;
i++)
        {
            if (displayWords[i] == correctWord)
            {
                correctIndex = i;
                wordItems[i].Init(displayWords[i].RussianWord, OnCorrectClicked);
            }
            else
            {
                wordItems[i].Init(displayWords[i].RussianWord, OnWrongClicked);
            }
        }
    }

    protected virtual void EnableHintButtons()
    {
        if (fiftyFiftyButton != null) fiftyFiftyButton.interactable = true;
        if (skipButton != null) skipButton.interactable = true;
    }

    protected virtual void OnQuestionPicked(WordData picked) { }

    protected void OnWrongClicked(WordItemUI word)
    {
        ApplyWrongEffects(word);
        var correctItem = wordItems[correctIndex];
        ApplyCorrectEffects(correctItem);
        if (hintCoroutine != null) StopCoroutine(hintCoroutine);
        if (bonusStarted)
        {
            progressBar.DoResetAnimation();
            bonusFailed = true;
        }
        StartCoroutine(RestartDelayed());
    }

    protected void OnCorrectClicked(WordItemUI word)
    {
        currentCorrectCount++;
        progressBar.UpdateProgress(currentCorrectCount);

        ApplyCorrectEffects(word);
        TryCollectAsyq();

        if (hintCoroutine != null) StopCoroutine(hintCoroutine);
        StartCoroutine(RestartDelayed());
    }

    private void TryCollectAsyq()
    {
        if (currentCorrectCount == wordGameConfig.sessionGoal)
        {
            GlobalVar.Asyq++;
            PlayerPrefs.SetInt("Asyq", GlobalVar.Asyq);
            asyqCollect.PlaySmallAsyqCollect();
        }

        if (currentCorrectCount == wordGameConfig.sessionGoal +
wordGameConfig.bonusWordCount)
        {
            GlobalVar.Asyq++;
            PlayerPrefs.SetInt("Asyq", GlobalVar.Asyq);
            asyqCollect.PlayBigAsyqCollect();
        }
    }

    protected virtual void ApplyWrongEffects(WordItemUI word)
    {
        // Default: requeue if not starred and play wrong animation
        if (!HasStar(currentWord))
        {
            ResetStarCount(currentWord);
            wordQueueManager.RequeueWordOnWrong(currentWord);
        }
        word.DoWrongAnimation();
    }

    protected virtual void ApplyCorrectEffects(WordItemUI word)
    {
        if (!HasStar(currentWord))
        {
            IncrementStar(currentWord);
            if (GetStarCount(currentWord) < 3)
            {
                wordQueueManager.RequeueWordOnCorrect(currentWord);
            }
            else
            {
                SetHasStar(currentWord, true);
            }
        }
        word.DoCorrectAnimation();
    }

    private IEnumerator RestartDelayed()
    {
        FileLoader.SaveWordToJSON(currentWord);
        foreach (var wi in wordItems) wi.DisableButton();
        yield return new WaitForSeconds(2f);
        SetUpQuestion();
    }

    protected virtual void OnFiftyFiftyClicked()
    {
        if (fiftyFiftyUsedThisQuestion || correctIndex < 0) return;

        if (hintCoroutine != null) StopCoroutine(hintCoroutine);

        var wrongIndices = new List<int>();
        for (int i = 0; i < wordItems.Count; i++)
        {
            if (i != correctIndex) wrongIndices.Add(i);
        }

        for (int j = 0; j < 2 && wrongIndices.Count > 0; j++)
        {
            int pick = Random.Range(0, wrongIndices.Count);
            int idx = wrongIndices[pick];
            wrongIndices.RemoveAt(pick);
            wordItems[idx].DisableButton();
        }

        fiftyFiftyUsedThisQuestion = true;
        if (fiftyFiftyButton != null) fiftyFiftyButton.interactable = false;
    }

    protected virtual void OnSkipClicked()
    {
        if (correctIndex < 0) return;
        if (skipButton != null) skipButton.interactable = false;
        if (hintCoroutine != null) StopCoroutine(hintCoroutine);
        var correctItem = wordItems[correctIndex];
        // Apply correct without granting star/progress; delegate to override
        ApplySkip(correctItem);
    }

    protected virtual void ApplySkip(WordItemUI correctItem)
    {
        // Default skip: treat as correct without adding star or progress
        // So we manually call animation and restart without star change
        correctItem.DoCorrectAnimation();
        StartCoroutine(RestartDelayed());
    }

    // Star handling (to be implemented by derived classes for different modes)
    protected abstract bool HasStar(WordData word);
    protected abstract void SetHasStar(WordData word, bool value);
    protected abstract int GetStarCount(WordData word);
    protected abstract void SetStarCount(WordData word, int value);
    protected virtual void ResetStarCount(WordData word) =>
SetStarCount(word, 0);
    protected virtual void IncrementStar(WordData word) =>
SetStarCount(word, GetStarCount(word) + 1);
  }
}



using System;
using System.Collections.Generic;
using System.Linq;
using Quiz.Scripts;
using TMPro;
using UnityEngine;
using UnityEngine.UI;

namespace Scenes.Quiz.Prefabs.Scripts.Dictionary
{
  public class DictionaryController : MonoBehaviour
  {
    [SerializeField]
    private DictionaryItem dictionaryItem;

    [SerializeField]
    private Transform content;

    [SerializeField]
```

```
    private DictionaryLetterLabel letterLabel;
    [SerializeField]
    private Transform letterContent;

    [SerializeField]
    private Button nextButton;
    [SerializeField]
    private Button previousButton;
    [SerializeField]
    private bool calculatePageSizeDynamically;
    [SerializeField]
    private int wordsPerPage;
    [SerializeField]
    private WordDisplay wordDisplay;

    [SerializeField]
    private TextMeshProUGUI pageText;

    [SerializeField] private bool autoSelectFirstLetter = false;


    private void Awake()
    {
        if(calculatePageSizeDynamically) CalculatePageSize();
        Init();
    }

    private void CalculatePageSize()
    {
        var height = content.GetComponent<RectTransform>().rect.height;
        var gridLayout = content.GetComponent<GridLayoutGroup>();
        var wordHeight = gridLayout.cellSize.y;
        var padding = gridLayout.padding.bottom + gridLayout.padding.top;
        var spacing = gridLayout.spacing.y;
        wordsPerPage = Mathf.FloorToInt((height - padding - spacing) /
(wordHeight + spacing)) * 2;
    }

    protected List<WordData> words;
    protected List<WordData> displayedWords;


    protected int currentPage;

    protected int totalPages;

    private DictionaryItem _selectedItem;
    private List<DictionaryItem> dictionaryItems;

    protected DictionaryLetterLabel _selectedLetterLabel;

    private void OnEnable()
    {
        ReloadWords();
    }

    protected virtual void ReloadWords()
    {
        if(!isInitialized) return;

        words = FileLoader.LoadWordsFromJSON();

        words.Sort();

        var letter = _selectedLetterLabel.Letter;
        displayedWords = words.Where((w) => w.KazakhWord.First() ==
letter[0]).ToList();

        LoadPage(currentPage);
    }


    private bool isInitialized = false;
    protected virtual void Init()
    {
        words = FileLoader.LoadWordsFromJSON();

        words.Sort();

        var letters = words.Select(x => x.KazakhWord.First()).Distinct().ToList();


        //words = words.Where(x => x.IsCollected).ToList();

        DictionaryLetterLabel firstLetterLabel = null;

        for (var i = 0; i < letters.Count; i++)
        {
            var letter = letters[i];
            var inst = Instantiate(letterLabel, letterContent);
            inst.Init(letter + "");

            if(i==0) firstLetterLabel = inst;

            var wordCount = words.Count(x => x.KazakhWord.First() == letter);
            if (wordCount <= 0) inst.Disable();

            inst.OnClick.RemoveAllListeners();
            inst.OnClick.AddListener(OnLetterClicked);
        }


        displayedWords = words;

        wordDisplay.DisableVisuals();

        dictionaryItems = new List<DictionaryItem>();

        CalculatePages();

        for (int i = 0; i < wordsPerPage; i++)
        {
            var item = Instantiate(dictionaryItem, content);
            dictionaryItems.Add(item);
        }

        nextButton.onClick.RemoveAllListeners();
        previousButton.onClick.RemoveAllListeners();

        nextButton.onClick.AddListener(NextPage);
        previousButton.onClick.AddListener(PreviousPage);

        LoadPage(currentPage);

        if(autoSelectFirstLetter) OnLetterClicked(firstLetterLabel);

        isInitialized = true;
    }

    protected void CalculatePages()
    {
        currentPage = 0;
        totalPages = Mathf.CeilToInt(displayedWords.Count / (float)
wordsPerPage);
    }

    protected virtual void OnLetterClicked(DictionaryLetterLabel label)
    {
        if (label == _selectedLetterLabel)
        {
            _selectedLetterLabel = null;
            label.Deselect();
            displayedWords = words.ToList();
        }
        else
        {
            if (_selectedLetterLabel) _selectedLetterLabel.Deselect();
            _selectedLetterLabel = label;
            label.Select();
            var letter = label.Letter;
            displayedWords = words.Where((w) => w.KazakhWord.First() ==
letter[0]).ToList();
        }


        CalculatePages();
        LoadPage(currentPage);
    }

    private void NextPage()
    {
        if(currentPage + 1 >= totalPages) return;

        currentPage++;
        LoadPage(currentPage);
    }

    private void PreviousPage()
    {
        if(currentPage - 1 < 0) return;

        currentPage--;
        LoadPage(currentPage);
    }

    protected void LoadPage(int page)
    {
        foreach (var item in dictionaryItems)
```

```
            {
                item.DisableVisuals();
            }

            for (int i = 0; i < wordsPerPage; i++)
            {
                var index = page * wordsPerPage + i;
                if (index >= displayedWords.Count) break;

                var word = displayedWords[index];
                dictionaryItems[i].Init(word);
                dictionaryItems[i].OnSelect.AddListener(OnSelect);
                dictionaryItems[i].SetDisabled(!word.IsCollected);
            }

            pageText.text = (page + 1) + "/" + totalPages;
        }

        protected virtual void OnSelect(DictionaryItem item)
        {
            if(_selectedItem) _selectedItem.Deselect();
            _selectedItem = item;
            item.Select();

            wordDisplay.Init(item.WordData);
        }
    }
}



using System.Collections.Generic;
using System.Linq;
using Quiz.Scripts;
using TMPro;
using UnityEngine;
using UnityEngine.UI;

namespace Scenes.Quiz.Prefabs.Scripts.Dictionary
{
    public class DictionaryControllerMobile : DictionaryController
    {
        [Space]
        [Header("Mobile")]

        [SerializeField] private Button letterSelectButton;
        [SerializeField] private Button letterCloseButton;
        [SerializeField] private TextMeshProUGUI letterSelectText;
        [SerializeField] private TextMeshProUGUI wordCountText;
        [SerializeField] private GameObject letterSelectPanel;
        [SerializeField] private AudioSource audioSource;



        protected override void Init()
        {
            base.Init();

            letterSelectButton.onClick.AddListener(OpenSelectLetterPanel);
            letterCloseButton.onClick.AddListener(CloseSelectLetterPanel);
        }

        protected override void ReloadWords()
        {
            base.ReloadWords();

            UpdateWordCountDisplay();
        }

        private void UpdateWordCountDisplay()
        {
            if(words == null || words.Count == 0) words =
FileLoader.LoadWordsFromJSON();

            var wordCount = words.Count;
            var collectedCount = words.Count(w => w.IsCollected);
            wordCountText.text = $"{collectedCount}/{wordCount}";
        }


        protected override void OnLetterClicked(DictionaryLetterLabel label)
        {
            if (label == _selectedLetterLabel) return;


            if (_selectedLetterLabel) _selectedLetterLabel.Deselect();
            _selectedLetterLabel = label;
            label.Select();
            var letter = label.Letter;
            displayedWords = words.Where((w) => w.KazakhWord.First() ==
letter[0]).ToList();

            UpdateLetterLabelText(label.Letter);
            CloseSelectLetterPanel();

            CalculatePages();
            LoadPage(currentPage);
        }

        protected override void OnSelect(DictionaryItem item)
        {
            LoadAndPlayAudio(item.WordData);;
            base.OnSelect(item);
        }

        private void LoadAndPlayAudio(WordData word)
        {
            StartCoroutine(FileLoader.GetWordAudioClipStreaming(word, (clip) =>
            {
                if(audioSource.isPlaying) return;
                audioSource.clip = clip;
                audioSource.Play();
            }));
        }

        private void UpdateLetterLabelText(string text)
        {
            letterSelectText.text = text;
        }

        private void OpenSelectLetterPanel()
        {
            letterSelectPanel.SetActive(true);
        }

        private void CloseSelectLetterPanel()
        {
            letterSelectPanel.SetActive(false);
        }


    }
}



using System.Collections.Generic;
using DG.Tweening;
using FIMSpace;
using Scenes.UI;
using TMPro;
using UnityEngine;
using UnityEngine.Events;
using UnityEngine.UI;

namespace Scenes.Quiz.Prefabs.Scripts.Dictionary
{
    public class DictionaryItem : MonoBehaviour
    {
        [SerializeField]
        private GameObject visuals;
        [SerializeField]
        private TextMeshProUGUI wordText;
        [SerializeField]
        private Button selectButton;
        [SerializeField]
        private Image backgroundImage;

        [SerializeField] private List<Image> starImages;

        [SerializeField] private Sprite starOnSprite;
        [SerializeField] private Sprite starOffSprite;

        // New serialized fields for disabled state
        [SerializeField] private Color disabledTextColor = new Color(0.6f, 0.6f, 0.6f,
1f);
        [SerializeField] private Color defaultTextColor = new Color(1f, 1f, 1f, 1f);
        [SerializeField] private Sprite starDisabledSprite;

        [SerializeField]
        private UITweener appearTweener;
        [SerializeField]
        private UITweener textTweener;

        public UnityEvent<DictionaryItem> OnSelect;

        private WordData _wordData;
        private bool _isDisabled;
```

```
    public WordData WordData => _wordData;
    public void Init(WordData wordData)
    {
        _wordData = wordData;

        selectButton.onClick.RemoveAllListeners();
        selectButton.onClick.AddListener(OnSelectButtonClicked);
        OnSelect.RemoveAllListeners();

        EnableVisuals();
        InitVisuals();
    }

    private void OnSelectButtonClicked()
    {
        if (_isDisabled) return;
        OnSelect?.Invoke(this);
    }

    private void InitVisuals()
    {
        if (wordText != null) wordText.text =
_wordData.KazakhWord.CapitalizeFirstLetter();
        // Reset background instantly; if disabled, ensure no tween plays
        if (backgroundImage != null)
        {
            if (_isDisabled) backgroundImage.DOKill();
            backgroundImage.DOFade(0f, 0f);
        }

        // Initialize stars depending on disabled state
        if (starImages != null && starImages.Count >= 3)
        {
            if (_isDisabled && starDisabledSprite != null)
            {
                starImages[0].sprite = starDisabledSprite;
                starImages[1].sprite = starDisabledSprite;
                starImages[2].sprite = starDisabledSprite;
            }
            else
            {
                starImages[0].sprite = _wordData.HasWordStar ? starOnSprite :
starOffSprite;
                starImages[1].sprite = _wordData.HasSentenceStar ? starOnSprite
: starOffSprite;
                starImages[2].sprite = _wordData.HasAudioStar ? starOnSprite :
starOffSprite;
            }
        }

        // Apply text color depending on state
        if (wordText != null)
        {
            wordText.color = _isDisabled ? disabledTextColor : defaultTextColor;
        }

        // Interactivity
        if (selectButton != null)
        {
            selectButton.interactable = !_isDisabled;
        }
    }

    public void DisableVisuals()
    {
        visuals.SetActive(false);
    }

    public void Select()
    {
        if (_isDisabled) return;
        backgroundImage.DOFade(1f, 0.1f);
    }

    public void Deselect()
    {
        if (_isDisabled) return;
        backgroundImage.DOFade(0f, 0.1f);
    }

    public void EnableVisuals()
    {
        visuals.SetActive(true);
    }

    // Public API to toggle disabled state
    public void SetDisabled(bool disabled)
    {
        _isDisabled = disabled;

        // Stop any running tweens/animations
        if (backgroundImage != null) backgroundImage.DOKill();
        // If appearTweener supports stopping, disable it via Unity lifecycle
        if (appearTweener != null) appearTweener.enabled = !disabled;
        if (textTweener != null) textTweener.enabled = !disabled;

        // Update visuals without animations
        InitVisuals();
    }
  }
}


using DG.Tweening;
using Scenes.UI;
using TMPro;
using UnityEngine;
using UnityEngine.Events;
using UnityEngine.EventSystems;
using UnityEngine.UI;

namespace Scenes.Quiz.Prefabs.Scripts.Dictionary
{
  public class DictionaryLetterLabel : MonoBehaviour
  {
    [SerializeField] private TextMeshProUGUI letterText;
    [SerializeField] private Button button;
    [SerializeField] private EventTrigger eventTrigger;

    public UnityEvent<DictionaryLetterLabel> OnClick;

    private string _letter;

    [SerializeField]
    private Color defaultColor;
    [SerializeField]
    private Color selectedColor;
    [SerializeField]
    private Color disabledColor;
    public string Letter => _letter;

    public void Init(string letter)
    {
        _letter = letter;
        letterText.text = letter.ToUpper();
        letterText.color = defaultColor;
        button.onClick.RemoveAllListeners();
        button.onClick.AddListener(OnSelectClicked);
    }

    private void OnSelectClicked()
    {
        OnClick?.Invoke(this);
    }


    public void Select()
    {
        letterText.DOColor(selectedColor, 0.2f);
    }

    public void Deselect()
    {
        letterText.DOColor(defaultColor, 0.2f);
    }

    public void Disable()
    {
        button.interactable = false;
        eventTrigger.enabled = false;
        letterText.color = disabledColor;
    }
  }
}


using System.Collections;
using DG.Tweening;
using FIMSpace;
using Quiz.Scripts;
using TMPro;
using UnityEngine;
using UnityEngine.UI;

namespace Scenes.Quiz.Prefabs.Scripts.Dictionary
{
  public class WordDisplay : MonoBehaviour
```

```
    {
        [SerializeField] private GameObject visuals;
        [SerializeField] private TextMeshProUGUI kazakhWord;
        [SerializeField] private TextMeshProUGUI russianWord;
        [SerializeField] private Color defaultKzColor;
        [SerializeField] private Color defaultRuColor;
        [SerializeField] private Color fillColor;

        private WordData _wordData;

        public void Init(WordData wordData)
        {
            _wordData = wordData;

            InitVisuals();
            EnableVisuals();
        }

        private void InitVisuals()
        {
            kazakhWord.text = _wordData.KazakhWord.CapitalizeFirstLetter();
            russianWord.text = _wordData.RussianWord.CapitalizeFirstLetter();
        }

        public void DisableVisuals()
        {
            visuals.SetActive(false);
        }

        public void EnableVisuals()
        {
            visuals.SetActive(true);
        }

        //Text recolor effect
        private void RecolorText(float t)
        {
            var kazakhMesh = kazakhWord.mesh;
            var russianMesh = russianWord.mesh;

            //collect all verts from text in order

            var kazakhColors = kazakhMesh.colors;
            var russianColors = russianMesh.colors;

            for (var i = 0; i < kazakhColors.Length; i++)
            {
                if ((float)i / (float)kazakhColors.Length < t)
                {
                    kazakhColors[i] = fillColor;
                }
                else
                {
                    kazakhColors[i] = defaultKzColor;
                }
            }

            for (var i = 0; i < russianColors.Length; i++)
            {
                if ((float)i / (float)russianColors.Length < t)
                {
                    russianColors[i] = fillColor;
                }
                else
                {
                    russianColors[i] = defaultRuColor;
                }
            }

            kazakhMesh.colors = kazakhColors;
            russianMesh.colors = russianColors;

            kazakhWord.UpdateGeometry(kazakhMesh, 0);
            russianWord.UpdateGeometry(russianMesh, 0);

            kazakhWord.ForceMeshUpdate();
            russianWord.ForceMeshUpdate();
        }

    }
}


using UnityEditor;
using UnityEngine;
using Quiz.Scripts;
using System.Collections.Generic;

public class WordEditMenu : EditorWindow
{
    private List<WordData> words;
    private Vector2 scrollPosition;

    [MenuItem("SteppeRider/Edit Word Progress")]
    public static void ShowWindow()
    {
        var window = GetWindow<WordEditMenu>("Edit Word Progress");
        var words = FileLoader.LoadWordsFromJSON();
        words.Sort();
        window.words = words;

    }

    private void OnGUI()
    {
        if (GUILayout.Button("Reload"))
        {
            words = FileLoader.LoadWordsFromJSON();
        }

        if (GUILayout.Button("Reset All Progress"))
        {
            foreach (var word in words)
            {
                word.IsCollected = false;
                word.HasWordStar = false;
                word.WordStarCount = 0;
                word.HasAudioStar = false;
                word.AudioStarCount = 0;
                word.HasSentenceStar = false;
                word.SentenceStarCount = 0;
            }
            FileLoader.SaveWordsToJSON(words);
        }

        if (GUILayout.Button("Reset Stars"))
        {
            foreach (var word in words)
            {
                word.WordStarCount = 0;
                word.HasWordStar = false;
                word.AudioStarCount = 0;
                word.HasAudioStar = false;
                word.SentenceStarCount = 0;
                word.HasSentenceStar = false;
            }
            FileLoader.SaveWordsToJSON(words);
        }

        if (GUILayout.Button("Collect all words"))
        {
            foreach (var word in words)
            {
                word.IsCollected = true;
            }
            FileLoader.SaveWordsToJSON(words);
        }

        int totalWords = 0;
        int totalCollectedWords = 0;
        int collectedWordStars = 0;
        int collectedAudioStars = 0;
        int collectedSentenceStars = 0;

        foreach (var word in words)
        {
            totalWords++;

            if (word.IsCollected) totalCollectedWords++;
            if (word.HasWordStar) collectedWordStars++;
            if (word.HasAudioStar) collectedAudioStars++;
            if (word.HasSentenceStar) collectedSentenceStars++;
        }

        EditorGUILayout.LabelField($"Collected Words:
{totalCollectedWords}/{totalWords}");
        EditorGUILayout.LabelField($"Word Stars:
{collectedWordStars}/{totalWords}");
        EditorGUILayout.LabelField($"Audio Stars:
{collectedAudioStars}/{totalWords}");
        EditorGUILayout.LabelField($"Sentence Stars:
{collectedSentenceStars}/{totalWords}");

        scrollPosition = EditorGUILayout.BeginScrollView(scrollPosition);

        foreach (var word in words)
        {
            EditorGUILayout.BeginHorizontal();
```

```
            EditorGUILayout.LabelField(word.KazakhWord, GUILayout.Width(100));
            EditorGUILayout.LabelField(word.RussianWord,
GUILayout.Width(100));

            word.IsCollected = EditorGUILayout.Toggle("Collected",
word.IsCollected);

            word.HasWordStar = EditorGUILayout.Toggle("Word Star",
word.HasWordStar);
            word.WordStarCount = EditorGUILayout.IntField("Word Stars",
word.WordStarCount);

            word.HasAudioStar = EditorGUILayout.Toggle("Audio Star",
word.HasAudioStar);
            word.AudioStarCount = EditorGUILayout.IntField("Audio Stars",
word.AudioStarCount);

            word.HasSentenceStar = EditorGUILayout.Toggle("Sentence Star",
word.HasSentenceStar);
            word.SentenceStarCount = EditorGUILayout.IntField("Sentence Stars",
word.SentenceStarCount);

            EditorGUILayout.EndHorizontal();
        }

        EditorGUILayout.EndScrollView();

        if (GUILayout.Button("Save Changes"))
        {
            FileLoader.SaveWordsToJSON(words);
        }
    }
}


#if UNITY_EDITOR

using System.Collections.Generic;
using NUnit.Framework;
using Quiz.Scripts;
using UnityEngine;

public class WordAudioPathValidationTests
{
    [Test]
    public void Validate_Audio_Files_Exist_And_Report()
    {
        var words = FileLoader.LoadWordsFromJSON();
        if (words == null || words.Count == 0)
        {
            Debug.Log("[AudioPathReport] words.json not found or empty for this
environment. Skipping validation.");
            Assert.Pass();
        }

        int ok;
        int skipped;
        List<string> errors = FileLoader.ValidateWordAudioFilesExist(words, out
ok, out skipped);

        Debug.Log($"[AudioPathReport] Total={words.Count} | OK={ok} |
SkippedNoPath={skipped} | MissingFiles={errors.Count}");
        foreach (var e in errors)
        {
            Debug.LogError("[AudioPathMissing] " + e);
        }

        // Do not fail the build because of data issues; ensure the check ran for all
        Assert.AreEqual(words.Count, ok + skipped + errors.Count, "Not all entries
were processed");
    }
}

#endif



using TMPro;
using UnityEngine;

namespace Scenes.Quiz.Prefabs.Scripts.UI
{
    public class NewWordItemUI : MonoBehaviour
    {
        private WordData word;

        [SerializeField] private TMP_Text ruWordText;
        [SerializeField] private TMP_Text kzWordText;

        public void Init(WordData word)
        {
            this.word = word;
            ruWordText.text = word.RussianWord;
            kzWordText.text = word.KazakhWord;
        }


    }
}


using System.Collections.Generic;
using Quiz.Scripts.UI;
using UnityEngine;
using UnityEngine.UI;

namespace Scenes.Quiz.Prefabs.Scripts.UI
{
    public class NewWordsUI : UIPanel
    {
        [SerializeField]
        private NewWordItemUI wordItems;
        [SerializeField]
        private RectTransform content;


        public void SetWords(List<WordData> words)
        {
            ClearWords();

            foreach (var word in words)
            {
                var newItem = Instantiate(wordItems.gameObject, content);
                newItem.SetActive(true);
                newItem.GetComponent<NewWordItemUI>().Init(word);
            }

            LayoutRebuilder.ForceRebuildLayoutImmediate(content);
        }

        private void ClearWords()
        {
            foreach (Transform child in content)
            {
                Destroy(child.gameObject);
            }
        }
    }
}


using System;
using TMPro;
using UnityEngine;
using UnityEngine.UI;

// Holds serialized references for the Pause Menu UI.
public class PauseUI : MonoBehaviour
{
    [SerializeField]
    private Button resumeButton;
    [SerializeField]
    private Button quitButton;

    [SerializeField]
    private PlayerController playerController;

    private void OnEnable()
    {
        // Pause the game time when the pause UI is shown
        Time.timeScale = 0f;
        // Wire up button events
        if (resumeButton != null)
resumeButton.onClick.AddListener(ResumeGame);
        if (quitButton != null) quitButton.onClick.AddListener(QuitGame);
    }

    private void OnDisable()
    {
        // Restore game time when the pause UI is hidden
        Time.timeScale = 1f;
        // Unwire button events
        if (resumeButton != null)
resumeButton.onClick.RemoveListener(ResumeGame);
        if (quitButton != null) quitButton.onClick.RemoveListener(QuitGame);
    }

    private void ResumeGame()
```

```
    {
        // Continue the gameplay via player controller and hide the pause UI
        if (playerController != null)
        {
            playerController.ContinueGame();
        }
        gameObject.SetActive(false);
    }

    private void QuitGame()
    {
        // Stop the current game/session via player controller
        if (playerController != null)
        {
            playerController.StopGame();
        }
        gameObject.SetActive(false);
    }
}
```

```
using System;
using DG.Tweening;
using Scenes.UI;
using TMPro;
using UnityEngine;
using UnityEngine.UI;

namespace Scenes.Quiz.Prefabs.Scripts.UI
{
    public class PopupUI : MonoBehaviour
    {
        [SerializeField]
        private Image popupBackground;
        [SerializeField]
        private TextMeshProUGUI popupText;
        [SerializeField]
        private UITweener popupTween;

        [SerializeField]
        private float animationDuration;

        [SerializeField] private bool initializeOnStart = true;

        private int totalCharacters;

        private void Start()
        {
            if(initializeOnStart) Init();
        }

        public void Init()
        {
            totalCharacters = popupText.text.Length;
            popupBackground.DOFade(0f, 0f);
            popupTween.SetStartTransform();
        }

        public void PlayForwardAnimation()
        {
            popupBackground.DOFade(1f, animationDuration);
            popupText.maxVisibleCharacters = 0;
            DOTween.To(() => popupText.maxVisibleCharacters, x =>
popupText.maxVisibleCharacters = x, totalCharacters, animationDuration);
            popupTween.DoForwardAnimation();
        }

        public void PlayBackwardAnimation()
        {
            popupBackground.DOFade(0f, animationDuration);
            popupText.maxVisibleCharacters = totalCharacters;
            DOTween.To(() => popupText.maxVisibleCharacters, x =>
popupText.maxVisibleCharacters = x, 0, animationDuration);
            popupTween.DoBackwardAnimation();
        }
    }
}
```

```
using System.Collections.Generic;
using DG.Tweening;
using UnityEngine;
using UnityEngine.UI;

namespace Scenes.Quiz.Prefabs.Scripts.UI
{
    public class ProgressBarUI : MonoBehaviour
    {
        [SerializeField]
        private Slider progressSlider;

        [SerializeField] private List<RawImage> starOverlay;
        [SerializeField] private Image progressFill;

        [SerializeField] private Color defaultColor;
        [SerializeField] private Color bonusColor;
        public void Init(float min, float max)
        {
            progressFill.color = defaultColor;
            foreach (var star in starOverlay)
            {
                star.color = new Color(1, 1, 1, 0);
            }

            progressSlider.minValue = min;
            progressSlider.maxValue = max;

            progressSlider.value = min;
        }

        public void DoBonusAnimation()
        {
            foreach (var star in starOverlay)
            {
                star.DOFade(1, 0.5f);
            }

            progressFill.DOColor(bonusColor, 0.5f);
        }

        public void DoResetAnimation()
        {
            foreach (var star in starOverlay)
            {
                star.DOFade(0, 0.5f);
            }
            progressFill.DOColor(defaultColor, 0.5f);
        }


        public void UpdateProgress(float newValue)
        {
            progressSlider.DOValue(newValue, 0.5f);
        }
    }
}
```

```
using System;
using System.Collections;
using System.Linq;
using TMPro;
using UnityEngine;
using UnityEngine.UI;

namespace Scenes.Quiz.Prefabs.Scripts.UI
{
    public class RegisterUI : UIPanel
    {
        [SerializeField]
        private Button registerButton;
        [SerializeField]
        private TMP_InputField usernameInput;

        [SerializeField]
        private int minUsernameLength;
        [SerializeField]
        private int maxUsernameLength;

        [SerializeField] private ApiSender apiSender;


        protected override void OnEnable()
        {
            usernameInput.onValueChanged.AddListener(ValidateUsername);
            ValidateUsername(usernameInput.text);
            registerButton.onClick.AddListener(Register);
        }

        private void Register()
        {
            StartCoroutine(RegisterFlow());
        }

        private IEnumerator RegisterFlow()
        {
            usernameInput.interactable = false;
```

```
            registerButton.interactable = false;
            yield return apiSender.CreateUser(usernameInput, r => {
CloseWindow(); });
        }

        private void CloseWindow()
        {
            usernameInput.interactable = true;
            registerButton.interactable = true;
            OnCloseClicked();
        }

        protected override void OnDisable()
        {
            usernameInput.onValueChanged.RemoveListener(ValidateUsername);
            registerButton.onClick.RemoveListener(Register);
        }

        private void ValidateUsername(string newName)
        {
            registerButton.interactable = IsUsernameValid(newName);
        }

        private char[] invalidChars = { '\n', '\t', ' ', '\\', '/', '"', '\'', '&' };

        private bool IsUsernameValid(string username)
        {
            username = username.Trim();
            if (username.Length < minUsernameLength) return false;
            if (username.Length > maxUsernameLength) return false;
            if (invalidChars.Any(c => username.Contains(c)) ) return false;


            return true;
        }
    }
}


using System;
using TMPro;
using UnityEngine;
using UnityEngine.UI;

namespace Scenes.Quiz.Prefabs.Scripts.UI
{
    public class RespawnUI : MonoBehaviour
    {
        [SerializeField]
        private Button continueButton;
        [SerializeField]
        private Button stopButton;


        [SerializeField]
        private PlayerController playerController;

        private void OnEnable()
        {
            continueButton.onClick.AddListener(ContinueGame);
            stopButton.onClick.AddListener(StopGame);
        }

        private void OnDisable()
        {
            continueButton.onClick.RemoveListener(ContinueGame);
            stopButton.onClick.RemoveListener(StopGame);
        }

        private void StopGame()
        {
            playerController.StopGame();
        }

        private void ContinueGame()
        {
            playerController.RespawnGame();
        }
    }
}


using System.Collections;
using System.Collections.Generic;
using Scenes.UI.Scripts;
using UnityEngine;
using UnityEngine.UI;

namespace Scenes.Quiz.Prefabs.Scripts.UI
{
    public class SettingsSpriteAnimation : UITweenerBase
    {
        [SerializeField]
        private float secondsPerFrame;

        [SerializeField]
        private List<Sprite> sprites;

        [SerializeField]
        private Image targetImage;

        private IEnumerator AnimationCoroutine(bool isReverse)
        {
            if (isReverse)
            {
                for (int i = sprites.Count -1; i >= 0; i--)
                {
                    targetImage.sprite = sprites[i];
                    yield return new WaitForSeconds(secondsPerFrame);
                }
            }
            else
            {
                for (int i = 0; i < sprites.Count; i++)
                {
                    targetImage.sprite = sprites[i];
                    yield return new WaitForSeconds(secondsPerFrame);
                }
            }
        }


        public override void DoForwardAnimation()
        {
            StartCoroutine(AnimationCoroutine(true));
        }

        public override void DoBackwardAnimation()
        {
            StartCoroutine(AnimationCoroutine(false));
        }

        public override float Duration => sprites.Count * secondsPerFrame;
        public override void ResetToStart()
        {
            targetImage.sprite = sprites[0];
        }
    }
}


using System;
using System.Collections.Generic;
using DG.Tweening;
using UnityEngine;
using UnityEngine.UI;

public class StartCounterUI : MonoBehaviour
{
    [SerializeField]
    private Slider starSlider;
    [SerializeField]
    private Image fillImage;

    private int _starsCount;
    private int _lastStarsCount;

    private void Init(int starsCount)
    {
        fillImage.color = starsCount == 0 ? Color.clear : Color.white;
        starSlider.value = starsCount;
        _lastStarsCount = starsCount;
    }

    public void UpdateStars(int starsCount, bool doAnimation = true)
    {
        _starsCount = starsCount;
        if (!doAnimation)
        {
            Init(starsCount);
            return;
        }

        if (_lastStarsCount == 0 && starsCount > 0)
        {
            DoColorAnimation(true);
        }
        else if (starsCount == 0)
```

```
        {
            DoColorAnimation(false);
        }

        starSlider.DOValue(starsCount, 0.3f);


        _lastStarsCount = starsCount;
    }

    private void DoColorAnimation(bool isOn)
    {
        var newColor = isOn ? Color.white : Color.clear;
        fillImage.DOColor(newColor, 0.3f);
    }

    public void DoWrongAnimation()
    {

    }

    private void SetStarsColor()
    {

    }
    private void PlayThreeStarAnimation()
    {

    }
}



using System;
using DG.Tweening;
using UnityEngine;
using UnityEngine.UI;

public class StarUI : MonoBehaviour
{
    [SerializeField]
    private Image starImage;

    [SerializeField] private Sprite enabledSprite;
    [SerializeField] private Sprite disabledSprite;
    [SerializeField] private Color threeStarsColor;
    private void Start()
    {
        UpdateVisual(false);
    }

    public void UpdateVisual(bool isSelected)
    {
        starImage.sprite = isSelected ? enabledSprite : disabledSprite;
        starImage.color = Color.white;
    }

    public void SetColor()
    {
        starImage.color = threeStarsColor;
    }

    public void DoJump()
    {
        var startPosition = starImage.rectTransform.localPosition;
        starImage.rectTransform.DOLocalJump(startPosition, 10f, 1, 0.3f);
    }

    public void DoWrongAnimation()
    {
        var startPosition = starImage.rectTransform.localPosition;
        starImage.rectTransform.DOLocalJump(startPosition, -10f, 1, 0.3f);
    }

    public void DoColor()
    {
        starImage.DOColor(threeStarsColor, 0.3f);
    }
}



using System;
using TMPro;
using UnityEngine;
using UnityEngine.UI;

namespace Quiz.Scripts.UI
{
    public class TopicItemUI : MonoBehaviour
    {
        [SerializeField]
        private Button selectButton;

        [SerializeField]
        private TextMeshProUGUI topicText;

        public ThemeData ThemeItem { get; private set; }
        public void Init(ThemeData theme, Action<TopicItemUI> onClick)
        {
            ThemeItem = theme;
            selectButton.interactable = true;
            topicText.text = theme.RussianWord;
            selectButton.onClick.AddListener(() => onClick?.Invoke(this));
        }

        public void Disable()
        {
            selectButton.interactable = false;
        }

        private void OnDestroy()
        {
            selectButton.onClick.RemoveAllListeners();
        }
    }
}



using UnityEngine;
using Scenes.GameScript;

namespace Scenes.Quiz.Prefabs.Scripts.UI
{
    public class TutorialUI : MonoBehaviour
    {
        [SerializeField]
        private TutorialController tutorialController;

        [Header("Hint Objects")]
        [SerializeField] private GameObject leftSwipeHint;
        [SerializeField] private GameObject rightSwipeHint;
        [SerializeField] private GameObject jumpHint;

        private void OnEnable()
        {
            if (tutorialController != null)
            {
                tutorialController.OnShowHint += ApplyHint;
            }
            // Ensure UI starts hidden until a hint is broadcast
            ApplyHint(TutorialController.Dir.None);
        }

        private void OnDisable()
        {
            if (tutorialController != null)
            {
                tutorialController.OnShowHint -= ApplyHint;
            }
        }

        private void ApplyHint(TutorialController.Dir dir)
        {
            SetActive(leftSwipeHint, dir == TutorialController.Dir.Left);
            SetActive(rightSwipeHint, dir == TutorialController.Dir.Right);
            SetActive(jumpHint, dir == TutorialController.Dir.Jump);
        }

        private static void SetActive(GameObject go, bool state)
        {
            if (go != null) go.SetActive(state);
        }
    }
}



using System;
using FIMSpace;
using TMPro;
using UnityEngine;
using UnityEngine.UI;

namespace Quiz.Scripts.UI
{
    public class WordItemUI : MonoBehaviour
```

```
    {
        [SerializeField]
        private Button selectButton;

        [SerializeField]
        private TextMeshProUGUI wordText;

        [SerializeField]
        private Image background;

        [SerializeField]
        private Color correctColor;

        [SerializeField]
        private Color wrongColor;

        [SerializeField]
        private Color defaultColor;

        public void Init(string topicName, Action<WordItemUI> onClick)
        {
            selectButton.interactable = true;
            background.color = defaultColor;
            selectButton.onClick.RemoveAllListeners();
            wordText.text = topicName.CapitalizeFirstLetter();
            selectButton.onClick.AddListener(() => onClick?.Invoke(this));
        }

        public void DisableButton()
        {
            selectButton.interactable = false;
        }

        public void DoCorrectAnimation()
        {
            background.color = correctColor;
        }

        public void DoWrongAnimation()
        {
            background.color = wrongColor;
        }



    }
}


using UnityEditor;
using UnityEngine;

namespace Scenes.UI.Editor
{
    [CustomEditor(typeof(AdvancedUITweener))]
    public class AdvancedUITweenerInspector : UnityEditor.Editor
    {
        private AdvancedUITweener uiTweener;

        private void OnEnable()
        {
            uiTweener = (AdvancedUITweener)target;
        }

        public override void OnInspectorGUI()
        {
            DrawDefaultInspector();

            EditorGUILayout.Space();

            if (!Application.isPlaying)
            {
                if (GUILayout.Button("Record State"))
                {
                    uiTweener.RecordState();
                    EditorUtility.SetDirty(uiTweener);
                }

                EditorGUILayout.Space();

                if (GUILayout.Button("Reset to Start Position"))
                {
                    uiTweener.SetStartTransform();
                }
            }
            else
            {
                if (GUILayout.Button("Play animation"))
                {
                    uiTweener.DoAnimation();
                }
            }
        }
    }
}



using UnityEditor;
using UnityEngine;

namespace Scenes.UI.Editor
{
    [CustomEditor(typeof(UITweener))]
    public class UITweenerInspector : UnityEditor.Editor
    {
        private UITweener uiTweener;

        private void OnEnable()
        {
            uiTweener = (UITweener)target;
        }

        public override void OnInspectorGUI()
        {
            DrawDefaultInspector();

            EditorGUILayout.Space();

            if (!Application.isPlaying)
            {
                if (GUILayout.Button("Record Start Position"))
                {
                    uiTweener.RecordStartTransform();
                    EditorUtility.SetDirty(uiTweener);
                }

                if (GUILayout.Button("Record End Position"))
                {
                    uiTweener.RecordEndTransform();
                    EditorUtility.SetDirty(uiTweener);
                }

                EditorGUILayout.Space();

                if (GUILayout.Button("Reset to Start Position"))
                {
                    uiTweener.SetStartTransform();
                }

                if (GUILayout.Button("Reset to End Position"))
                {
                    uiTweener.SetEndTransform();
                }
            }
        }
    }
}



using System;
using System.Collections.Generic;
using DG.Tweening;
using Scenes.UI.Scripts;
using UnityEngine;
using UnityEngine.Events;

namespace Scenes.UI
{
    public class AdvancedUITweener : UITweenerBase
    {
        [SerializeField] private RectTransform targetTransform;
        [Serializable]
        private class UiState
        {
            public Vector2 position;
            public Vector2 scale;
            public Vector2 anchorMax;
            public Vector2 anchorMin;
            public Vector2 sizeDelta;
            public Quaternion rotation;
            [Space]
            public float duration;
            public Ease ease;
        }

        [SerializeField]
```

```
    private List<UiState> uiSequence;

    // Start is called once before the first execution of Update after the
MonoBehaviour is created

    public void RecordState()
    {
        var state = new UiState();

        state.position = targetTransform.anchoredPosition;
        state.scale = targetTransform.localScale;
        state.anchorMax = targetTransform.anchorMax;
        state.anchorMin = targetTransform.anchorMin;
        state.sizeDelta = targetTransform.sizeDelta;
        state.rotation = targetTransform.rotation;

        uiSequence.Add(state);
    }

    public void DoAnimation()
    {
        var sequence = DOTween.Sequence();

        foreach (var state in uiSequence)
        {
            var ease = state.ease;
            sequence.Append(targetTransform.DOAnchorPos(state.position,
state.duration).SetEase(ease));
            sequence.Join(targetTransform.DOScale(state.scale,
state.duration).SetEase(ease));
            sequence.Join(targetTransform.DOAnchorMax(state.anchorMax,
state.duration).SetEase(ease));
            sequence.Join(targetTransform.DOAnchorMin(state.anchorMin,
state.duration).SetEase(ease));
            sequence.Join(targetTransform.DORotateQuaternion(state.rotation,
state.duration).SetEase(ease));
            sequence.Join(targetTransform.DOSizeDelta(state.sizeDelta,
state.duration).SetEase(ease));
        }

        sequence.OnComplete(() => onAnimationEnd.Invoke());

        sequence.PlayForward();
    }

    public void SetStartTransform()
    {
        var startState = uiSequence[0];

        targetTransform.anchoredPosition = startState.position;
        targetTransform.localScale = startState.scale;
        targetTransform.anchorMax = startState.anchorMax;
        targetTransform.anchorMin = startState.anchorMin;
        targetTransform.sizeDelta = startState.sizeDelta;
        targetTransform.rotation = startState.rotation;
    }

    private void OnEnable()
    {
        SetStartTransform();
    }

    public override void DoForwardAnimation()
    {
        DoAnimation();
    }

    public override void DoBackwardAnimation()
    {
        var sequence = DOTween.Sequence();

        foreach (var state in uiSequence)
        {
            var ease = state.ease;
            sequence.Append(targetTransform.DOAnchorPos(state.position,
state.duration).SetEase(ease));
            sequence.Join(targetTransform.DOScale(state.scale,
state.duration).SetEase(ease));
            sequence.Join(targetTransform.DOAnchorMax(state.anchorMax,
state.duration).SetEase(ease));
            sequence.Join(targetTransform.DOAnchorMin(state.anchorMin,
state.duration).SetEase(ease));
            sequence.Join(targetTransform.DORotateQuaternion(state.rotation,
state.duration).SetEase(ease));
            sequence.Join(targetTransform.DOSizeDelta(state.sizeDelta,
state.duration).SetEase(ease));
        }

        sequence.OnComplete(() => onAnimationEnd.Invoke());
        sequence.PlayBackwards();
    }

    public override float Duration
    {
        get
        {
            var totalDuration = 0f;
            foreach (var state in uiSequence)
            {
                totalDuration += state.duration;
            }
            return totalDuration;
        }
    }

    public override void ResetToStart()
    {
        SetStartTransform();
    }
  }
}



using System;
using UnityEngine;
using UnityEngine.Audio;

namespace Scenes.UI.Scripts
{
  // Handles audio volume logic and persistence. No UI responsibilities here.
  public class AudioSettings : MonoBehaviour
  {
    [Header("Mixer")]
    [SerializeField] private AudioMixer audioMixer;

    [Header("Volume Range (in Mixer dB)")]
    [SerializeField] private float volumeMax = 0f;     // 0 dB

    [SerializeField] private float volumeDefault = 0f;
    [SerializeField] private float volumeMin = -80f;   // -80 dB (mute)

    private void Start()
    {
        var defaultVolume = Mathf.Lerp(volumeMin, volumeMax,
volumeDefault);

        ApplyVolumeFromPrefs("MusicVolume", defaultVolume);
        ApplyVolumeFromPrefs("SoundVolume", defaultVolume);
        ApplyVolumeFromPrefs("VoiceVolume", defaultVolume);
    }

    public void ToggleVoiceover(bool isOn)
    {
        PlayerPrefs.SetInt("WordVoiceover", isOn ? 1 : 0);
        GlobalVar.WordVoiceover = isOn;
    }

    // Set volume for a given mixer parameter using normalized [0..1] value.
    public void SetVolume(string mixerParameterKey, float normalizedVolume)
    {
        if (audioMixer == null) return;
        float db = Mathf.Lerp(volumeMin, volumeMax,
Mathf.Clamp01(normalizedVolume));
        audioMixer.SetFloat(mixerParameterKey, db);
        if(normalizedVolume == 0f) audioMixer.SetFloat(mixerParameterKey, -
80f);
        PlayerPrefs.SetFloat(mixerParameterKey,
Mathf.Clamp01(normalizedVolume));
    }

    // Get saved normalized volume or default if not saved yet.
    public float GetSavedVolume(string mixerParameterKey, float @default =
1f)
    {
        return PlayerPrefs.GetFloat(mixerParameterKey,
Mathf.Clamp01(@default));
    }

    // Apply saved volume to mixer for a given key.
    public void ApplyVolumeFromPrefs(string mixerParameterKey, float
@default = 1f)
    {
        var normalized = GetSavedVolume(mixerParameterKey, @default);
        if (audioMixer == null) return;
        float db = Mathf.Lerp(volumeMin, volumeMax,
Mathf.Clamp01(normalized));
```

```
            audioMixer.SetFloat(mixerParameterKey, db);
        }
    }
}



using DG.Tweening;
using DG.Tweening.Core;
using DG.Tweening.Plugins.Options;
using TMPro;
using UnityEngine;
using UnityEngine.UI;

namespace Scenes.UI.Scripts
{
    public class BoosterUI : MonoBehaviour
    {
        [SerializeField]
        private Image boosterImage;
        [SerializeField]
        private Image boosterFill;
        [SerializeField]
        private TMP_Text boosterText;

        [SerializeField] private GameObject visuals;

        private TweenerCore<float, float, FloatOptions> boosterTween;

        public void SetBoosterSprite(Sprite sprite)
        {
            boosterImage.sprite = sprite;
        }
        public void ActivateBoosterUI(string busterLabel, float duration)
        {
            boosterTween?.Kill();

            visuals.SetActive(true);
            boosterText.text = busterLabel;
            boosterFill.fillAmount = 1f;
            boosterTween = boosterFill.DOFillAmount(0f,
duration).SetEase(Ease.Linear).OnComplete(() =>
            {
                visuals.SetActive(false);
            });
        }



        public void ClearBoosterUI()
        {
            visuals.SetActive(false);
        }
    }
}


using System;
using System.Collections;
using System.Collections.Generic;
using DG.Tweening;
using TMPro;
using UnityEngine;
using UnityEngine.UI;

namespace Scenes.UI.Scripts
{
    public enum Difficulty
    {
        Easy = 0,
        Normal = 1,
        Hard = 2
    }

    public class CollectWordUIController : MonoBehaviour
    {
        [Header("Собираемые слова")]
        [SerializeField] private TMP_Text collectWordText;
        [SerializeField] private UITweener scrollTweener;
        [SerializeField] private TMP_Text russianWordText;
        [SerializeField] private GameObject letterUI;
        [SerializeField] private RectTransform kazakhLettersParent;

        private float targetLetterAlpha = 0.6f;

        public void PlayWordOpenAnimation()
        {
            scrollTweener.DoForwardAnimation();
        }

        private IEnumerator SlowWordReveal()
        {
            foreach (var letter in letterUIItems)
            {
                letter.PlayShowAnimation(targetLetterAlpha);
                yield return new WaitForSeconds(0.1f);
            }
        }

        private IEnumerator SlowWordHide()
        {
            if(letterUIItems == null || letterUIItems.Count == 0) yield break;

            for (var i = 0; i < letterUIItems.Count; i++)
            {
                if(i >= letterUIItems.Count) break;
                var letter = letterUIItems[i];
                letter.PlayHideAnimation();
                yield return new WaitForSeconds(0.1f);
            }
        }

        public void PlayWordCloseAnimation()
        {
            scrollTweener.DoBackwardAnimation();
            StartCoroutine(SlowWordHide());
        }

        public void CollectLetter(int index)
        {
            HighlightLetter(index);
        }

        private WordData currentWord;
        public void SetUpWord(WordData wordData, Difficulty difficulty, int
missingIndex = 0)
        {
            currentWord = wordData;

            switch (difficulty)
            {
                case Difficulty.Easy:
                    DisplayWordEasy(wordData);
                    targetLetterAlpha = 0.6f;
                    break;
                case Difficulty.Normal:
                    DisplayWordMedium(wordData, missingIndex);
                    targetLetterAlpha = 1f;
                    break;
                case Difficulty.Hard:
                    DisplayWordHard(wordData);
                    targetLetterAlpha = 1f;
                    break;
                default:
                    throw new ArgumentOutOfRangeException(nameof(difficulty),
difficulty, null);
            }

            StartCoroutine(SlowWordReveal());
        }

        private void DisplayWordHard(WordData word)
        {
            russianWordText.text = word.RussianWord;

            var displayWord = new string('_', word.KazakhWord.Length);

            SetUpKazakhLetters(displayWord);
        }

        private void DisplayWordEasy(WordData word)
        {
            russianWordText.text = word.RussianWord;

            SetUpKazakhLetters(word.KazakhWord);
        }

        private void DisplayWordMedium(WordData word, int missingIndex)
        {
            russianWordText.text = word.RussianWord;

            var kazakhDisplay = $"{word.KazakhWord.Substring(0,
missingIndex)}_{word.KazakhWord.Substring(missingIndex + 1)}";

            SetUpKazakhLetters(kazakhDisplay);
        }
```

```
        private List<UILetter> letterUIItems;
        private void SetUpKazakhLetters(string word)
        {
            letterUIItems ??= new List<UILetter>();

            DestroyUILetters();

            foreach (var letter in word)
            {
                var letterObj = Instantiate(letterUI, kazakhLettersParent);
                var tmp = letterObj.GetComponent<UILetter>();
                tmp.Init(letter);
                letterUIItems.Add(tmp);
            }


            LayoutRebuilder.ForceRebuildLayoutImmediate(kazakhLettersParent);
        }

        private void DestroyUILetters(bool doForceRebuild = false)
        {
            if (letterUIItems.Count > 0)
            {
                foreach (var item in letterUIItems)
                {
                    Destroy(item.gameObject);
                }
                letterUIItems.Clear();
            }

            if(doForceRebuild)
LayoutRebuilder.ForceRebuildLayoutImmediate(kazakhLettersParent);
        }

        private void HighlightLetter(int letterIndex)
        {
            var letter = currentWord.KazakhWord[letterIndex];
            var letterUiItem = letterUIItems[letterIndex];
            letterUiItem.Reveal(letter);
        }

        public void CleanWordUI()
        {
            PlayWordCloseAnimation();
            DestroyUILetters(true);
        }

        public void RefreshWordDisplay()
        {
            //CleanWordUI();
        }
    }
}


using System;
using Scenes.Quiz.Prefabs.Scripts;
using TMPro;
using UnityEngine;
using UnityEngine.UI;

namespace Scenes.UI.Scripts
{
    public class HealthAndCoinUI : MonoBehaviour
    {
        [SerializeField] private TextMeshProUGUI wordCountText;
        [SerializeField] private AsyqLabel asyqLabel;

        [SerializeField] private Image coinFillImage;
        [SerializeField] private Image healthFillImage;


        private void Update()
        {
            var currentHealth =  GlobalVar.Health;
            var currentCoin = GlobalVar.Coin;
            var currentWords = GlobalVar.currCollectWord;

            var maxCoins = GlobalVar.AsyqtoCoin;
            var maxHealth = GlobalVar.MaxHealth;

            var coinFillAmount = (float)currentCoin/(float)maxCoins;
            var healthFillAmount = currentHealth/(float)maxHealth;

            coinFillImage.fillAmount = coinFillAmount;
            healthFillImage.fillAmount = healthFillAmount;

            asyqLabel.UpdateLabel();
            wordCountText.text = currentWords.ToString();
        }
    }
}


using System;
using TMPro;
using UnityEngine;
using UnityEngine.UI;

namespace Scenes.UI.Scripts
{
    public class LeaderboardItemUI : MonoBehaviour
    {
        [SerializeField]
        private TextMeshProUGUI nameText;
        [SerializeField]
        private TextMeshProUGUI wordText;
        [SerializeField]
        private TextMeshProUGUI timeText;
        [SerializeField]
        private TextMeshProUGUI placeText;
        [Space]
        [SerializeField]
        private Image placeImage;
        [Space]
        [SerializeField]
        private Sprite place1;
        [SerializeField]
        private Sprite place2;
        [SerializeField]
        private Sprite place3;
        [SerializeField]
        private Sprite defaultPlace;

        public void Init(LeaderboardItem item)
        {
            if (item == null)
                return;

            if (nameText != null)
                nameText.text = string.IsNullOrEmpty(item.Name) ? "—" : item.Name;

            if (wordText != null)
                wordText.text = item.WordCount.ToString();

            if (timeText != null)
                timeText.text = FormatTime(item.Time);

            if (placeText != null)
                placeText.text = item.Place > 3 ? item.Place.ToString() : string.Empty;

            if (placeImage != null)
            {
                placeImage.sprite = GetPlaceSprite(item.Place);
                placeImage.enabled = placeImage.sprite != null;
            }
        }

        private string FormatTime(TimeSpan ts)
        {
            if (ts == TimeSpan.Zero)
                return "00:00.00";

            return string.Format("{0:D2}:{1:D2}.{2:D2}",
                (int)ts.TotalMinutes,
                ts.Seconds,
                ts.Milliseconds);
        }

        private Sprite GetPlaceSprite(int place)
        {
            switch (place)
            {
                case 1: return place1 != null ? place1 : defaultPlace;
                case 2: return place2 != null ? place2 : defaultPlace;
                case 3: return place3 != null ? place3 : defaultPlace;
                default: return defaultPlace;
            }
        }
    }

    public class LeaderboardItem
    {
        public string Name;
        public int WordCount;
        public TimeSpan Time;
        public int Place;
```

```
        public static explicit operator LeaderboardItem(LeaderItem leaderItem)
        {
            var item = new LeaderboardItem();

            item.Name = leaderItem.login;
            item.WordCount = leaderItem.words;

            // split sting into min, sec, ms
            var split = leaderItem.session.Split(':');
            var split2 = split[2].Split('.');
            var sec = int.Parse(split2[0]);
            var ms = int.Parse(split2[1][..2]);
            item.Time = new TimeSpan(0, int.Parse(split[0]), int.Parse(split[1]) , sec,
ms);

            return item;
        }
    }
}


using System;
using System.Collections.Generic;
using System.Linq;
using UnityEngine;

namespace Scenes.UI.Scripts
{
    public class LeaderboardUI : MonoBehaviour
    {
        [SerializeField]
        private Transform content;
        [SerializeField]
        private GameObject leaderboardItem;
        private LeaderCache leaderCache => LeaderCache.Instance;

        private readonly List<GameObject> leaderboardItems = new
List<GameObject>();
        private List<LeaderboardItem> currentItems = new
List<LeaderboardItem>();

        private void OnEnable()
        {
            SetItems(leaderCache.GetLeaderboard());
        }

        public void SetItems(IEnumerable<LeaderboardItem> items)
        {
            currentItems = items != null ? items.ToList() : new
List<LeaderboardItem>();
            Refresh();
        }

        public void Refresh()
        {
            Clear();
            if (content == null || leaderboardItem == null)
                return;

            if (currentItems == null || currentItems.Count == 0)
                return;

            bool hasExplicitPlaces = currentItems.Any(i => i != null && i.Place > 0);

            IEnumerable<LeaderboardItem> ordered = hasExplicitPlaces
                ? currentItems.Where(i => i != null).OrderBy(i => i.Place)
                : currentItems.Where(i => i != null)
                    .OrderByDescending(i => i.WordCount)
                    .ThenBy(i => i.Time)
                    .ThenBy(i => i.Name);

            var top10 = ordered.Take(10).ToList();

            // If places were not provided, assign based on order
            if (!hasExplicitPlaces)
            {
                for (int idx = 0; idx < top10.Count; idx++)
                {
                    top10[idx].Place = idx + 1;
                }
            }

            foreach (var item in top10)
            {
                var go = Instantiate(leaderboardItem, content);
                leaderboardItems.Add(go);
                var ui = go.GetComponent<LeaderboardItemUI>();
                if (ui != null)
                {
                    ui.Init(item);
                }
            }
        }

        public void Clear()
        {
            // Destroy previously instantiated items
            for (int i = 0; i < leaderboardItems.Count; i++)
            {
                var go = leaderboardItems[i];
                if (go != null)
                {
                    Destroy(go);
                }
            }
            leaderboardItems.Clear();
        }
    }
}


using System;
using System.Collections.Generic;
using System.Linq;
using UnityEngine;

namespace Scenes.UI.Scripts
{
    public class LeaderCache : MonoBehaviour
    {
        [SerializeField]
        private ApiSender apiSender;
        private List<LeaderboardItem> cache = new List<LeaderboardItem>();

        public static LeaderCache Instance;

        private void Start()
        {
            Instance = this;
            GetDataFromApi();
        }


        public List<LeaderboardItem> GetLeaderboard()
        {
            RefreshCache();
            return cache;
        }

        private void RefreshCache()
        {
            GetDataFromApi();
        }

        private void GetDataFromApi()
        {
            var items = new List<LeaderboardItem>();
            apiSender.TopPlayers((lb) =>
            {
                if(items == null) return;
                items = lb.Select(x => (LeaderboardItem)x).ToList();
                cache = items;
            });;
        }

    }
}


using System;
using DG.Tweening;
using UnityEngine;
using UnityEngine.Events;
using UnityEngine.UI;
using UnityEngine.EventSystems;

namespace Scenes.UI.Scripts
{
    public enum Menus
    {
        MainMenu,
        DictionaryMenu,
        QuizMenu,
        LeaderMenu,
        SkinMenu,
        GameMenu
    }
```

```
    public class MainMenuButton : MonoBehaviour, IPointerClickHandler
    {
        [SerializeField]
        private UITweener ImageTweener;

        [SerializeField] private Image bgImage;
        [SerializeField] private Image iconImage;
        [SerializeField] private Sprite activeSprite;
        [SerializeField] private Color activeColor;
        [SerializeField] private Color inactiveColor;

        [SerializeField] private Color disabledColor;
        [SerializeField] private Sprite disabledSprite;

        [SerializeField] private Button selectButton;
        [SerializeField] private Vector2 activeSize;
        [SerializeField] private Vector2 inactiveSize;
        [SerializeField] private LayoutElement layoutElement;
        [SerializeField] private float duration = 0.2f;
        public bool IsSelected { get; set; }

        public Menus Menu;

        public UnityEvent<MainMenuButton> OnSelectButtonClicked;
        public UnityEvent<MainMenuButton> OnDisabledButtonClicked;

        private void OnEnable()
        {
            selectButton.onClick.AddListener(OnSelectClicked);
        }

        private void OnDisable()
        {
            selectButton.onClick.RemoveListener(OnSelectClicked);
        }

        public void DisableButton()
        {
            selectButton.interactable = false;
            iconImage.sprite = disabledSprite;
            bgImage.color = disabledColor;
        }

        public void EnableButton()
        {
            selectButton.interactable = true;
            iconImage.sprite = activeSprite;
            bgImage.color = activeColor;
        }

        private void OnSelectClicked()
        {
            OnSelectButtonClicked?.Invoke(this);
        }

        public void OnPointerClick(PointerEventData eventData)
        {
            if (!selectButton.interactable)
            {
                OnDisabledButtonClicked?.Invoke(this);
            }
        }

        public void Select()
        {
            IsSelected = true;
            layoutElement.DOPreferredSize(activeSize, duration);
            bgImage.DOColor(activeColor, duration);
            ImageTweener.DoForwardAnimation();
        }

        public void Deselect()
        {
            IsSelected = false;
            layoutElement.DOPreferredSize(inactiveSize, duration);
            bgImage.DOColor(inactiveColor, duration);
            ImageTweener.DoBackwardAnimation();
        }
    }
}



using System;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.Events;

namespace Scenes.UI.Scripts
{
    public class MainMenuSelectGroup : MonoBehaviour
    {
        [SerializeField]
        private List<MainMenuButton> buttons;
        [SerializeField]
        private MainMenuButton defaultButton;

        public MainMenuButton SelectedButton { get; private set; }

        public UnityEvent<MainMenuButton> OnSelectButtonClicked;

        private void OnEnable()
        {
            foreach (var btn in buttons)
            {
                btn.OnSelectButtonClicked.AddListener(SelectNewButton);
            }

            SelectNewButton(defaultButton);
        }

        private void OnDisable()
        {
            foreach (var btn in buttons)
            {
                btn.OnSelectButtonClicked.RemoveListener(SelectNewButton);
            }
        }

        public void DisableButton(MainMenuButton button)
        {
            button.DisableButton();
        }

        public void EnableButton(MainMenuButton button)
        {
            button.EnableButton();
        }

        private void SelectNewButton(MainMenuButton button)
        {
            if(SelectedButton == button) return;

            if(SelectedButton) SelectedButton.Deselect();
            button.Select();
            SelectedButton = button;

            OnSelectButtonClicked?.Invoke(button);
        }
    }
}



using System;
using System.Collections;
using Quiz.Scripts;
using UnityEngine;
using UnityEngine.UI;

namespace Scenes.UI.Scripts
{
    public class NavBarController : MonoBehaviour
    {
        [SerializeField] private RectTransform canvasParent;
        [SerializeField] private MainMenuSelectGroup mainMenuSelectGroup;
        [SerializeField] private MainMenuRefs mainMenuRefs;
        [SerializeField] private MenuController menuController;
        [SerializeField] private UITweenerBase closeTweener;
        [SerializeField] private UITweenerBase settingsTweener;
        [SerializeField] private UITweenerBase pauseTweener;
        [SerializeField] private GraphicRaycaster raycaster;

        private GameObject mainMenuCanvas;
        [Header("Префабы")]
        [SerializeField] private GameObject dictCanvasPrefab;
        [SerializeField] private GameObject quizCanvasPrefab;
        [SerializeField] private GameObject leaderCanvasPrefab;

        [Header("UI")]
        [SerializeField] private GameObject gameCanvas;
        [SerializeField] private GameObject skinCanvas;

        [SerializeField] private GameObject settingCanvas;
        [SerializeField] private GameObject pauseCanvas;
        [SerializeField] private Button settingButton;
        [SerializeField] private Button pauseButton;
        [SerializeField] private float pauseAnimationDelay;
```

```
[Header("Викторина")]
[SerializeField] private MainMenuButton quizButton;
[SerializeField] private GameObject quizLockedPopup;
[SerializeField] private WordGameConfig wordGameConfig;

private GameObject dictCanvas;
private GameObject quizCanvas;
private GameObject leaderCanvas;

private void Start()
{
    mainMenuCanvas = menuController.mainMenu;

    InitCanvas();
    SubscribeEvents();
}

public void CloseNavBar()
{
    closeTweener.DoBackwardAnimation();
    settingsTweener.DoBackwardAnimation();
    raycaster.enabled = false;
    StartCoroutine(ShowPauseWithDelay(pauseAnimationDelay));
    DisableAllCanvas();
}

private IEnumerator ShowPauseWithDelay(float delay)
{
    yield return new WaitForSeconds(delay);
    raycaster.enabled = true;
    pauseButton.interactable = true;
    pauseTweener.DoForwardAnimation();
}

public void OpenNavBar()
{
    closeTweener.DoForwardAnimation();
    settingsTweener.DoForwardAnimation();
    pauseTweener.DoBackwardAnimation();
    raycaster.enabled = true;
    CheckWordCount();
    StartCoroutine(EnableStartButtonWithDelay(pauseAnimationDelay));
    OnMenuSelected(mainMenuSelectGroup.SelectedButton);
}

private IEnumerator EnableStartButtonWithDelay(float delay)
{
    yield return new WaitForSecondsRealtime(delay + 0.3f);
    mainMenuRefs.EnableButtons();
}

public void DisablePauseButton()
{
    pauseButton.interactable = false;
}

private void InitCanvas()
{
    mainMenuCanvas.transform.SetParent(canvasParent);
    gameCanvas.transform.SetParent(canvasParent);
    dictCanvas = Instantiate(dictCanvasPrefab, canvasParent);
    quizCanvas = Instantiate(quizCanvasPrefab, canvasParent);
    leaderCanvas = Instantiate(leaderCanvasPrefab, canvasParent);

    DisableAllCanvas();

    gameCanvas.SetActive(false);
    mainMenuCanvas.SetActive(true);

    settingButton.onClick.AddListener(OnSettingsButtonClicked);
    pauseButton.onClick.AddListener(OnPauseButtonClicked);

    CheckWordCount();
    //force rebuild layout
    LayoutRebuilder.ForceRebuildLayoutImmediate(canvasParent);
}

private void OnSettingsButtonClicked()
{
    settingCanvas.SetActive(true);

}

private void OnPauseButtonClicked()
{
    pauseCanvas.SetActive(true);
}

    private void CheckWordCount()
    {
        if(PlayerPrefs.GetInt("QuizUnlocked", 0) == 1) return;

        if (GlobalVar.GetCollectedWordsCount() <
wordGameConfig.wordGameUnlockCount)
        {
            mainMenuSelectGroup.DisableButton(quizButton);
        }
        else
        {
            mainMenuSelectGroup.EnableButton(quizButton);
            PlayerPrefs.SetInt("QuizUnlocked", 1);
            menuController.ShowQuizUnlockPopup();
        }
    }

    private void DisableAllCanvas()
    {
        mainMenuCanvas.SetActive(false);
        skinCanvas.SetActive(false);
        dictCanvas.SetActive(false);
        quizCanvas.SetActive(false);
        gameCanvas.SetActive(false);
        leaderCanvas.SetActive(false);
    }

    private void SubscribeEvents()
    {

mainMenuSelectGroup.OnSelectButtonClicked.AddListener(OnMenuSelected);

mainMenuCanvas.GetComponent<MainMenuRefs>().BtnStart.onClick.AddListe
ner(OnStartButtonClicked);

mainMenuCanvas.GetComponent<MainMenuRefs>().BtnStart2.onClick.AddList
ener(OnStartButtonClicked);

        if (quizButton != null)
        {
            // When quiz is disabled, clicking it should show the unlock popup

quizButton.OnDisabledButtonClicked.AddListener(OnDisabledMenuButtonClick
ed);
        }
    }

    private void OnStartButtonClicked()
    {
        CloseNavBar();
        SwitchToMenu(Menus.GameMenu);
    }

    public void SwitchToMenu(Menus menu)
    {
        DisableAllCanvas();
        OpenMenu(menu);
    }

    private void OnMenuSelected(MainMenuButton button)
    {
        SwitchToMenu(button.Menu);
    }

    private void OnDisabledMenuButtonClicked(MainMenuButton button)
    {
        if (button == quizButton)
        {
            ShowQuizPopup();
        }
    }

    public void ShowQuizPopup()
    {
        quizLockedPopup.SetActive(true);
    }

    private void OpenMenu(Menus menu)
    {
        var theme = MainTheme.Menu;

        switch (menu)
        {
            case Menus.MainMenu:
                mainMenuCanvas.SetActive(true);
                break;
            case Menus.DictionaryMenu:
                dictCanvas.SetActive(true);
                theme = MainTheme.Quiz;
```

```
                break;
            case Menus.QuizMenu:
                quizCanvas.SetActive(true);
                theme = MainTheme.Quiz;
                break;
            case Menus.LeaderMenu:
                leaderCanvas.SetActive(true);
                break;
            case Menus.SkinMenu:
                skinCanvas.SetActive(true);
                break;
            case Menus.GameMenu:
                gameCanvas.SetActive(true);
                break;
            default:
                throw new ArgumentOutOfRangeException();
        }

        menuController.PlayMainTheam(theme);
    }
  }
}


using UnityEngine;
using UnityEngine.UI;

public class PlatformSpecificSprite : MonoBehaviour
{

  [SerializeField] private Image image;

  [SerializeField] private Sprite androidSprite;
  [SerializeField] private Sprite iosSprite;

  // Start is called once before the first execution of Update after the
MonoBehaviour is created
  void Start()
  {
    #if UNITY_ANDROID
    image.sprite = androidSprite;
    #endif

    #if Unity_IOS
    image.sprite = iosSprite;
    #endif
  }
}



using Scenes.GameScript;
using Scenes.Object.UI.Translater;
using TMPro;
using UnityEngine;
using UnityEngine.Audio;
using UnityEngine.UI;

namespace Scenes.UI.Scripts
{
  [RequireComponent(typeof(AudioSettings))]
  public class SettingsUI : MonoBehaviour
  {
    [SerializeField]
    private Button backButton;

    [Header("Громкость")]
    [SerializeField] private AudioSettings audioSettings;

    [SerializeField] private Slider musicSlider;
    [SerializeField] private Slider sfxSlider;
    [SerializeField] private Slider voiceSlider;
    [SerializeField] private Toggle voiceToggle;

    [Header("Язык")]
    [SerializeField] private Toggle toggleRu;
    [SerializeField] private Toggle toggleKz;
    [SerializeField] private ToggleGraphic toggleKzGraphic;
    [SerializeField] private ToggleGraphic toggleRuGraphic;

    [Header("Сложность")]
    [SerializeField] private Button difficultySelectButton;
    [SerializeField] private TextMeshProUGUI difficultyText;
    [SerializeField] private GameObject difficultySelectPanel;
    [Space]
    [SerializeField] private Button easyButton;
    [SerializeField] private Button normalButton;
    [SerializeField] private Button hardButton;

    private void Awake()
    {
      if (audioSettings == null)
        audioSettings = GetComponent<AudioSettings>();
    }

    private void OnEnable()
    {
      Time.timeScale = 0;
      // Ensure settings are loaded and GlobalVar is initialized
      SettingLoader.Initialize();
      Init();
    }

    private void OnLanguageChanged(int lang)
    {
      UpdateLabels();
    }

    private void OnDisable()
    {
      Time.timeScale = 1;
      UnsubscribeListeners();
    }



    private void ResetSliders()
    {
      musicSlider.value = audioSettings.GetSavedVolume("MusicVolume",
1f);
      sfxSlider.value = audioSettings.GetSavedVolume("SoundVolume", 1f);
      voiceSlider.value = audioSettings.GetSavedVolume("VoiceVolume", 1f);
    }

    private void InitVolume()
    {
      ResetSliders();

      musicSlider.onValueChanged.AddListener((volume) =>
audioSettings.SetVolume("MusicVolume", volume));
      sfxSlider.onValueChanged.AddListener((volume) =>
audioSettings.SetVolume("SoundVolume", volume));
      voiceSlider.onValueChanged.AddListener((volume) =>
audioSettings.SetVolume("VoiceVolume", volume));
    }

    private void UnsubscribeListeners()
    {
      musicSlider.onValueChanged.RemoveAllListeners();
      sfxSlider.onValueChanged.RemoveAllListeners();
      voiceSlider.onValueChanged.RemoveAllListeners();

      toggleRu.onValueChanged.RemoveAllListeners();
      toggleKz.onValueChanged.RemoveAllListeners();


voiceToggle.onValueChanged.RemoveListener(audioSettings.ToggleVoiceover
);

      difficultySelectButton.onClick.RemoveAllListeners();

      easyButton.onClick.RemoveAllListeners();
      normalButton.onClick.RemoveAllListeners();
      hardButton.onClick.RemoveAllListeners();

      backButton.onClick.RemoveAllListeners();


LocalizationManager.Instance.OnLanguageChanged.RemoveListener(OnLangu
ageChanged);
    }

    private void InitLanguage()
    {
      toggleKz.onValueChanged.AddListener(isOn => { if (isOn)
ChangeLang(0); });
      toggleRu.onValueChanged.AddListener(isOn => { if (isOn)
ChangeLang(1); });

LocalizationManager.Instance.OnLanguageChanged.AddListener(OnLanguage
Changed);

      // Apply current language loaded into GlobalVar by SettingLoader
      LocalizationManager.Instance.UpdateLanguage(GlobalVar.Lang);

      ChangeLangToggle();
    }
```

```
        private void ChangeLangToggle()
        {
            var index = GlobalVar.Lang;
            if (index == 0)
            {
                toggleKz.isOn = true;
                toggleKzGraphic.UpdateVisuals(true);
            }
            else
            {
                toggleRu.isOn = true;
                toggleRuGraphic.UpdateVisuals(true);
            }
        }

        public void ChangeLang(int index)
        {
            PlayerPrefs.SetInt("Lang", index);
            GlobalVar.Lang = index;
            LocalizationManager.Instance.UpdateLanguage(index);
        }

        private void OpenDifficultySelectPanel()
        {
            difficultySelectPanel.SetActive(true);
        }

        private void CloseDifficultySelectPanel()
        {
            difficultySelectPanel.SetActive(false);
        }

        private void InitDifficulty()
        {
            difficultySelectButton.onClick.AddListener(OpenDifficultySelectPanel);
            easyButton.onClick.AddListener(() => { SetDifficulty(Difficulty.Easy); });
            normalButton.onClick.AddListener(() => { SetDifficulty(Difficulty.Normal);
});
            hardButton.onClick.AddListener(() => { SetDifficulty(Difficulty.Hard); });

            SetDifficulty((Difficulty)GlobalVar.difficulty);
        }

        private void SetDifficulty(Difficulty difficulty)
        {
            GlobalVar.difficulty = (int)difficulty;
            PlayerPrefs.SetInt("difficulty", GlobalVar.difficulty);
            UpdateLabels();

            CloseDifficultySelectPanel();
        }

        private void UpdateLabels()
        {
            var difficulty = (Difficulty)GlobalVar.difficulty;
            difficultyText.text = $"{"Difficulty".Localize()}:
{difficulty.ToString().Localize()}";
        }

        private void Init()
        {
            InitVolume();
            InitLanguage();
            InitDifficulty();
            InitVoiceover();

            backButton.onClick.AddListener(() => gameObject.SetActive(false));
        }

        private void InitVoiceover()
        {
            voiceToggle.isOn = GlobalVar.WordVoiceover;
            voiceToggle.SetIsOnWithoutNotify(GlobalVar.WordVoiceover);


voiceToggle.onValueChanged.AddListener(audioSettings.ToggleVoiceover);
        }
    }
}


using TMPro;
using UnityEngine;

namespace Scenes.UI.Scripts
{
    public class TileUIController : MonoBehaviour
    {
        [Header("Название локации")]
        [SerializeField] private TMP_Text tileName;
        [SerializeField] private UITweener tileAnimator;

        public void PlayOpenAnimation()
        {
            tileAnimator.DoForwardAnimation();
        }

        public void PlayCloseAnimation()
        {
            tileAnimator.DoBackwardAnimation();
        }

        public void UpdateTileName(string newName)
        {
            PlayOpenAnimation();
            tileName.text = newName;
        }
    }
}


using System;
using UnityEngine;
using UnityEngine.UI;

namespace Scenes.UI.Scripts
{
    public class ToggleGraphic : MonoBehaviour
    {
        [SerializeField] private Image image;
        [SerializeField] private Sprite onSprite;
        [SerializeField] private Sprite offSprite;

        private Toggle toggle;

        private void Start()
        {
            toggle = GetComponent<Toggle>();
        }

        private void OnEnable()
        {
            if(!toggle) toggle = GetComponent<Toggle>();
            UpdateVisuals(toggle.isOn);
            toggle.onValueChanged.AddListener(UpdateVisuals);
        }

        private void OnDisable()
        {
            toggle.onValueChanged.RemoveListener(UpdateVisuals);
        }

        public void UpdateVisuals(bool isOn)
        {
            image.sprite = isOn ? onSprite : offSprite;
        }
    }
}


using System;
using System.Collections;
using System.Collections.Generic;
using Scenes.GameScript;
using UnityEngine;
using UnityEngine.UI;

namespace Scenes.UI.Scripts
{
    [Serializable]
    public class TutorialStep
    {
        [SerializeField] private string name;
        [SerializeField] public List<GameObject> HighLights;
        [SerializeField] public List<GameObject> EnableObjects;
        [SerializeField] public Vector2 maskPosition;
        [SerializeField] public bool hasDarkBg;
    }

    public class TutorialPlayer : MonoBehaviour
    {
        [SerializeField] private TutorialStep[] Steps;
        [SerializeField] private UIHighlighter UIHighlighter;
        [SerializeField] private RectTransform mask;
        private int currentStep;

        [SerializeField] private Button StartButton;
        [SerializeField] private Button StartButton2;
```

```
[SerializeField] private TutorialController tutorialController;
[SerializeField] private float lookAheadDistance;
[SerializeField] private Button skipButton;

private void Start()
{
    if (!tutorialController.IsTutorialActive)
    {
        UIHighlighter.SetEnableDarkBg(false);
        return;
    }

    currentStep = -1;
    StartNextStep();
}

private void StartNextStep()
{
    if(currentStep >= 0) DeactictivateStep(Steps[currentStep]);
    currentStep++;
    ActivateStep(Steps[currentStep]);
    HandleStep(currentStep);
}

private void DeactictivateStep(TutorialStep step)
{
    foreach (var higlight in step.HighLights)
    {
        UIHighlighter.UnhighLightObject(higlight);
    }

    foreach (var enable in step.EnableObjects)
    {
        enable.SetActive(false);
    }


}



private void ActivateStep(TutorialStep step)
{
    foreach (var highlight in step.HighLights)
    {
        UIHighlighter.HighLightObject(highlight);
    }

    foreach (var enable in step.EnableObjects)
    {
        enable.SetActive(true);
    }

    mask.anchoredPosition = step.maskPosition;

    UIHighlighter.SetEnableDarkBg(step.hasDarkBg);
}

private void HandleStep(int step)
{
    switch (step)
    {
        case 0:
            StartButton.onClick.AddListener(StartNextStep);
            StartButton2.onClick.AddListener(StartNextStep);
            break;
        case 1:
            StartButton.onClick.RemoveListener(StartNextStep);
            StartButton2.onClick.RemoveListener(StartNextStep);
            StartCoroutine(DetectCoinCoroutine());
            break;
        case 2:
            Time.timeScale = 0f;
            skipButton.onClick.AddListener(StartNextStep);
            break;
        case 3:
            Time.timeScale = 1f;
            skipButton.onClick.RemoveListener(StartNextStep);
            StartCoroutine(DetectEnemyCoroutine());
            break;
        case 4:
            Time.timeScale = 0f;
            skipButton.onClick.AddListener(StartNextStep);
            break;
        case 5:
            Time.timeScale = 1f;
            skipButton.onClick.RemoveListener(StartNextStep);
            tutorialController.CompleteTutorial();
            break;
        default:
            break;
    }
}

private IEnumerator DetectEnemyCoroutine()
{
    while (true)
    {
        if (tutorialController.IsEnemyAhead(out RaycastHit hit, 10f))
        {
            StartNextStep();
            var worldPoint = hit.point;
            var screenPoint = Camera.main.WorldToScreenPoint(worldPoint);

            mask.position = screenPoint;
            yield break;
        }

        yield return new WaitForEndOfFrame();
    }
}

private IEnumerator DetectCoinCoroutine()
{
    while (true)
    {
        if (tutorialController.IsCoinAhead(lookAheadDistance, out RaycastHit
hit))
        {
            StartNextStep();
            var worldPoint = hit.point;
            var screenPoint = Camera.main.WorldToScreenPoint(worldPoint);

            mask.position = screenPoint;
            yield break;
        }

        yield return new WaitForEndOfFrame();
    }
}
}
}



using System.Collections.Generic;
using DG.Tweening;
using UnityEngine;

namespace Scenes.UI.Scripts
{
    public class UIComboTweener : UITweenerBase
    {
        [SerializeField]
        private List<UITweenerBase> tweens;


        public override float Duration{
            get
            {
                var maxDuration = 0f;
                foreach (var tween in tweens)
                {
                    maxDuration = Mathf.Max(maxDuration, tween.Duration);
                }

                return maxDuration;
            }
        }

        public override void ResetToStart()
        {
            foreach (var tween in tweens)
            {
                tween.ResetToStart();
            }
        }

        public override void DoForwardAnimation()
        {
            foreach (var tween in tweens)
            {
                tween.DoForwardAnimation();
            }
            //wait a duration
            var seq = DOTween.Sequence();
            seq.AppendInterval(Duration);
            seq.OnComplete(() =>
```

```
            {
                onAnimationEnd?.Invoke();
            });
        }

        public override void DoBackwardAnimation()
        {
            foreach (var tween in tweens)
            {
                tween.DoBackwardAnimation();
            }
            var seq = DOTween.Sequence();
            seq.AppendInterval(Duration);
            seq.OnComplete(() =>
            {
                onAnimationEnd?.Invoke();
            });
        }

    }
}


using System;
using DG.Tweening;
using UnityEngine;
using UnityEngine.UI;

public class UIGlow : MonoBehaviour
{
    [SerializeField]
    private Image glowImage;
    [SerializeField]
    private float glowDuration = 0.2f;

    private void OnEnable()
    {
        glowImage.DOFade(0f, 0f);
    }

    public void DoForward()
    {
        glowImage.DOFade(1f, glowDuration);
    }

    public void DoBackward()
    {
        glowImage.DOFade(0f, glowDuration);
    }
}



using System.Collections;
using UnityEngine;
using UnityEngine.UI;

namespace Scenes.UI.Scripts
{
    public class UIHighlighter : MonoBehaviour
    {
        [SerializeField] private GameObject darkBG;
        [SerializeField] private int highlightSortOrder;


        public void SetEnableDarkBg(bool enable)
        {
            darkBG.SetActive(enable);
        }

        public void HighLightObject(GameObject obj)
        {
            StartCoroutine(HighlightOverTime(obj));
        }

        private IEnumerator HighlightOverTime(GameObject obj)
        {
            var tutorialCanvas = obj.AddComponent<Canvas>();
            yield return new WaitForEndOfFrame();
            tutorialCanvas.overrideSorting = true;
            tutorialCanvas.sortingOrder = highlightSortOrder;

            obj.AddComponent<GraphicRaycaster>();
        }

        public void UnhighLightObject(GameObject obj)
        {
            var tutorialRaycaster = obj.GetComponent<GraphicRaycaster>();
            Destroy(tutorialRaycaster);
            var tutorialCanvas = obj.GetComponent<Canvas>();
            Destroy(tutorialCanvas);
        }
    }
}


using System;
using System.Collections.Generic;
using System.Linq;
using DG.Tweening;
using DG.Tweening.Plugins.Core.PathCore;
using UnityEngine;

namespace Scenes.UI.Scripts
{
    public class UIPathTweener : UITweenerBase
    {
        private enum PathTypeInternal
        {
            Linear = 0,
            CatmullRom = 1,
        }

        [SerializeField]
        private Transform targetTransform;
        [SerializeField]
        private float duration;
        [Space]
        [SerializeField]
        private List<Transform> path;
        [SerializeField]
        private PathTypeInternal pathType;
        [SerializeField]
        private Ease ease;

        private Path pathData;
        private Path revertedPathData;

        private void Start()
        {
            BuildPath();
        }

        private void BuildPath()
        {
            pathData = new Path((PathType)pathType, path.Select(x =>
x.position).ToArray(), 1);
            revertedPathData = new Path((PathType)pathType, path.Select(x =>
x.position).Reverse().ToArray(), 1);
        }

        public override void DoForwardAnimation()
        {
            if(pathData == null) BuildPath();

            targetTransform.DOPath(pathData, duration)
                .SetEase(ease)
                .OnComplete(() =>
                {
                    onAnimationEnd?.Invoke();
                });
        }

        public override void DoBackwardAnimation()
        {
            if(revertedPathData == null) BuildPath();

            targetTransform.DOPath(revertedPathData, duration)
                .SetEase(ease)
                .OnComplete(() =>
                {
                    onAnimationEnd?.Invoke();
                });
        }

        public override float Duration => duration;
        public override void ResetToStart()
        {
            targetTransform.position = path[0].position;
        }
    }
}


using UnityEngine;
using UnityEngine.UI;
```

```
namespace Scenes.UI.Scripts
{
    public class UiScroll : MonoBehaviour
    {
        [SerializeField]
        private Vector2 scrollSpeed;
        [SerializeField]
        private Vector2 scrollMod;
        [SerializeField]
        private RawImage image;

        private void Update()
        {
            var ypos = (Time.time * scrollSpeed.y)%scrollMod.y;
            var xpos = (Time.time * scrollSpeed.x)%scrollMod.x;
            var rect = image.uvRect;
            rect.y = ypos;
            rect.x = xpos;

            image.uvRect = rect;
        }
    }
}
```

```
using System;
using DG.Tweening;
using Scenes.UI.Scripts;
using UnityEngine;

namespace Scenes.UI
{
    public class UITweener : UITweenerBase
    {
        [SerializeField] private float duration;
        [SerializeField] private RectTransform targetTransform;
        [SerializeField] private bool resetOnEnable = true;
        [SerializeField] private Ease ease;
        [SerializeField] private bool useUnscaledTime;

        [Header("Config")]
        [SerializeField] private bool doPosition;
        [SerializeField] private bool doScale;
        [SerializeField] private bool doSizeDelta;
        [SerializeField] private bool doRotation;
        [SerializeField] private bool doAnchors;

        [Header("Start")]
        [SerializeField] private Vector2 startPosition;
        [SerializeField] private Vector2 startScale;
        [SerializeField] private Vector2 startAnchorMax;
        [SerializeField] private Vector2 startAnchorMin;
        [SerializeField] private Vector2 startSizeDelta;
        [SerializeField] private Quaternion startRotation;

        [Header("End")]
        [SerializeField] private Vector2 endPosition;
        [SerializeField] private Vector2 endScale;
        [SerializeField] private Vector2 endAnchorMax;
        [SerializeField] private Vector2 endAnchorMin;
        [SerializeField] private Vector2 endSizeDelta;
        [SerializeField] private Quaternion endRotation;

        // Start is called once before the first execution of Update after the
MonoBehaviour is created

        public void RecordStartTransform()
        {
            if(targetTransform != null)
            {
                if(doPosition) startPosition = targetTransform.anchoredPosition;
                if(doScale) startScale = targetTransform.localScale;
                if(doAnchors) startAnchorMax = targetTransform.anchorMax;
                if(doAnchors) startAnchorMin = targetTransform.anchorMin;
                if(doSizeDelta) startSizeDelta = targetTransform.sizeDelta;
                if(doRotation) startRotation = targetTransform.rotation;
            }
        }

        public void RecordEndTransform()
        {
            if(targetTransform != null)
            {
                if(doPosition) endPosition = targetTransform.anchoredPosition;
                if(doScale) endScale = targetTransform.localScale;
                if(doAnchors) endAnchorMax = targetTransform.anchorMax;
                if(doAnchors) endAnchorMin = targetTransform.anchorMin;
                if(doSizeDelta) endSizeDelta = targetTransform.sizeDelta;
                if(doRotation) endRotation = targetTransform.rotation;
            }
        }

        public override void DoForwardAnimation()
        {
            if(!isActiveAndEnabled) return;
            if(targetTransform == null) return;

            if (doSizeDelta) targetTransform.DOSizeDelta(endSizeDelta,
duration).SetEase(ease).SetUpdate(true);
            if(doAnchors) targetTransform.DOAnchorMax(endAnchorMax,
duration).SetEase(ease).SetUpdate(true);
            if(doAnchors) targetTransform.DOAnchorMin(endAnchorMin,
duration).SetEase(ease).SetUpdate(true);
            if(doPosition) targetTransform.DOAnchorPos(endPosition,
duration).SetEase(ease).SetUpdate(true);
            if(doScale) targetTransform.DOScale(endScale,
duration).SetEase(ease).SetUpdate(true);
            if(doRotation) targetTransform.DORotateQuaternion(endRotation,
duration).SetEase(ease).SetUpdate(true);
        }

        public override void DoBackwardAnimation()
        {
            if(!isActiveAndEnabled) return;
            if(targetTransform == null) return;

            if(doSizeDelta) targetTransform.DOSizeDelta(startSizeDelta,
duration).SetEase(ease).SetUpdate(true);
            if(doAnchors) targetTransform.DOAnchorMax(startAnchorMax,
duration).SetEase(ease).SetUpdate(true);
            if(doAnchors) targetTransform.DOAnchorMin(startAnchorMin,
duration).SetEase(ease).SetUpdate(true);
            if(doPosition) targetTransform.DOAnchorPos(startPosition,
duration).SetEase(ease).SetUpdate(true);
            if(doScale) targetTransform.DOScale(startScale,
duration).SetEase(ease).SetUpdate(true);
            if(doRotation) targetTransform.DORotateQuaternion(startRotation,
duration).SetEase(ease).SetUpdate(true);
        }

        public override float Duration => duration;
        public override void ResetToStart()
        {
            SetStartTransform();
        }

        public void SetStartTransform()
        {
            if(targetTransform == null) return;

            if(doPosition) targetTransform.anchoredPosition = startPosition;
            if(doScale) targetTransform.localScale = startScale;
            if(doAnchors) targetTransform.anchorMax = startAnchorMax;
            if(doAnchors) targetTransform.anchorMin = startAnchorMin;
            if(doSizeDelta) targetTransform.sizeDelta = startSizeDelta;
            if(doRotation) targetTransform.rotation = startRotation;
        }

        public void SetEndTransform()
        {
            if(targetTransform == null) return;

            if(doPosition) targetTransform.anchoredPosition = endPosition;
            if(doScale) targetTransform.localScale = endScale;
            if(doAnchors) targetTransform.anchorMax = endAnchorMax;
            if(doAnchors) targetTransform.anchorMin = endAnchorMin;
            if(doSizeDelta) targetTransform.sizeDelta = endSizeDelta;
            if(doRotation) targetTransform.rotation = endRotation;
        }

        private void OnEnable()
        {
            if(resetOnEnable) SetStartTransform();
        }
    }
}
```

```
using UnityEngine;
using UnityEngine.Events;

namespace Scenes.UI.Scripts
{
    public abstract class UITweenerBase : MonoBehaviour
    {
        public abstract void DoForwardAnimation();
```

```
        public abstract void DoBackwardAnimation();

        public abstract float Duration { get; }

        public UnityEvent onAnimationEnd;

        public abstract void ResetToStart();
    }
}


using System;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

namespace Scenes.UI.Scripts
{
    public class UiTweenerLoop : MonoBehaviour
    {
        [SerializeField]
        private UITweenerBase tweener;
        [SerializeField]
        private float loopTime;
        [SerializeField]
        private bool loop;
        [SerializeField]
        private bool playOnStart;
        [SerializeField]
        private bool pingPong;

        private void Start()
        {
            if(playOnStart) Play();
        }

        public void Play()
        {
            StartCoroutine(PlayCoroutine());
        }


        private IEnumerator PlayCoroutine()
        {
            do
            {
                tweener.DoForwardAnimation();
                yield return new WaitForSecondsRealtime(loopTime);
                if (pingPong)
                {
                    tweener.DoBackwardAnimation();
                    yield return new WaitForSecondsRealtime(loopTime);
                }
                tweener.ResetToStart();
            } while (loop);
        }
    }
}


#if UNITY_EDITOR
using System.Collections.Generic;
using System.Linq;
using UnityEditor;
using UnityEngine;
using Scenes.UI.Scripts;

[CustomEditor(typeof(UIComboTweener))]
[CanEditMultipleObjects]
public class UIComboTweenerEditor : UnityEditor.Editor
{
    public override void OnInspectorGUI()
    {
        // Draw the default inspector first so the list is still visible/editable
        DrawDefaultInspector();

        EditorGUILayout.Space();
        EditorGUILayout.LabelField("Tools", EditorStyles.boldLabel);

        if (GUILayout.Button("Collect Tweens In Children"))
        {
            CollectTweensInChildrenForTargets(targets);
        }
    }

    private void CollectTweensInChildrenForTargets(Object[] targetObjects)
    {
        foreach (var obj in targetObjects)
        {
            var combo = obj as UIComboTweener;
            if (combo == null) continue;

            var so = new SerializedObject(combo);
            var tweensProp = so.FindProperty("tweens");
            if (tweensProp == null)
            {
                Debug.LogWarning("UIComboTweenerEditor: Could not find
serialized property 'tweens'.");
                continue;
            }

            // Get all UITweenerBase in children (including inactive), exclude self
            var found = combo.GetComponentsInChildren<UITweenerBase>(true)
                .Where(t => t != combo)
                .Distinct()
                .ToList();

            // Prepare for undo/overrides
            Undo.RecordObject(combo, "Collect Tweens In Children");

            // Assign to the serialized list
            so.Update();
            tweensProp.ClearArray();
            tweensProp.arraySize = found.Count;
            for (int i = 0; i < found.Count; i++)
            {
                var element = tweensProp.GetArrayElementAtIndex(i);
                element.objectReferenceValue = found[i];
            }
            so.ApplyModifiedProperties();

            // Mark dirty so changes persist in scenes/prefabs
            EditorUtility.SetDirty(combo);
        }
    }
}
#endif
```

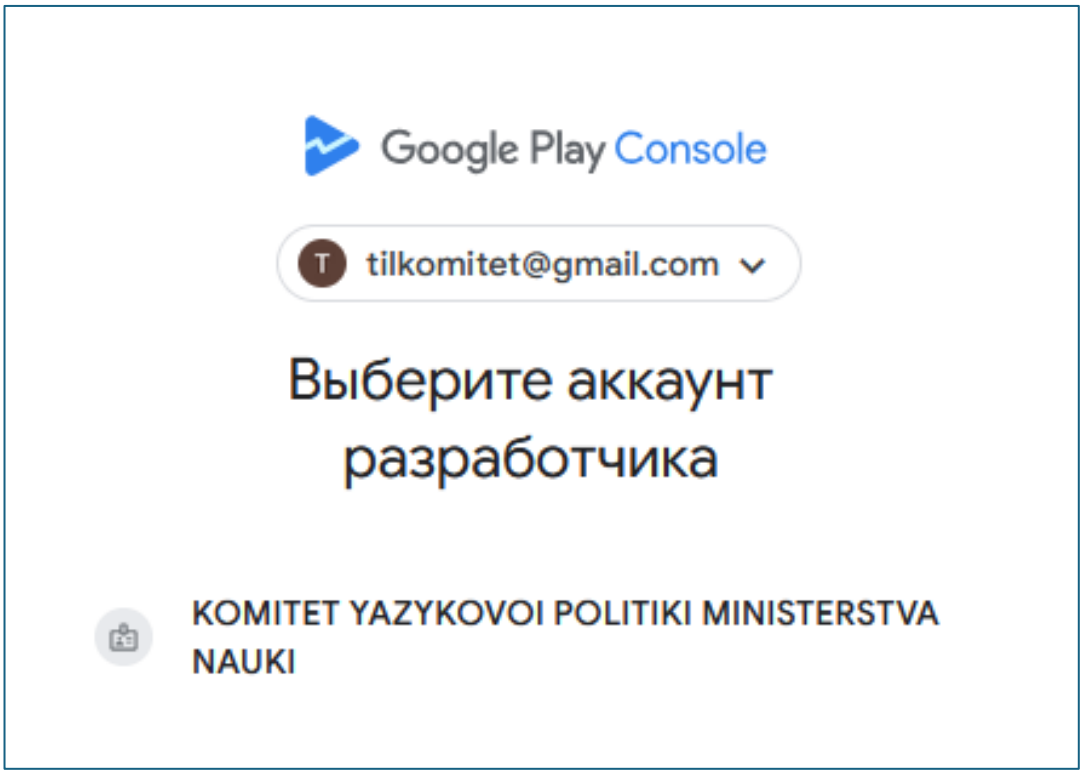
Қосымша-2

Ойын жарияланған платформалардың логиндері мен құпия сөздері

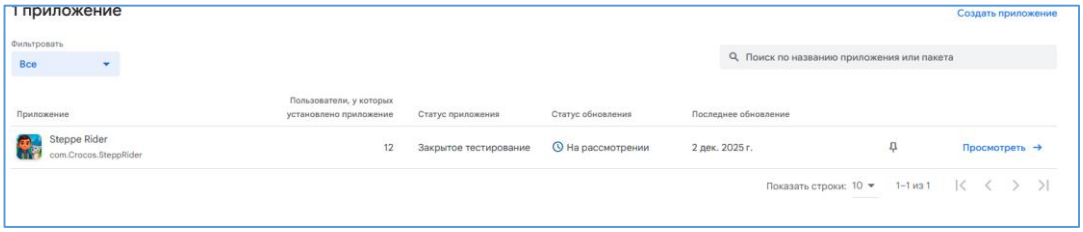
PLAY STORE

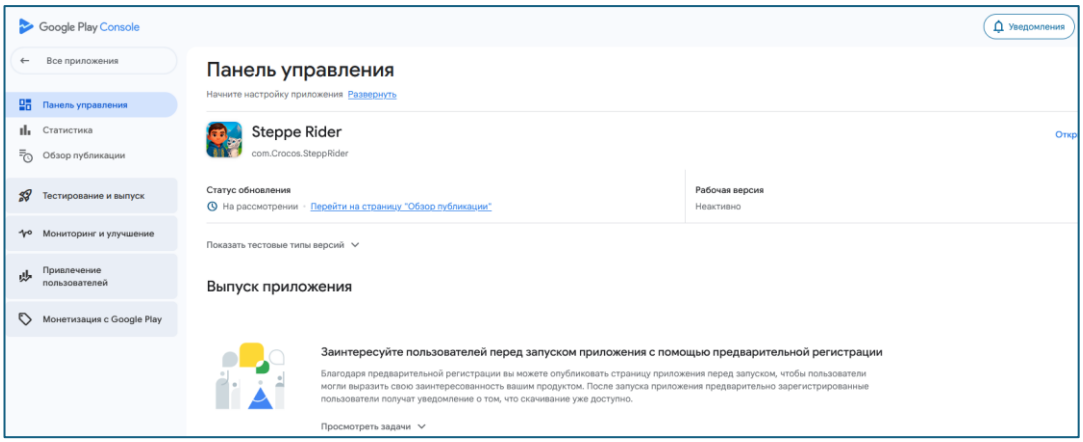
Логин: tilkomitet@gmail.com
Пароль: Astana2025@

1) Перейти на сайт https://play.google.com/console
2) Зайти в аккаунт
3) Выбрать аккаунт разработчика “KOMITET YAZUKOVOI POLITIKI…”



4) Выбрать приложение Steppe Rider

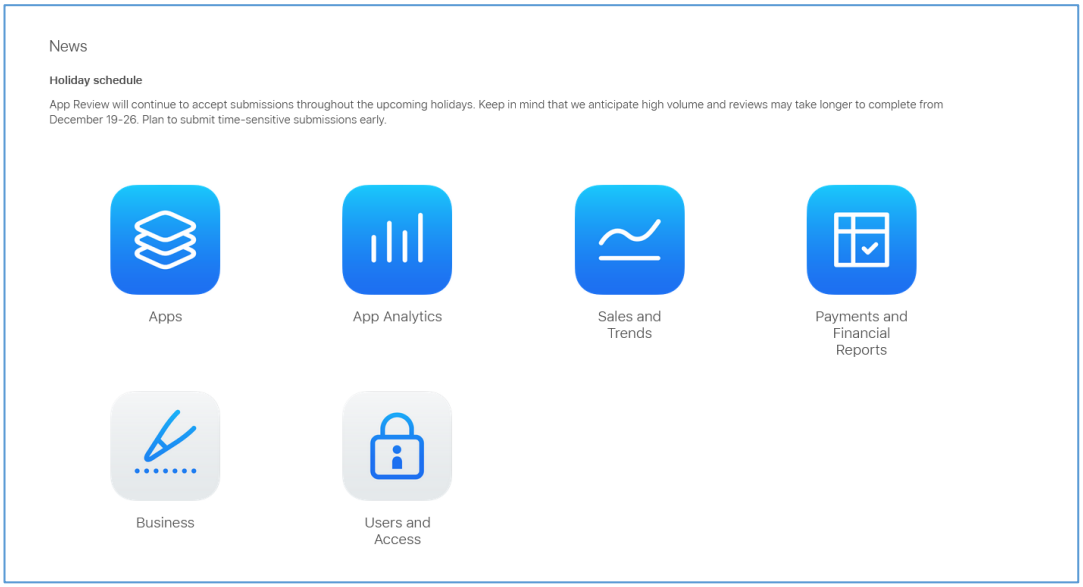




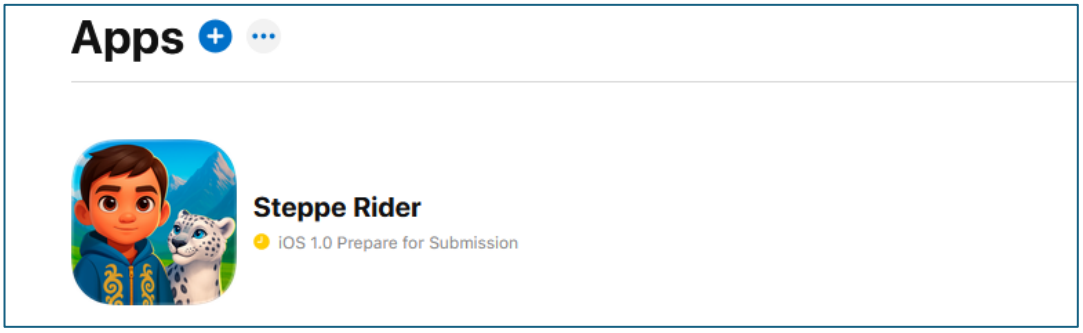
APP STORE

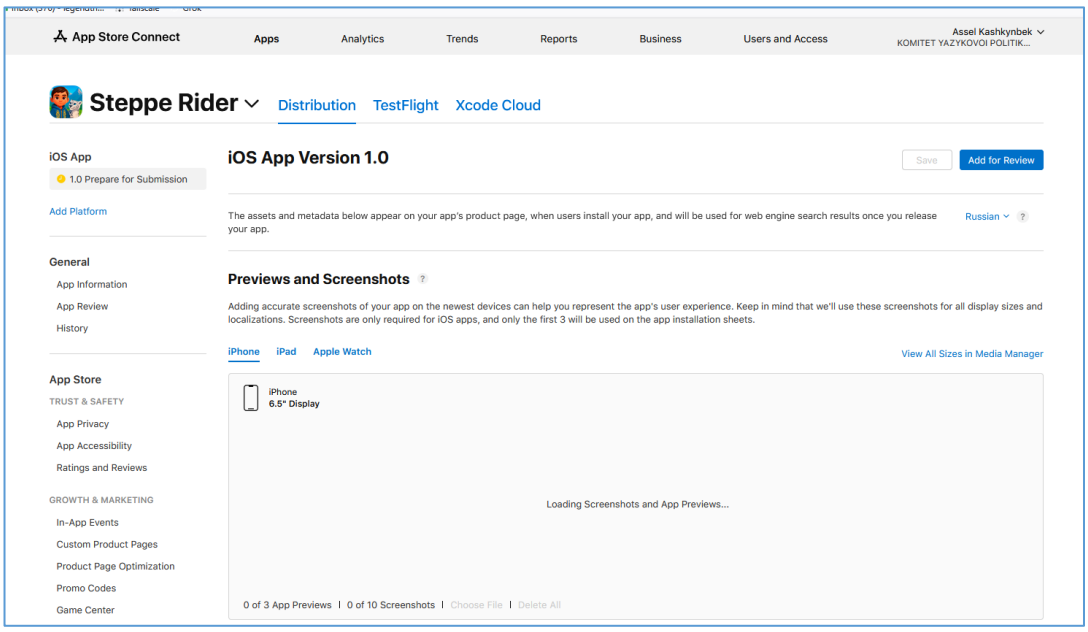
Логин: tilkomitet@gmail.com
Пароль: Astana2025@

1) Перейти на сайт https://appstoreconnect.apple.com/login
2) Ввести логин, пароль
3) Зайти в раздел “Apps”



4) Выбрать приложение Steppe Rider

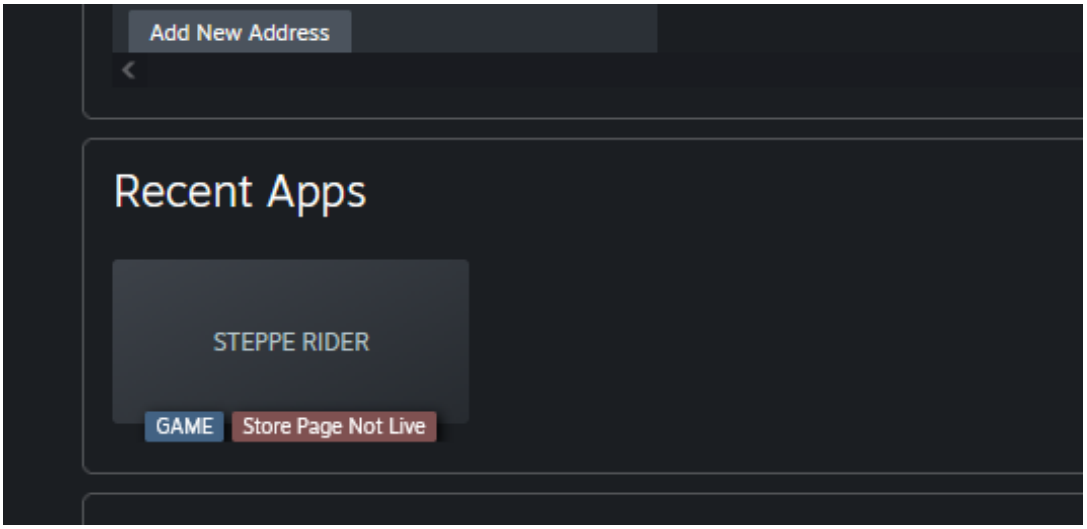


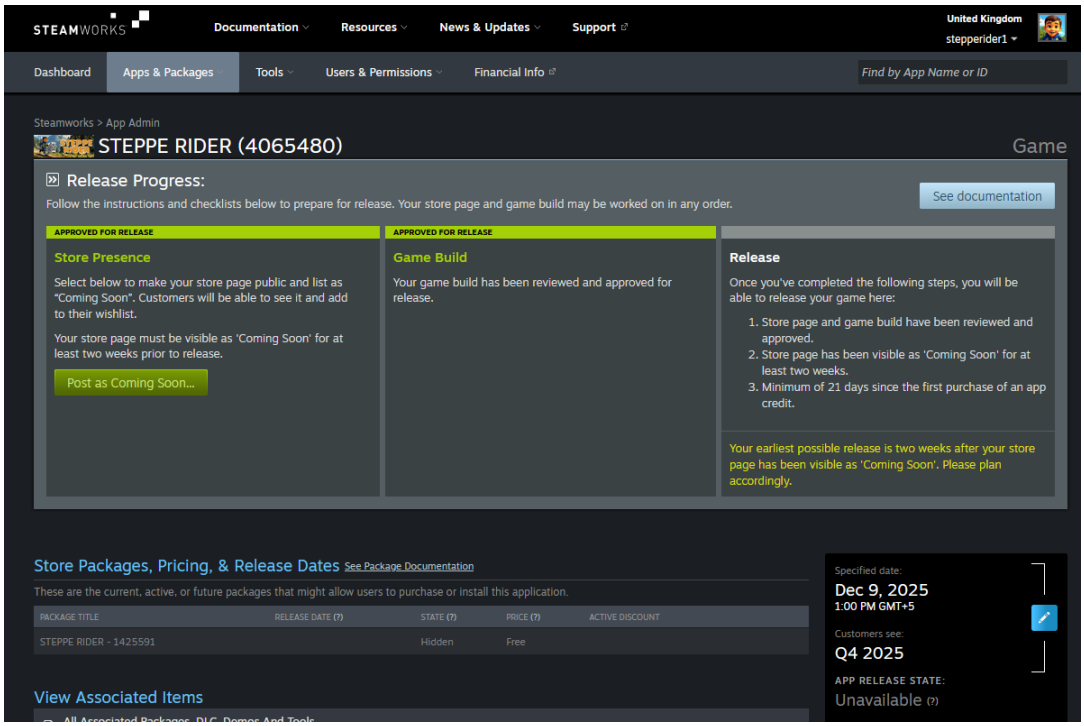


STEAM

Логин: stepperider1
Пароль: Stepperider2025!

1) Зайти на сайт https://partner.steamgames.com
2) Ввести логин/пароль
3) Выбрать приложение Stepper Rider





Қосымша-3

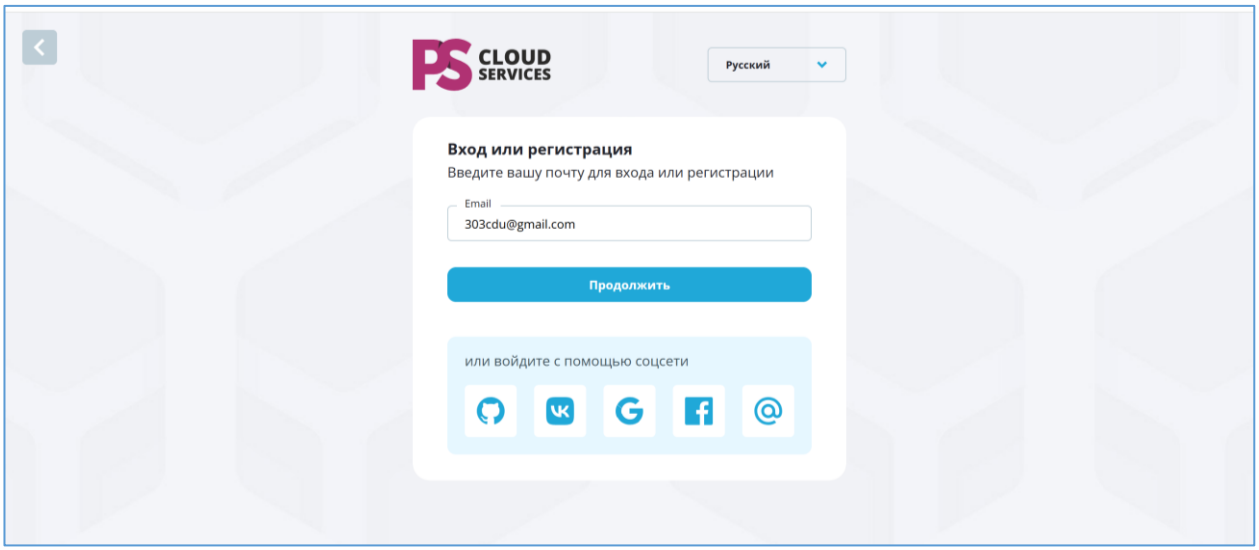
Серверлік бөлігінің конфигурациясы

Жоба PS.KZ платформасының серверінде орналастырылған. Төменде сервер сілтемесі, логин мен құпиясөздер және конфигурациясы келтірілген:

https://auth.ps.kz/?redirect_uri=https%3A%2F%2Fwww.ps.kz%2F%3F_gl%3D1*khk8k0*_ga*MTQ2NzExNzE1MS4xNzYwNjg4MTYy*_ga_GXGDY64BNM*czE3NjQ4NDQ3NzckbzE0JGcxJHQxNzY0ODQ0NzgxJGo1NiRsMCRoMA..

Келесі парақша ашылады:

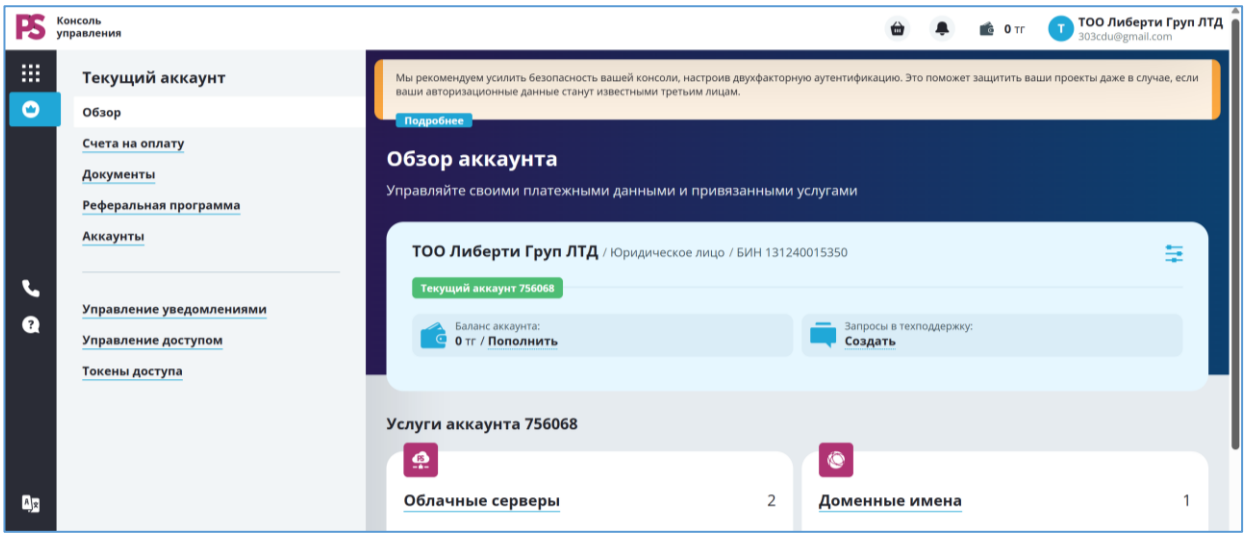
Логин: 303cdu@gmail.com
Құпиясөз: 123456!Qq

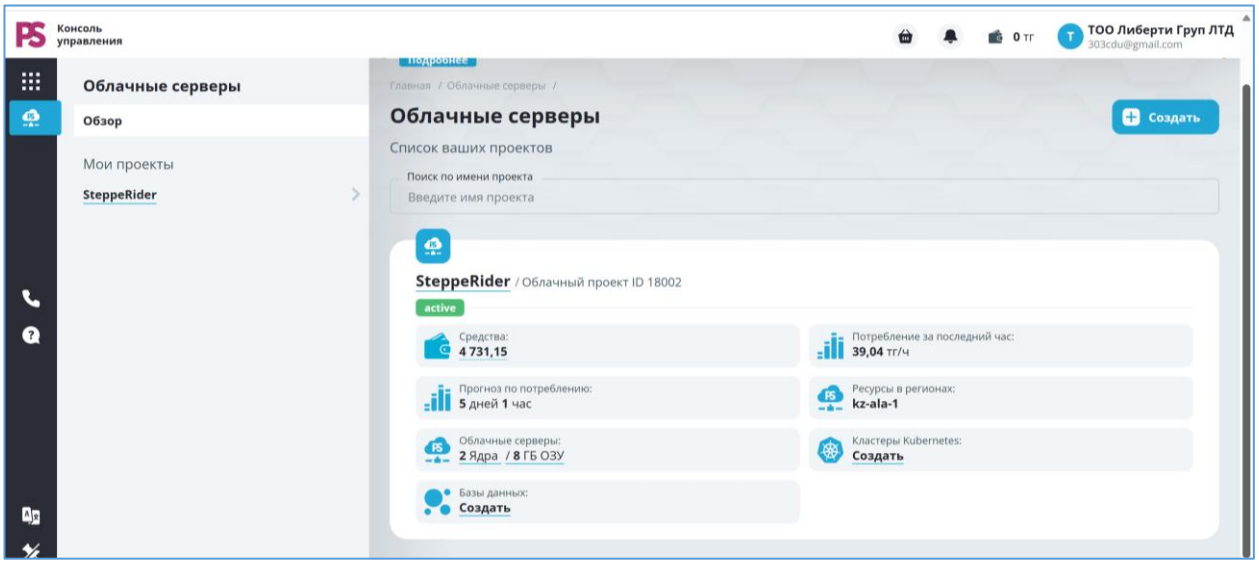
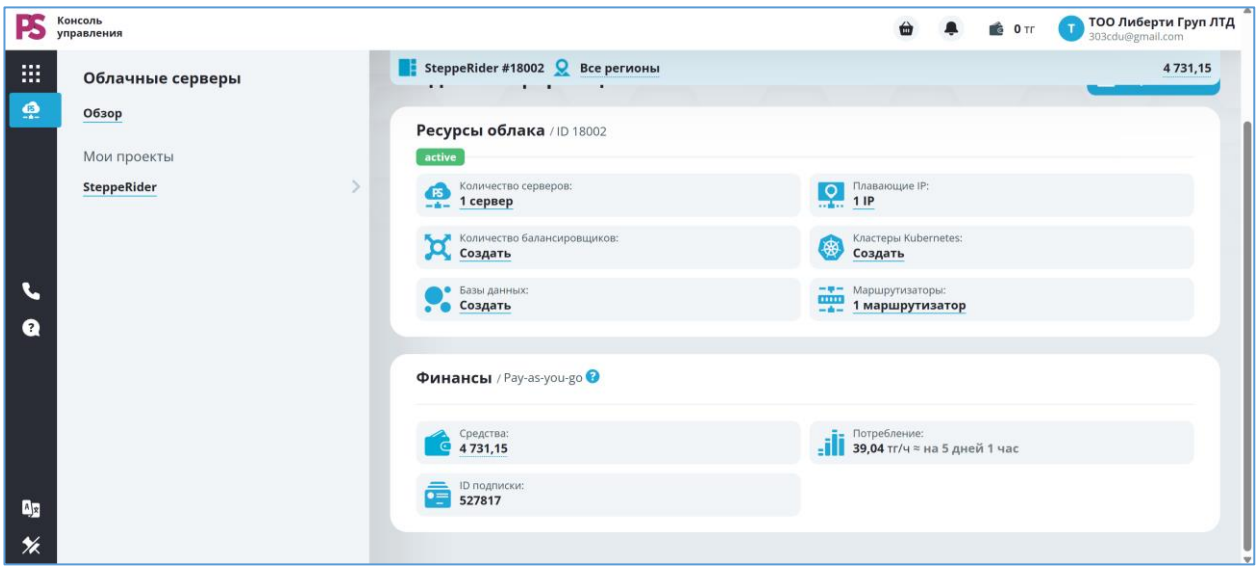


Келесі сілтемеге өту қажет:

https://console.ps.kz/account?_gl=1*1dt6gu6*_ga*MTQ2NzExNzE1MS4xNzYwNjg4MTYy*_ga_GXGDY64BNM*czE3NjQ4NDQ3NzckbzE0JGcxJHQxNzY0ODQ1MTEyJGo1MyRsMCRoMA..

Бұл сервердің конфигурация бөлімі:



Ашылған парақшада «Облачные серверы» ұяшығын басып, ойынның серверлік бөлігіне өтуге болады:

https://console.ps.kz/cloud