МІЖНАРОДНИЙ ГУМАНІТАРНИЙ УНІВЕРСИТЕТ

Факультет кібербезпеки, програмної інженерії та комп'ютерних наук
Кафедра інформаційних технологій

Пояснювальна записка

до кваліфікаційної роботи
другого (магістерського) рівня

на тему **РОЗРОБКА ІГРОВОГО ANDROID-ДОДАТКУ У ЖАНРІ ПОКРОКОВОЇ СТРАТЕГІЇ НА БАЗІ UNITY/C#**

Виконав: студент 2 курсу, групи ІПЗ
спеціальності
121 Інженерія програмного забезпечення

_____ Кудрицький А. О.

Керівник _____ Стрелковська І.В.

Рецензент _____ О. П. Русу

Одеса – 2023 р.

Д О В І Д К А

кафедри ІТ про виконану магістерську роботу

студента 2 курсу ФКПІ та КН групи ІПЗ

Кудрицького Андрія Олеговича

на тему Розробка ігрового Android-додатку у жанрі покрокової стратегії на базі Unity/C#

Висновок нормоконтролера пояснювальна записка до кваліфікаційної роботи викон. з незнач. поруш. ДСТУ, виконано з вимогами внутрішнього положення НУ

Нормоконтролер викл. каф. ІТ 15.12.2023 Клімішина І.В.
(науковий ступінь, вчене звання, посада) (підпис, дата) (і. б. прізвище)

Висновок відповідального за наявність академічного плагіату згідно з сертифікатом ID 10000 14 33 унікальність роботи підтверджено

Відповідальна особа викл. каф. ІТ 15.12.2023 Клімішина І.В.
(науковий ступінь, вчене звання, посада) (підпис, дата) (і. б. прізвище)

Попередня експертиза (захист) магістерської роботи
(бакалаврської роботи чи магістерської роботи)

студ. Кудрицького А.О. проведена " 13 " 12 20 23 р.
(прізвище і.б.)

Висновки Виконана робота відповідає стандарту. В роботі розроблено ігровий Android-додаток у жанрі покрокової гри стратегії, базуючись на базі Unity за допомогою мови програмування C#. Магістр має достатню теоретичну підготовку, добре володіє мовою програмування C#, яку він застосовував. Ціль роботи наведено і сформульовано. Виконана робота магістра відповідає вимогам стандартам та рекомендується до захисту.

Члени комісії д.т.н. проф. Стрельковська І.В.
(підпис) (науковий ступінь, вчене звання, посада, прізвище і.б.)

к.т.н. доц. Григор'єва Т.Л.
(підпис) (науковий ступінь, вчене звання, посада, прізвище і.б.)

к.т.н. доц. Горбачов В.Є.
(підпис) (науковий ступінь, вчене звання, посада, прізвище і.б.)

МІЖНАРОДНИЙ ГУМАНІТАРНИЙ УНІВЕРСИТЕТ

Факультет кібербезпеки, програмної інженерії та комп'ютерних наук
Кафедра інформаційних технологій
Освітній ступінь магістр
Галузь знань 12 Інформаційні технології
Спеціальність 121 Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри ІТ
к.т.н., доц.
Т.І. Григор'єва
"25" 09 2023 року

З А В Д А Н Н Я
НА МАГІСТЕРСЬКУ РОБОТУ

Кудрицькому Андрію Олеговичу

1. Тема роботи Розробка ігрового Android-додатку у жанрі покрокової стратегії на базі Unity/C#
керівник роботи . Стрелковська І. В., д.т.н., професор кафедри ІТ .
затверджені наказом закладу вищої освіти від від 25.09.2023 р. № 1957 зі змінами від 04.12.2023 р. № 3102

2. Строк подання студентом роботи 11.12.2023 р.

3. Вхідні дані до роботи .
. Технічна документація Unity;
. Пристрої для тестування продукту;
. Інформація про конкурентні проекти;
. Опис алгоритму покрокової стратегії.

4. Зміст розрахунково-пояснювальної записки
. Розділ 1 – Аналіз процесу розробки ігрового Android-додатку і актуальності предметної області;
. Розділ 2 – Формулювання функціоналу гри та огляд використаних технологій;
. Розділ 3 – Проектування процесу розробки і архітектури компонентів гри;
. Розділ 4 – Розробка компонентів гри;
. Розділ 5 – Технічне і функціональне тестування продукту.

5. Перелік графічного матеріалу (з зазначенням обов'язкових креслень)
Слайд 1 – Актуальність проекту і огляд інструментів.
Слайд 2 – Основні характеристики роботи.
Слайд 3 – Вибір цільової платформи.
Слайд 4 – Характеристики мобільних платформ
Слайд 5 – Порівняння головних характеристик найбільш популярних «движків»
.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання _____

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів магістерської роботи	Строк виконання етапів роботи	Примітка
1	Підготовка першого розділу	25.09.2023-01.10.2023	
2	Підготовка другого розділу	02.10.2023-08.10.2023	
3	Підготовка третього розділу	09.10.2023-15.10.2023	
4	Підготовка четвертого розділу	16.10.2023-05.11.2023	
5	Підготовка п'ятого розділу	06.11.2023-12.11.2023	
6	Підготовка реферату, вступу, висновків	13.11.2023-19.11.2023	
7	Підготовка слайдів презентації	20.11.2023-26.11.2023	
8	Перелік джерел посилання	27.11.2023-03.12.2023	
9	Додаток А Перелік копій демонстраційного матеріалу	04.12.2023-10.12.2023	

Студент _____ А.О. Кудрицький
(підпис)

Керівник роботи _____ І.В. Стрелковська
(підпис)

ВІДГУК

наукового керівника на кваліфікаційну

магістерську роботу

на тему «РОЗРОБКА ІГРОВОГО ANDROID-ДОДАТКУ У ЖАНРІ ПОКРОКОВОЇ СТРАТЕГІЇ НА БАЗІ UNITY/C#»

студента Кудрицького А.О.

Спеціальність: 121 Інженерія програмного забезпечення

Сучасні мобільні пристрої обладнані потужними технічними характеристиками, що дозволяє створювати високоякісні ігри. Гра у жанрі стратегії ідеально підходить для мобільних платформ, дозволяючи гравцям повертатися до гри в будь-якому місці та часі. Тема магістерської роботи є актуальною, оскільки розробка у жанрі стратегії, зокрема на основі Unity/C#, створює зручні умови для реалізації різноманітних стратегій монетизації. Це може включати в себе внутрішні покупки, рекламу, підписки тощо. Такий підхід відкриває перед розробниками можливість створення успішних та прибуткових проектів.

В роботі розроблено ігровий Android-додаток у жанрі покрокової стратегії, базуючись на потужному движку Unity.

Результати дослідження представлені у тезах доповідей IX Всеукраїнської науково-практичної конференції студентів, аспірантів та молодих учених «Гуманітарний і інноваційний ракурс професійної майстерності: пошуки молодих вчених».

Магістерська кваліфікаційна робота виконана у відповідності з завданням із дотриманням всіх вимог до кваліфікаційних робіт магістрів і заслуговує на оцінку "добре". Студент Кудрицький А.О. заслуговує присвоєння кваліфікації магістр з інженерії програмного забезпечення за заявленою спеціальністю 121 «Інженерія програмного забезпечення».

Науковий керівник:

доктор технічних наук, професор  І.В.Стрелковська

РЕЦЕНЗІЯ

на кваліфікаційну магістерську роботу
на тему «РОЗРОБКА ІГРОВОГО ANDROID-ДОДАТКУ У ЖАНРІ ПОКРОКОВОЇ СТРАТЕГІЇ НА БАЗІ UNITY/C#»

студента Кудрицького А.О.

Спеціальність: 121 Інженерія програмного забезпечення

Покрокові стратегії завжди користуються стабільною популярністю серед гравців, особливо тих, хто цінує стратегічне мислення, планування та тактичні рішення. Ключовими елементами для тривалого успіху гри є регулярні оновлення та активна спільнота гравців, що говорить про безумовну актуальність теми магістерської кваліфікаційної роботи. Використання платформи Unity та мови програмування C# відкриває безліч можливостей для легкого впровадження оновлень та взаємодії з гравцями через різноманітні канали.

Магістерська кваліфікаційна робота виконана у відповідності з завданням із дотриманням всіх вимог до кваліфікаційних робіт магістрів і заслуговує на оцінку "добре". Студент Кудрицький А.О. заслуговує присвоєння кваліфікації магістр з інженерії програмного забезпечення за заявленою спеціальністю 121 «Інженерія програмного забезпечення».

Рецензент,
доцент кафедри
комп'ютерних наук,
кандидат технічних наук О.П.Русу

UNICHECK
by Turnitin

Ім'я користувача:
Анна Серединко

Дата перевірки:
11.12.2023 23:54:31 MSK

Дата звіту:
12.12.2023 00:06:10 MSK

ID перевірки:
1015995239

Тип перевірки:
Doc vs Internet + Library

ID користувача:
100001433

Назва документа: Кудрицький_ диплом

Кількість сторінок: 95 Кількість слів: 9887 Кількість символів: 82190 Розмір файлу: 6.54 MB ID файлу: 1015677901

Виявлено модифікації тексту (можуть впливати на відсоток схожості)

16.9%
Схожість

Найбільша схожість: 4.93% з Інтернет-джерелом (http://repository.rshu.edu.ua/id/eprint/11126/1/%D0%9A%D1%83%D0..

15.9% Джерела з Інтернету | 925 ..... Сторінка 97

4.13% Джерела з Бібліотеки | 53 ..... Сторінка 101

0% Цитат

Вилучення цитат вимкнене

Вилучення списку бібліографічних посилань вимкнене

0%
Вилучень

Немає вилучених джерел

Модифікації

Виявлено модифікації тексту. Детальна інформація доступна в онлайн-звіті.

Підозріле форматування | 21 сторінка

РЕФЕРАТ

Пояснювальна записка до магістерської роботи: 59 с., 52 рис., 12 табл., 11 джерел, 2 додатка.

У даній роботі розглянуто аспекти процесу розробки гри інді-компанією, проблеми, які виникають при виборі інструментів та інші пов'язані питання.

Метою роботи є поліпшення навичок необхідних для створення програмного забезпечення, наприклад:

- проектування процесів;
- проектування архітектури програмного забезпечення;
- володіння мовами програмування;
- якісного тестування продукту.

Для цього було поставлено задачу розробити покрокову стратегію з нуля, що дало можливість зіткнутися із актуальними проблемами створення ігор і намагатися їх вирішити.

У результаті роботи було проведено аналіз великої кількості ресурсів, що стало основою для початку розробки ігрового додатку. Були розглянуті доступні платформи, операційні системи та їх технічні обмеження. Було отримано продукт, що задовольняє більшість критеріїв оцінювання ПЗ, таких як:

- повнота реалізованого функціоналу;
- оптимізація додатку для отримання максимальної продуктивності.

Отриманий додаток гри можна використовувати і розвивати у подальшому, для досягнення повноцінного релізу.

РОЗРОБКА ГРИ, ANDROID-ДОДАТОК, ПОКРОКОВА СТРАТЕГІЯ, ПЛАНУВАННЯ ДОДАТКУ, ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ, ІНДІ-РОЗРОБКА, ВІДЕОІГРИ.

ВСТУП

У сучасному світі створення відеоігор є одним із найбільших сегментів індустрії розваг. Масштаб ігрової індустрії порівнянний, наприклад, з кіноіндустрією. Що стосується темпів зростання за останні п’ять років, індустрія відеоігор значно випередила її.

За ступенем впливу на споживачів та їх залученість до інтерактивного середовища, пропонованого відеоіграми, цей сегмент вже давно відрізняється від інших видів розваг.

Сьогодні більшість розроблюваних продуктів, на жаль, це лише бізнес-плани, ціль яких помножити прибутки компанії. Такі додатки зазвичай не спираються на відгуки аудиторії і не захоплюють гравців.

Інді-розробка є одним із способів створення ігор разом із майбутніми гравцями, постійно отримуючи зауваження та коментарі, що в теорії допоможе додатку стати краще.

Але незалежні проекти-додатків ігор зазвичай дуже обмежені у ресурсах, тож необхідне правильні планування і підхід. Саме проблеми, з якими стикаються новачки і способи їх вирішення буде розглянуто у даній роботі. Також по результатам роботи можна буде оцінити можливість створити гідний продукт не маючи великої команди.

1 АНАЛІЗ ПРОЦЕСУ РОЗРОБКИ ІГРОВОГО ANDROID-ДОДАТКУ І АКТУАЛЬНОСТІ ПРЕДМЕТНОЇ ОБЛАСТІ

На даний момент комп’ютерні та мобільні ігри є одним з найпопулярніших видів розваг серед людей молодшого та середнього віку. А тих, хто на будь-якому рівні залишився не знайомим із поняттям відеоігор – майже немає. У даному розділі ми розглянемо як і чому цей вид відпочинку почав захоплювати наш час, переріс у сучасне мистецтво, та надає великі перспективи тим, хто вирішив зайнятися розробкою. Буде розглянуто аргументи по вибору тих чи інших інструментів, що були використані у додатку та їх стисле порівняння.

1.1 Історія розвитку відеоігор

Історія розвитку відеоігор налічує вже понад 60 років. Перші ігри були примітивними, проте саме вони задали вектор розвитку всієї ігрової індустрії. Еволюція відеоігор безпосередньо пов'язана з прогресом в сфері «ігрового заліза». Причому в початковій стадії це стосувалося, в першу чергу, ігрових приставок, тому що ПК стали доступні більшості користувачів набагато пізніше.

Створенню перших відеоігор передували деякі напрацювання і винаходи, завдяки яким і з'явилися на світ «прабатьки» сучасних ігор. Так само як патент на використання ЕПТ (електронно-променевої трубки) в ігрових цілях (1947 рік), створення алгоритму шахової гри для комп'ютера (1948 рік), а потім написання першої подібної програми під назвою «TUROCHAMP» (комп'ютерів, здатних її запустити тоді ще не було) в 1950-1951 роках.

Офіційно першою відеогрою вважається «OXO» - знамениті «хрестики-нулики», які з паперу були перенесені на екран комп'ютера А.С. Дугласом в 1952 році. Фактично ж першої відеогрою стала симуляція запуску ворожих ракет, яка була створена в 1947 році на базі військового обладнання (гра запускалася на моніторі радара).

У 1958 році вченим Вільямом Хігінботемом був створений перший симулятор тенісу під назвою «Tennis for Two». Грати в нього могли двоє людей: вони управляли рухливими платформами, якими відбивали м'ячик.

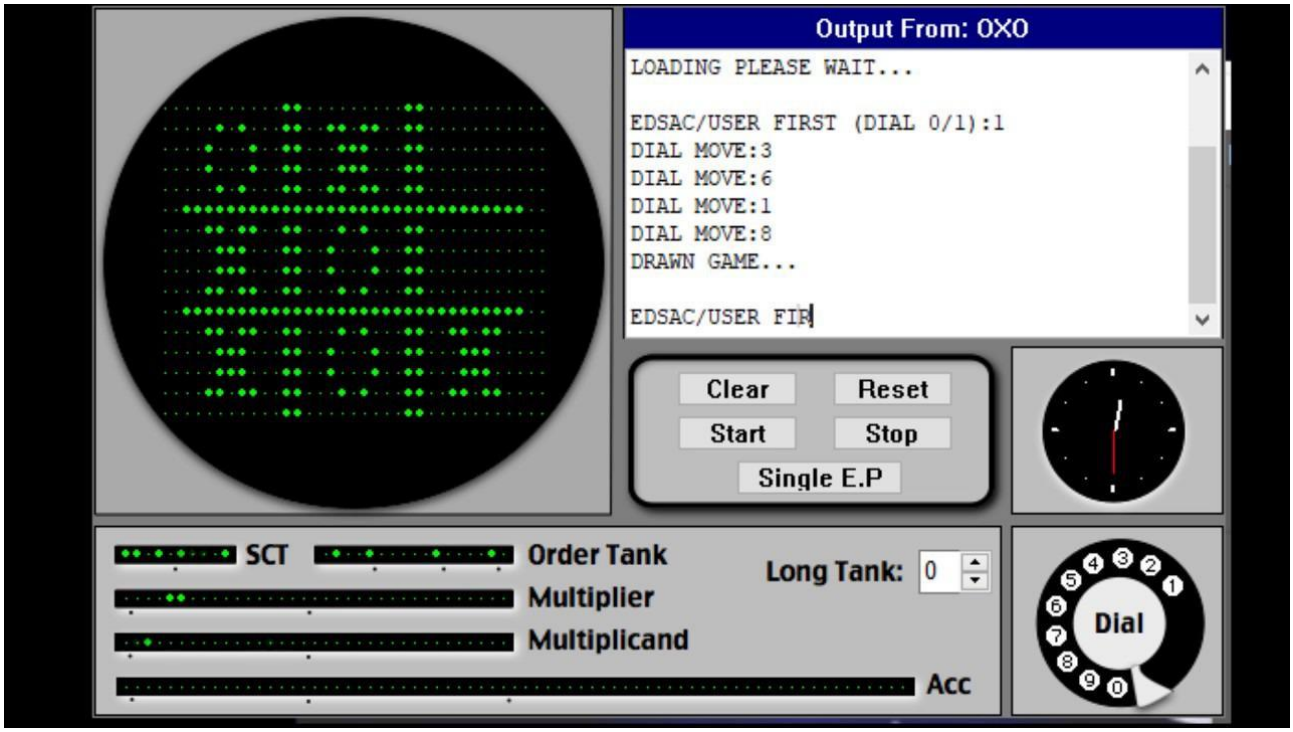


Рисунок 1.1 – Інтерфейс гри «OXO»

У 1962 році Стівом Расселом і групою його студентів з MIT (Массачусетський технологічний інститут) була створена «SpaceWar» - перша самостійна комп'ютерна гра (попередні були своєрідним «портом» існуючих настільних ігор).

Рисунок 1.2 – Ігровий процес «SpaceWar»

Сервіс цифрової дистрибуції ігор під назвою «Steam» вирішив віддати данину одному з перших представників індустрії і використовує космічну аркаду для тестування різних функцій сервісу на кшталт системи досягнень або Майстерні.

У 1976 році світ побачив «Breakout» - перший в світі арканоид, а також «Death Race» - перша гра, яка була заборонена через насильства і жорстокості, які були використані в її ігровому процесі.

У 1978 році була створена легендарна гра «Space Invaders», яка настільки сподобалася користувачам, що ігрові автомати з нею встановлювалися в США і Японії на кожному розі. Про ігри почали писати в газетах, знімати фільми - ця індустрія стрімко розкручувала маховик своєї популярності.

У 1980 році з'явився новий ігровий жанр Rogue-like (за назвою оригінальної гри Rogue), де гравцеві потрібно було переміщатися в підземеллях і битися з монстрами. 22 травня цього ж року в продажу з'являються ігрові автомати з «PacMan» на борту. Гра отримує схвальні відгуки і стає символом аркадного жанру і відеоігор, в цілому, навіть на сьогоднішній день.

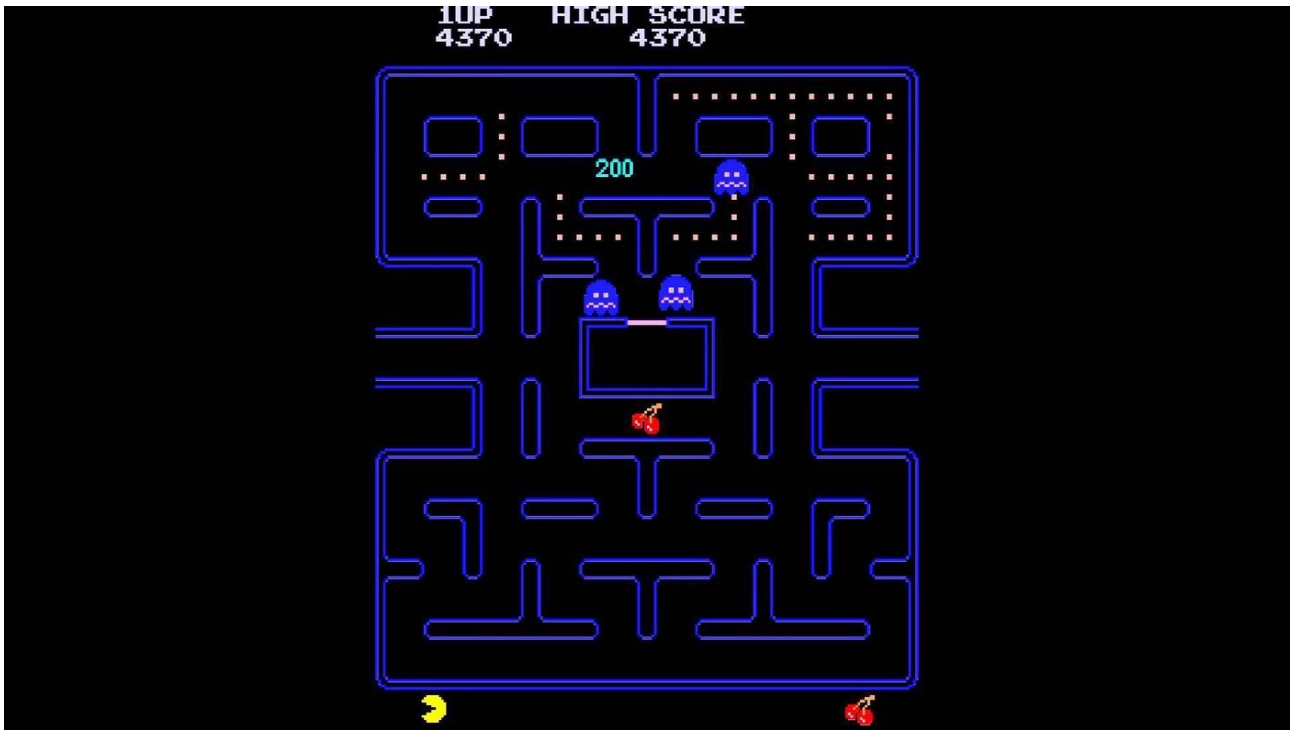


Рисунок 1.3 – Ігровий процес «Pacman»

У 1987 році комп'ютерні відеокарти починають підтримувати новий стандарт VGA (256 кольорів), що зробило відеоігри барвистими і все більш схожими на

сучасні. В цей же час була створена перша комп'ютерна звукова карта «AdLib», що є ще одним якісним кроком на шляху еволюції ігор.

1988 рік - початок продажів Sega Mega Drive - ігрової консолі 4-го покоління, що підтримує 16-ти бітну графіку і звук, що відповідало досягненням останніх ПК того часу.

Черговий значною віхою еволюції відеоігор став тривимірний шутер з видом від першої особи «Wolfenstein 3D», що вийшов в 1992 році. З'явився еталон файтингів «Mortal Kombat», а також перша стратегія в реальному часі (RTS) «Dune 2» і перша хоррор-гра « Alone in the Dark ».

Рисунок 1.4 – Ігровий процес «Wolfenstain 3D»

У 1994 році з'являються консолі п'ятого покоління, кращою з яких стала Sony PlayStation. В цей же час Blizzard створює свій перший шедевр «Warcraft: Orcs and Humans» - RTS, що стала одним з лідерів жанру.

1995 рік теж був багатий на новинки: з'явилася популярна покрокова стратегія «Heroes of Might and Magic», гоночна серія «Need for Speed», на ПК вийшов легендарний квест «Myst».

У грі «Highlander: The Last of the MacLeods» була вперше застосована технологія захоплення руху, що використовувалась раніше тільки в фільмах.

1996 року була випущена Voodoo I - перша відеокарта з підтримкою 3D-прискорення, що дало потужний поштовх до розвитку тривимірних ігор, таких як «Duke Nukem 3D», «Command & Conquer: Red Alert», «Tomb Raider», «Resident Evil», « Diablo ».

Рисунок 1.5 – Ігровий процес «Warcraft: Orcs and Humans»

Рисунок 1.6 – Сучасна технологія захоплення руху

2002 рік. Виходить третя частина культової серії РПГ під назвою «The Elder Scrolls III: Morrowind», що задає нові стандарти рольових ігор: величезний відкритий світ, опрацьована система прокачування персонажа, цікава сюжетна лінія. Починає набирати популярність «Dota» - гра, що представляє модифікацію однієї з карт «Warcraft III».

2015 рік. Була випущена «The Witcher 3: Wild Hunt» - найкраща РПГ за мотивами слов'янського фентезі. Отримала високі оцінки критиків і народне визнання.

2016 рік. Всього лише пару років тому дуже активно почала розвиватися ігрова індустрія віртуальної реальності, чому сприяв вихід на ринок таких VR-гарнітур як Oculus Rift, Sony Playstation VR, HTC Vive (2015 р). За цей час були випущені такі значущі VR-проекти як «The Elder Scrolls V: Skyrim VR», «Doom: VFR», «Fallout 4 VR». В цьому ж році світ був «захоплений» відомої AR-грою «Pokemon Go».

Рисунок 1.7 – Ігровий процес у VR-обладнанні

Замість висновку можна привести наступний факт: охоплення використовуваних технологій у ігрових додатках зростає кожен день, а бюджети рівняються і навіть перевищують кінематограф. Тому не залишається ніяких сумнівів, що за ігробудівництвом майбутнє розважальної індустрії.

1.2 Вибір цільової платформи

Одне з перших питань, що виникають у розробника – вибір цільової платформи для продукту. У великих компаній, зазвичай, ця проблема не стоїть дуже гостро, тому що вони вже мають готові продукти, а значить і досвід. А навіть якщо досвід відсутній – можна найняти його за відповідну частину бюджету.

Коли мова йде про інді-розробку головними перешкодами стають відсутність фінансування і дуже обмежений людський ресурс. Ці фактори спонукають до слідування наступному принципу: найменша кількість зусиль для отримання найкращого результату. Створити конкурентоспроможний додаток на персональний комп’ютер з кожним роком стає важче по причинам збільшення бюджетів і кількості технологій.

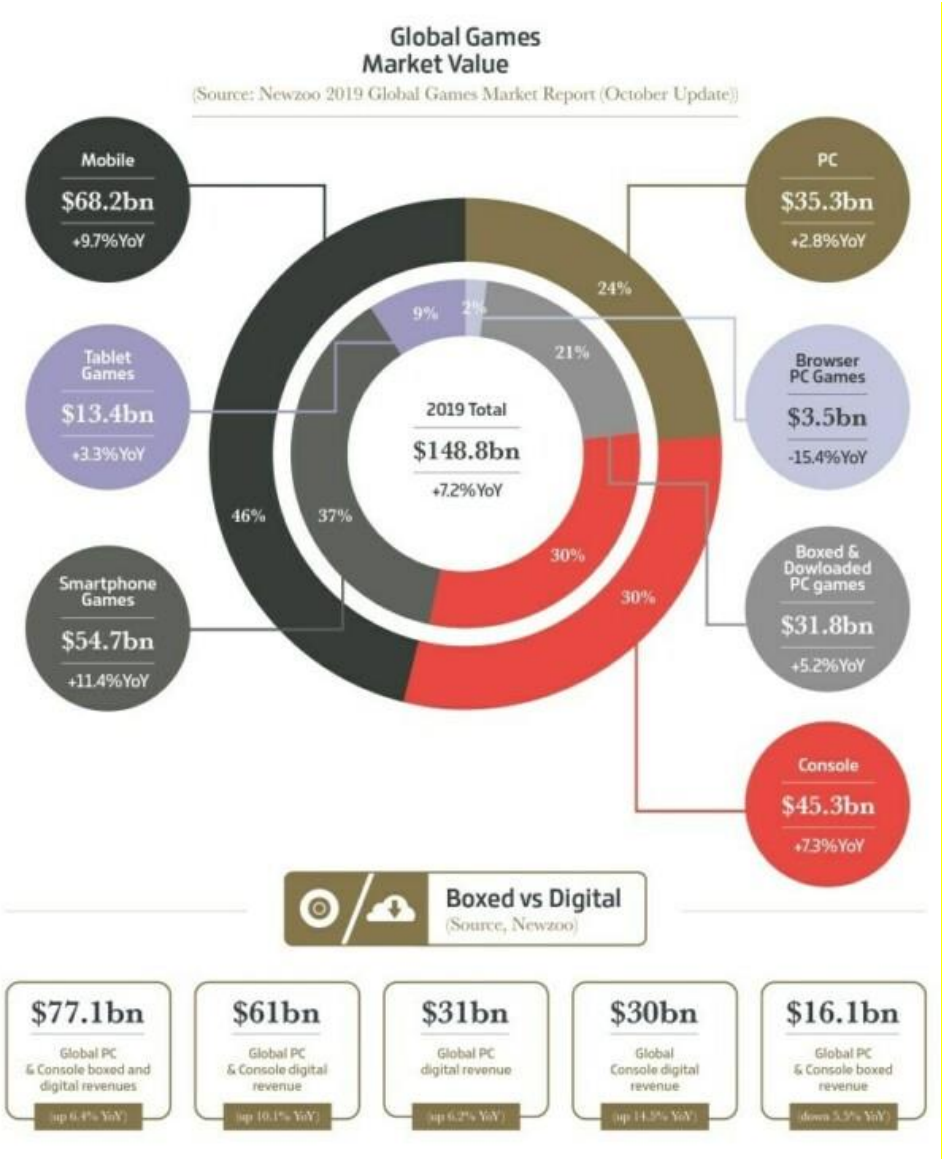


Рисунок 1.8 – Діаграма витрат на ринку відеоігор за 2023 рік

Наприклад, на розробку культової гри «Witcher 3» компанія CD Project RED витратила приблизно 67 мільйонів доларів США.

Зазвичай, ігри для мобільних платформ набагато менші у всіх аспектах: від витрат до самого продукту. Незважаючи на це, користувачі з охотою витрачають свої гроші у AppStore і Google Play, а загальна кількість грошового обороту на мобільних платформах вища за ПК, що видно на рис. 1.8. Це робить мобільні пристрої пріоритетною ціллю для інді-розробників.

Наступною проблемою стає вибір між операційними системами iOS і Android. Їх стислі характеристики з точки зору процесу розробки наведено у таблиці 1.1.

Таблиця 1.1 – Характеристики мобільних платформ

Назва	Технічні вимоги	Необхідність у платному ПЗ	Вимоги щодо публікації додатку
iOS	iPhone/iPad, iMac/MacBook	Відсутня за умови наявності аккаунта розробника	Аккаунт розробника, щорічний внесок у розмірі 100 USD
Android	Будь-який пристрій на Android, будь-який пристрій на Windows.	Відсутня	Аккаунт розробника, одноразовий внесок у розмірі 25 USD

Як видно, Android є більш гнучкою платформою для незалежних додатків, тому його було обрано відправною точкою. Але повністю відкидати iOS та AppStore не можна, так як, згідно зі статистикою на рис. 1.9, вони продовжують приносити більший за Google Play дохід.

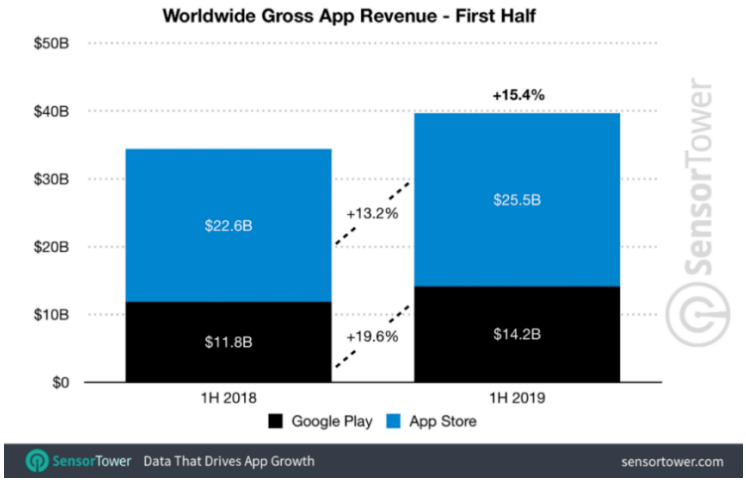


Рисунок 1.9 – Діаграма заробітків Google Play і App Store

1.3 Вибір середовища розробки

Сьогодні існує велика кількість так званих «ігрових движків», але всі вони мають основний напрямок і можуть бути як поганим, так і гарним вибором для того чи іншого додатку. Перш за все необхідно визначити яка мова програмування буде основною у додатку і чи необхідні інструменти для роботи з тяжкими 3д сценами. У моєму випадку ведеться розробка покрокової стратегії у 2д, а профільними мовами програмування є C++ та C#. Ці критерії і стали основними у процесі пошуку.

Таблиця 1.2 – Порівняння головних характеристик найбільш популярних «движків»

Назва	**Поріг входження**	**Ліцензія**	**Мова програмування**	**Рівень документованості**
Unity	низький	Free(until revenue < 100000 USD per year)	C#/Javascript	середній
CryEngine	високий	Free(non-commercial)	C++	низький
Unreal Engine	середній	Free(until revenue < 1000000 USD)	C++	середній

Беручи до уваги характеристики з таблиці 1.2 основними кандидатами стають Unity та Unreal Engine. Згадуючи принцип, яким користується більшість інді-розробників зупиняємося на Unity через низький поріг входження. Люди у ігровому додатку можуть змінюватися з великою швидкістю, що потребує загальної доступності інструментів для мінімізації часу адаптації.

1.4 Аналіз аналогів на ринку

Одним з головних аспектів будь-якого додатку, а тим паче розробки гри - є окупність. Якщо дохід з готового продукту не зможе перевищити або хоча б покрити кількість витрачених ресурсів – його вважають провальним. Зазвичай, монетизація гри, незалежно від типу, прямо залежить від кількості зацікавленої аудиторії. На початку дізнаємося, чи можливо розраховувати на окупність гри у жанрі покрокової стратегії. Для цього подивимось на успіхи конкурентів, а саме: кількість загрузок, ціну електронної копії тощо.

Усі далі розглянуті додатки розроблялися під Android і розповсюджуються через сервіс «Google Play». Першим продуктом для огляду стане гра «Braveland» інді-компанії «Tortuga Team». Розглянемо найцікавіші для нас її характеристики, що наведені у таблиці 1.3.

Таблиця 1.3 – Дані по «Braveland» з «Google Play»

Кількість загрузок	10 000+
Середня оцінка користувачів	4,2/5
Тип монетизації	Продаж електронних копій
Ціна за одну копію (на момент написання роботи)	70 UAH

Як видно, кількість користувачів платної гри відносно невелика. Одною з причин є те, що психологічно більшість людей не готові до витрат на ліцензійний «софт» під Android, а також не мають легкої змоги випробувати продукт перед покупкою.

Порахуємо приблизний дохід компанії, для чого перемножимо кількість загрузок і ціну за копію, віднявши 30%, які стягує «Google Play». Отримуємо наступну формулу:

$$\text{Дохід} = (\text{Кількість загрузок} * \text{ціна за копію}) - 30\% \quad (1.1)$$

Розрахунки: (10 000 * 70) – 30% = 490 000 UAH;

Майже п’ятсот тисяч гривень. Непоганий результат, але що як гра була б безкоштовною, а замість продажу цифрових копій використовувався б інший тип монетизації? Для відповіді на це запитання розглянемо ще одну гру від цієї ж студії - «Braveland Heroes». Вона розповсюджується безкоштовно, але має, так звані, «внутрішньоігрові» товари.

Таблиця 1.4 – Дані по «Braveland Heroes» з «Google Play»

Кількість загрузок	500 000+
Середня оцінка користувачів	4,5/5
Тип монетизації	Продаж ігрових товарів
Ціна за внутрішньоігрові покупки	Від 6 до 3000 UAH

У випадку монетизації вказаної у таб. 1.4 стає важче порахувати дохід з продукту, але ми можемо спробувати отримати приблизне число. Уявимо, що як мінімум половина гравців придбала найдешевший товар і порахуємо кількість зароблених компанією грошей:

Дохід = (Кількість загрузок / 2 * мінімальну ціну за ігровий товар) – 30% (1.2)

Розрахунки: (250 000 * 6) - 30% = 1 050 000 UAH;

Отриманий результат перевищує результати, що розповсюджується через продаж електронних копій, як мінімум у два рази. Ці розрахунки підтверджують наступний факт: у більшості випадків безкоштовне розповсюдження гри з вбудованими ігровими покупками є більш фінансово ефективним. Це знають і великі компанії, використовуючи описаний тип монетизації у своїх играх. Наприклад, гра «Меч и Магия. Герои: Эра хаоса» від «Ubisoft Mobile Games».

Таблиця 1.5 – Дані по «Меч и Магия. Герои: Эра хаоса» з «Google Play»

Кількість загрузок	1 000 000+
Середня оцінка користувачів	4,3/5
Тип монетизації	Продаж ігрових товарів
Ціна за внутрішньоігрові покупки	Від 25 до 3000 UAH

Використаємо вже знайому нам формулу 1.2 і визначимо можливий дохід компанії.
Розрахунки: (500 000 * 25) – 30% = 8 750 000 UAH;

Висновки до першого розділу

Отримані вище результати дають змогу зробити висновок, що додаток по розробці покрокової стратегії на платформу Android може розраховувати на окупність, або навіть прибуток.

Після ознайомлення із функціональною частиною продуктів близьких до розроблюваної у цій роботі гри «Edge of Devastation», було виявлено, що загалом їх ігровий процес складається з етапів, що зображено на рис. 1.10, а саме:

- управління арміями на полі бою;
- вибір завдання на карті;
- покращення армії або героя.

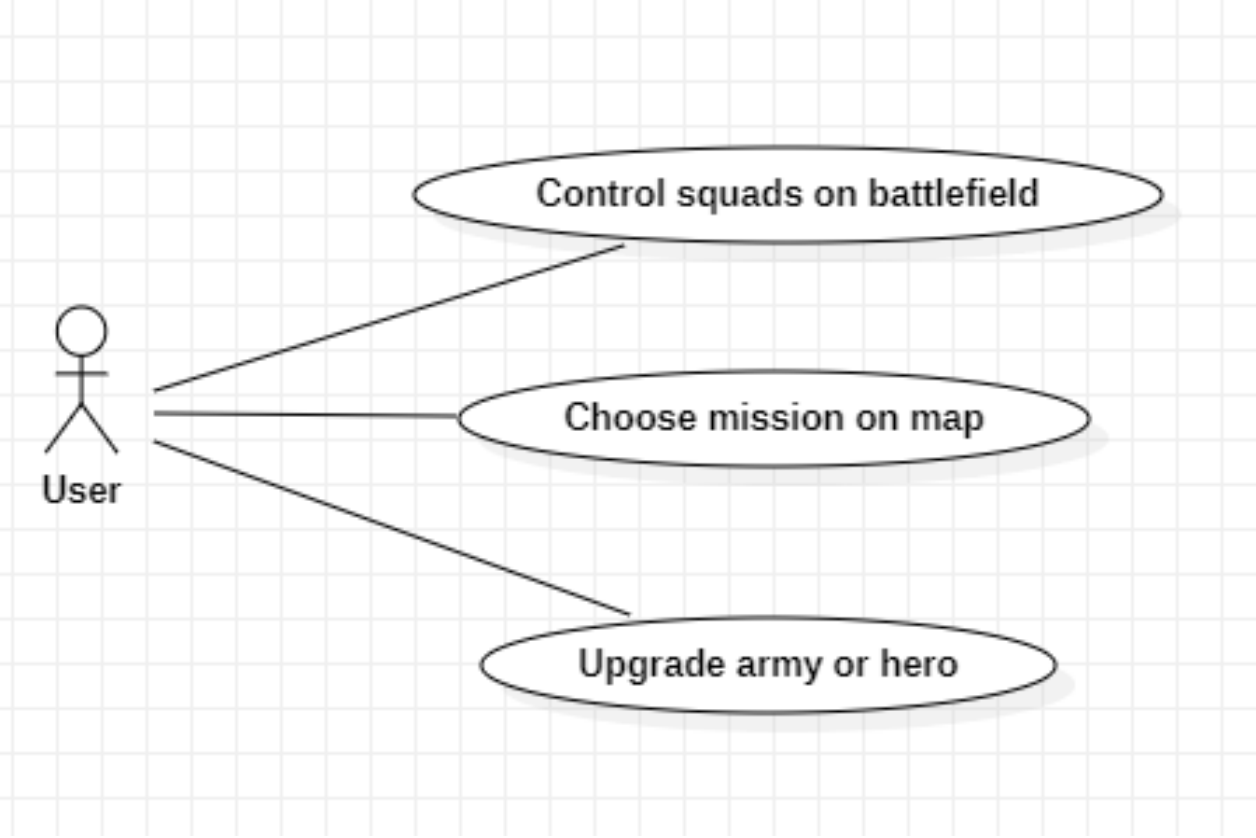


Рисунок 1.10 – Узагальнена діаграма варіантів використання конкурентних продуктів

Для підтримання конкурентоспроможності функціонал додатка було розширено, його буде детальніше розглянуто у другому розділі.

2 ФОРМУЛЮВАННЯ ФУНКЦІОНАЛУ ГРИ ТА АНАЛІЗ ВИКОРИСТАНИХ ТЕХНОЛОГІЙ

2.1 Основні вимоги до функціоналу

Основною характеристикою покрокових стратегічних ігор є дискретність ігрового процесу. Гра складається з фіксованих у часі моментів («кроків» або «ходів»), які завершуються тільки по команді гравця. Під час цих ходів гравець робить свої дії. Один хід може відповідати проміжку у багато років в ігровому світі, за які гравець встигає впоратися з подіями в кожному місті імперії і віддати накази сотням військових загонів.

Рисунок 2.1 – Ігровий процес магістерської роботи

Наприклад, на рис. 2.1 зображено гравця, який має змогу перемістити свого персонажа на будь-яку жовту клітинку. Але якщо гравець натисне кнопку кінця ходу, то більше не зможе діяти до тих пір, поки штучний інтелект або інші гравці не завершать свої ходи.

Вищеописані принципи ігрового процесу було взято за основу при формулюванні функціоналу для «Edge of Devastation». Але не можна навіть надіятись на зацікавленість у додатку, якщо він не пропонує нічого нового або індивідуального, тому далі ми розглянемо основні діаграми варіантів використання

розроблюваної гри. Це допоможе легше порівнювати продукт з конкурентами, а також дасть приблизне розуміння, які системи буде необхідно проектувати.

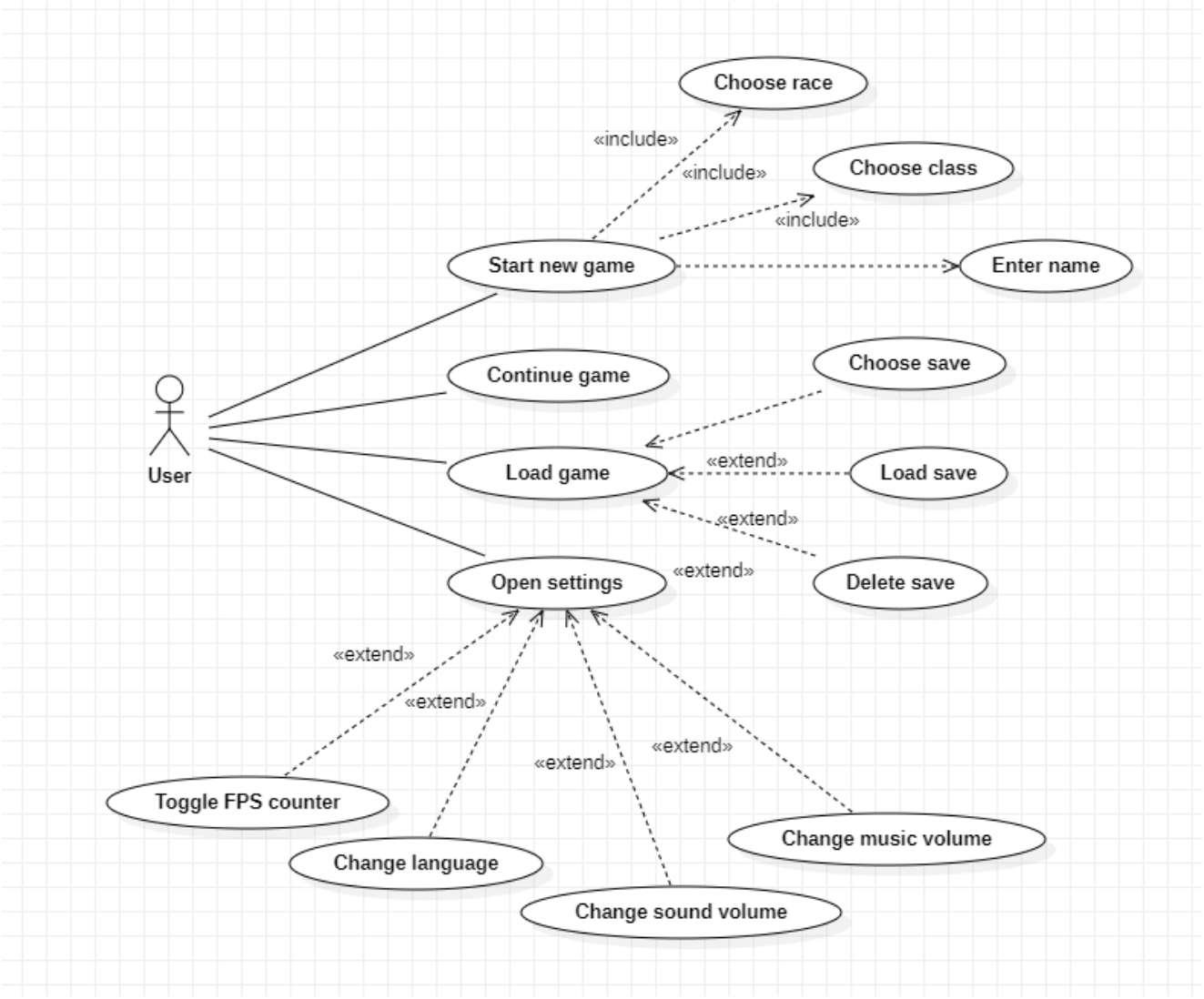


Рисунок 2.2 – Діаграма варіантів використання головного меню

Діаграма зображена на рис. 2.2 описує можливі дії користувача після входу у головне меню додатка. Дані вимоги дають зрозуміти, що потенційно необхідні наступні компоненти:

- система збереження прогресу гравця;
- система налаштувань гри.

Детальні пояснення щодо варіантів використання головного меню наведено у таб. 2.1.

Таблиця 2.1 – Пояснення до діаграми варіантів використання головного меню

Назва варіанту використання	**Детальний опис**	**Обов'язково включає додаткові дії**	**Необов'язкові можливі дії**
Розпочати нову гру	Гравець розпочинає нову історію з новим головним героєм	Вибір раси героя, вибір класу героя, вибір імені для героя	-
Продовжити гру	Система загружає останній сейв зроблений гравцем. Якщо сейв-файли відсутні – кнопку буде неможливо натиснути	-	-
Загрузити гру	Загрузити один з доступних сейв-файлів зроблених системою або гравцем.	Вибрати сейв-файл, загрузити сейв-файл	Видалити один з існуючих сейв-файлів
Відкрити налаштування	Перейти до екрану з усіма доступними для користувача налаштуваннями	-	Змінити мову, змінити гучність музики, змінити гучність звуків, включити/відключити лічильник FPS

Після початку нової, або загрузки існуючої гри, гравець потрапляє на головне місце подій – глобальну карту. З точки зору варіантів використання це найбільш насичена сцена. Усі можливі дії гравця зображено на рис. 2.3. Беручи до уваги розглянуту діаграму висуваємо припущення про необхідність розробки наступних компонентів:

- контролер інтерфейсу користувача;
- діалогова система;
- система генерації комірок;
- контролер камери;
- система взаємодії об'єктів;
- система завдань.

Детальні пояснення щодо варіантів використання головної ігрової сцени наведено у таб. 2.2.

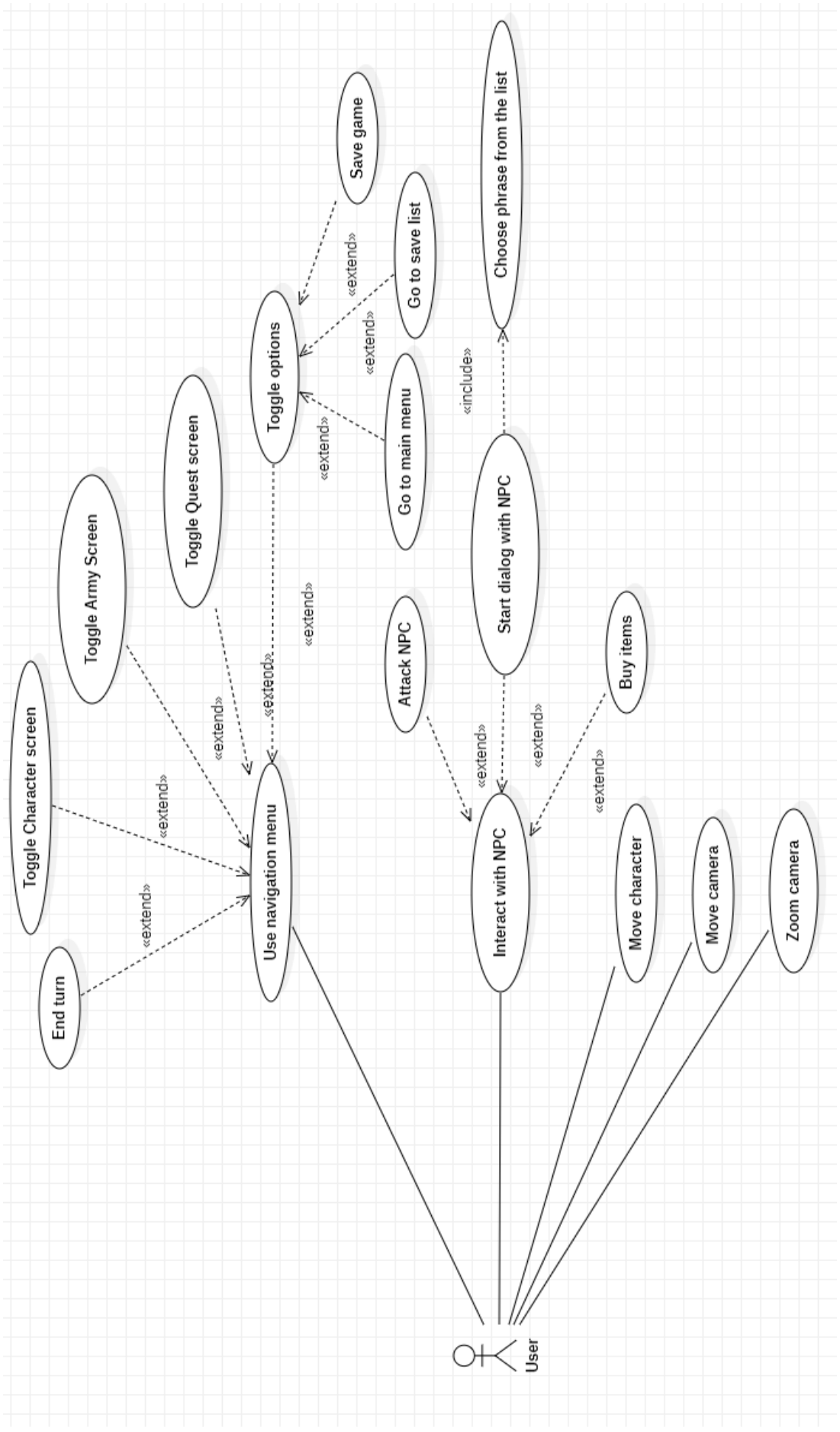


Рисунок 2.3 – Діаграма способів використання головної ігрової сцени

Таблиця 2.2 – Пояснення до діаграми варіантів використання головної сцени

Назва варіанту використання	**Детальний опис**	**Обов'язково включає додаткові дії**	**Необов'язкові можливі дії**
Використання навігаційного меню	Активація одної з доступних кнопок на навігаційному меню.		Закінчити хід, відкрити екран героя, відкрити екран армії, відкрити екран завдань, відкрити меню опцій.
Відкрити меню опцій	Відобразити на екрані невелике допоміжне меню з додатковими функціями.		Зберегти гру, перейти до списку доступних сейв-файлів, перейти у головне меню.
Взаємодія з NPC	Взаємодія з будь-яким типом NPC на глобальній карті. З нейтральними можливо розпочати діалог і вони не атакують головного героя. Агресивні персонажі атакують головного героя у радіусі 2 комірок.		Атакувати персонажа, розпочати діалог із персонажем, відкрити вікно покупки предметів.
Атакувати NPC	Означає ініціалізацію бойової сцени, де генеруються армії гравця і другого персонажу. Якщо у гравця армія відсутня – бій автоматично закінчується програшом.		
Розпочати діалог з NPC	Якщо персонаж має доступні діалоги з головним героєм – відкрити спеціальний інтерфейс, де видно доступні фрази.		

Продовження таблиці 2.2

Пересувати персонажа	Пересувати героя по ігровому полю. За один хід можливе пересування лише на де-яке встановлене число комірок, яке залежить від характеристик героя.	Вибрати одну з активних для переміщення комірок.	
Пересувати камеру	Пересувати камеру по ігровому полю до його границь. Якщо камера досягає границі – заблокувати пересування.		
Приближати/віддаляти камеру	Віддаляти або приближати камеру, тим самим збільшуючи або зменшуючи область видимості гравця. Можливо лише у заданому проміжку. Якщо віддалення камери спровокує вихід оглядової зони за границі ігрового поля – заборонити його.		

Останнім, але не менш важливим місцем подій є сцена бою. Тут відбувається зіткнення армій головного героя і супротивника.

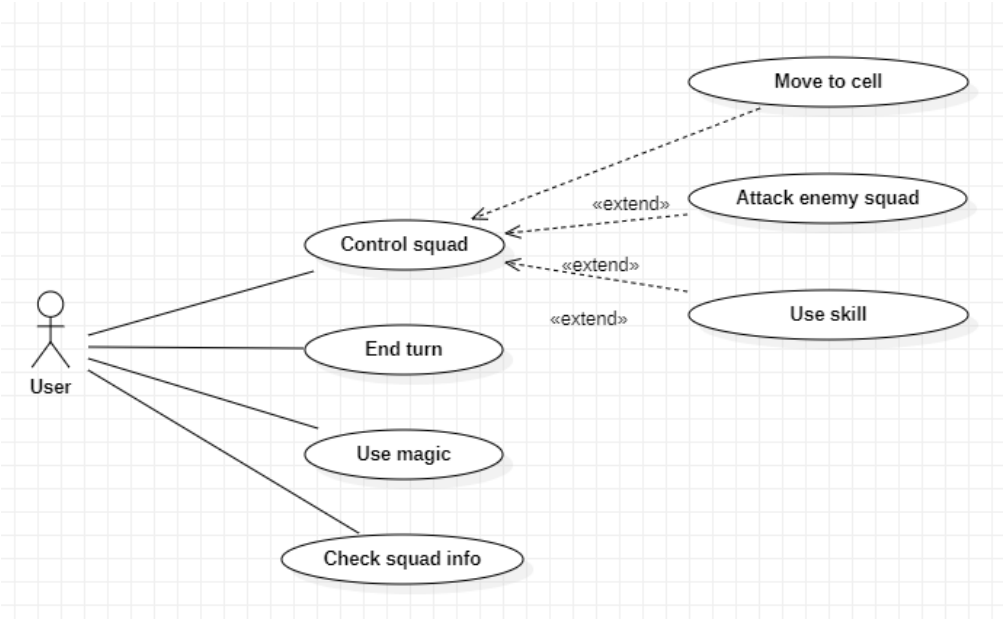


Рисунок 2.4 – Діаграма способів використання сцени бою

Полем бою є сітка комірок обмеженого розміру, по якій пересуваються юніти. Для того, щоб перемогти, необхідно здолати усі присутні ворожі армії.

Можливі дії гравця показано на рис. 2.4. Виходячи з даної діаграми і вищезазначених фактів може з’явитися необхідність у наступних компонентах:

- контролер битви;
- контролер інтерфейсу гравця на полі бою;
- контролер штучного інтелекту;
- система використання навиків.

Детальні пояснення щодо варіантів використання сцени бою наведено у таб. 2.3.

Таблиця 2.3 – Пояснення до діаграми варіантів використання сцени бою

Назва варіанту використання	**Детальний опис**	**Обов'язково включає додаткові дії**	**Необов'язкові можливі дії**
Управління арміями	Управління одною з армій, що розташовані на полі бою.	-	Перемістити армію до комірки, атакувати армію ворога, використати навик
Закінчити хід	Передати хід до ворожої армії. Завершується автоматично, якщо у армій гравця не залишилося очок пересування.	-	-
Використати магію	Якщо герой вже має магічні навички, відкрити меню доступних для використання на полі бою.	Вибрати один з навичок, використати вибраний навик.	-
Переглянути інформацію про армії	При виборі будь-якої армії переглянути відображені відомості, такі як кількість бійців, активні ефекти тощо.	-	-

2.2 Огляд можливостей Unity

Редактор Unity має простий Drag & Drop інтерфейс, який легко налаштовувати, що складається з різних вікон, завдяки чому можна проводити налагодження гри прямо в редакторі. Движок використовує для написання скриптів C #. Раніше підтримувалися також Boo (діалект Python, підтримку прибрали в 5-й версії) і модифікація JavaScript, відома як UnityScript (підтримка припинена в версії 2017.1). Розрахунки фізики виробляє фізичний движок PhysX від NVIDIA. Графічний API - DirectX (на даний момент DX 11, підтримується DX 12)

Проект додатку гри в Unity ділиться на сцени (рівні) - окремі файли, що містять свої ігрові світи зі своїм набором об'єктів, сценаріїв, і налаштувань. Сцени можуть містити в собі як, власне, об'єкти (моделі), так і порожні ігрові об'єкти - об'єкти, які не мають моделі («пустушки»). Об'єкти, в свою чергу містять набори компонентів, з якими і взаємодіють скрипти. Також у об'єктів є назва (в Unity допускається наявність двох і більше об'єктів з однаковими назвами), може бути тег (мітка) і шар, на якому він повинен відображатися. Так, у будь-якого об'єкта на сцені обов'язково присутній компонент Transform - він зберігає в собі координати місця розташування, повороту і розмірів об'єкта по всіх трьох осях. У об'єктів з видимої геометрією також за замовчуванням присутній компонент Mesh Renderer, що робить модель об'єкта видимою.

До об'єктів можна застосовувати колізії (в Unity т. Н. Колайдери - collider), яких існує декілька типів.

Також Unity підтримує фізику твердих тіл і тканини, а також фізику типу Ragdoll. У редакторі є система успадкування об'єктів; дочірні об'єкти будуть повторювати всі зміни позиції, повороту і масштабу батьківського об'єкта. Скрипти в редакторі прикріплюються до об'єктів у вигляді окремих компонентів.

При імпорті текстури в Unity можна згенерувати alpha-канал, mip-рівні, normal-map, light-map, карту відображень, проте безпосередньо на модель текстуру прикріпити не можна - буде створено матеріал, з яким буде призначений шейдер, і потім матеріал прикріпиться до моделі. Редактор Unity підтримує написання і редагування шейдеров. Редактор Unity має компонент для створення анімації, але також анімацію можна створити попередньо в 3D-редакторі та імпортувати разом з моделлю, а потім розбити на файли.

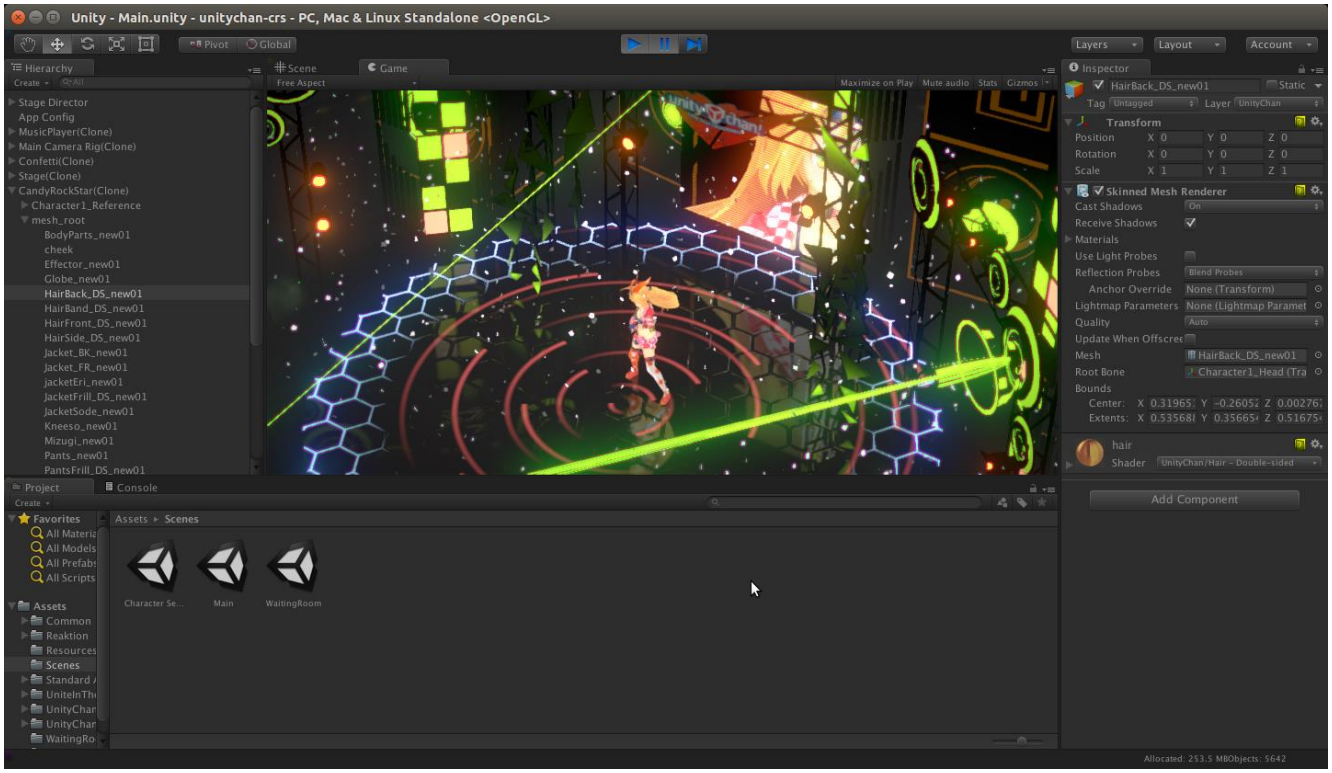


Рисунок 2.5 – Інтерфейс Unity

Unity 3D підтримує систему Level Of Detail (скор. LOD), суть якої полягає в тому, що на далекій відстані від гравця високодеталізовані моделі замінюються на менш деталізовані, і навпаки, а також систему Occlusion culling, суть якої в тому, що у об'єктів, що не потрапляють в поле зору камери не візуалізується геометрія і колізія, що знижує навантаження на центральний процесор і дозволяє оптимізувати код. При компіляції коду створюється виконуваний (.exe) файл гри (для Windows), а в окремій папці - дані гри (включаючи всі ігрові рівні і спільні бібліотеки).

Движок підтримує безліч популярних форматів. Моделі, звуки, текстури, матеріали, скрипти можна запаковувати в формат .unityassets і передавати іншим розробникам, або викладати у вільний доступ. Цей же формат використовується у внутрішньому магазині Unity Asset Store, в якому розробники можуть безкоштовно і за гроші викладати в загальний доступ різні елементи, потрібні при створенні ігор. Щоб використовувати Unity Asset Store, необхідно мати акаунт розробника Unity. Unity має всі потрібні компоненти для створення мультиплеєра. Також можна використовувати відповідний користувачеві спосіб контролю версій. Наприклад, Tortoise SVN або Source Gear.

2.3 Аналіз технічних можливостей

При розробці для мобільних платформ неможливо не звернути увагу на досить значні обмеження у потужності. Як показано на рисунку 2.2, сучасні смартфони майже зрівнялися із офісними машинами.

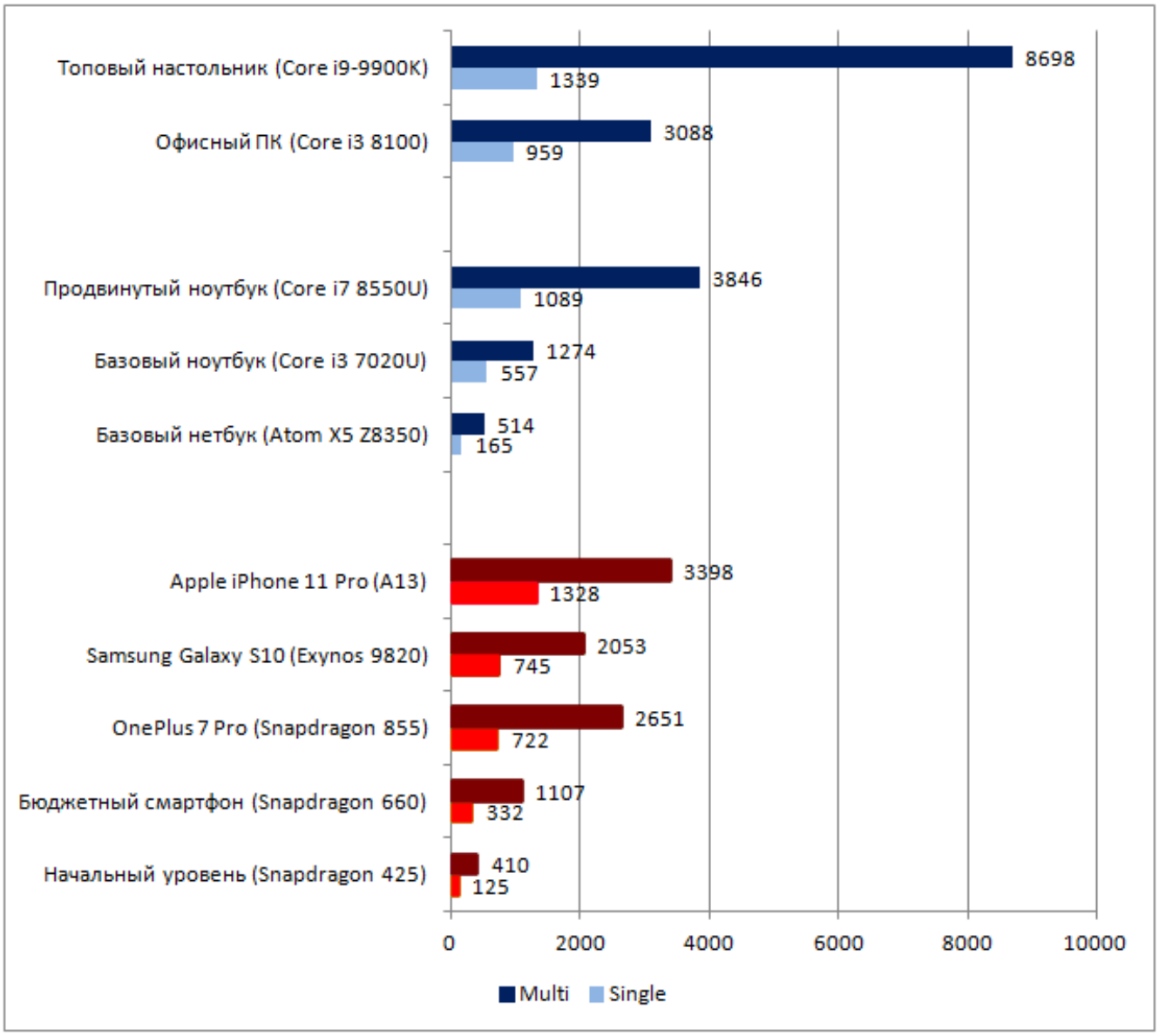


Рисунок 2.6 – Тести процесорів у програмі «Geekbench»

На жаль, мова йде лише про невеликі відрізки часу: через відсутність активного охолодження мобільний процесор змушений скидати робочі частоти, що веде до зменшення потужності.

Повноцінно порівняти процесорні архітектури дуже важко, так як ARM належить до типу RISC, а x86 до CISC. За рахунок меншого числа команд і меншої кількості блоків ARM-чіп повинен виконувати окремі команди швидше і енергоефективніше. Але як тільки мова заходить про виконання складних функцій, під які у x86 є заготовлені апаратні блоки і набори команд, ARM, в теорії, буде показувати гірші результати.

Показники продуктивності мобільних платформ нагадують про те, що оптимізація додатка повинна бути на досить високому рівні. Для контролю цього аспекту і виявлення недоліків буде необхідно проводити тестування білдів гри.

Висновки до другого розділу

Були розглянуті основні вимоги до функціоналу гри. Проведено аналіз великої кількості ресурсів, що стало основою для початку розробки додатку. Були розглянуті доступні платформи, операційні системи та їх технічні обмеження. Розглянуто можливості Unity. Обрано саме цей движок, тому що він підтримує безліч популярних форматів. Моделі, звуки, текстури, матеріали, скрипти можна запаковувати в формат .unityassets і передавати іншим розробникам, або викладати у вільний доступ.

3 ПРОЕКТУВАННЯ ПРОЦЕСУ РОЗРОБКИ І АРХІТЕКТУРИ КОМПОНЕНТІВ ГРИ

3.1 Етапи створення ігри

Коли мова йде про інді-розробку, стає важко слідувати стандартам і правилам, що сформувалися враховуючи ресурси і можливості великих компаній. Тому важливо проявляти гнучкість у більшості моментів, особливо – у плануванні людських ресурсів та релізів продукту.

Зазвичай, створення гри підкоряється системі етапів наведеній у таб. 3.1.

Таблиця 3.1 – Звичні етапи розробки додатку

Назва етапу	Опис
Pre-Alpha	Період від початку розвитку до виходу зі стадії Альфа. Також називають програми, які ще не досягли альфа-або бета-стадії, але пройшли стадію розробки, для початкової оцінки функціональності в дії. На відміну від альфа- та бета-версій, початковий етап може не включати повний спектр функціональності програми. У цьому випадку маються на увазі всі дії, які виконуються під час розробки та розробки програми, аж до тестування. Такі дії включають розробку дизайну, аналіз вимог, саму розробку додатків, а також налагодження окремих модулів.
Closed Alpha	Етап тестування програми в цілому проводиться тестерами, як правило, не розробниками програмного забезпечення, але, всередині організації або громади, що розробляє продукт. Це також може бути етапом додавання нових функціональних можливостей. Програми на цьому етапі можна використовувати лише для ознайомлення з майбутніми можливостями.
Open Alpha (опціонально)	Даний етап може йти паралельно з Closed Alpha, однак закінчиться все одно пізніше. Відкриту альфу використовують для декількох цілей: зібрати відгуки зацікавлених користувачів щодо функціоналу гри, можливо будуть розглянуті нові фічі, а також для раннього тестування і усунення проблем з продуктивністю гри на різних апаратних платформах.

Продовження таблиці 3.1

Closed Beta(опціонально)	Етап активного тестування та налагодження програми, яка пройшла альфа-тестування. Програми на цьому рівні можуть використовуватись іншими розробниками програмного забезпечення для перевірки сумісності. Однак програми цього етапу можуть містити досить велику кількість помилок. Оскільки бета-продукт не є остаточною версією, а публічне тестування проводиться на власний ризик та небезпеку, виробник не несе відповідальності за шкоду, заподіяну в результаті використання бета-версії. Закрите бета тестування розповсюджується лише з дозволу розробника обмеженому колу лиць.
Open Beta	Суть та сама, що й у закритого бета тестування, однак у відкритому бета тестуванні продукт може спробувати кожний бажаючий.
Release	Етап релізу передбачає завершення розробки і виправлення багів, і остаточну передачу готового продукту заказнику або користувачам. Починаючи с цього етапу і аж до кінця життєвого циклу гри розробник виділяє де-які ресурси для підтримки продукту.

Однак для «Edge of Devastation» ці етапи було спрощено, так як, наприклад, «Pre-Alpha» і «Closed Alpha» у контексті додатку, що спирається на постійний відгук і підтримку суспільства, сенсу не мають. Перероблені кроки розробки наведено у таб. 3.2.

Таблиця 3.2 – Етапи розробки «Edge of Devastation»

Назва етапу	**Опис**
Alpha	Загальний процес розробки, який диктує, що більшість функцій слід реалізувати тут. Поєднує ознаки закритого і відкритого альфа тестування, так як час від часу ком'юніті буде отримувати білди із новим функціоналом, що не був доведений до повноцінного робочого стану.
Closed Beta	Найбільша частина закритої бета-версії - це внутрішнє тестування, яке повинно дати нам відгук про всі ігрові системи та їх продуктивність. Тим не менш, деякі фічі можуть бути відкладені до цього етапу.
Open Beta	Ця частина призначена для відкритого тестування та виправлення багів, але все ж ми можемо спланувати тут декілька спірних фіч, які потребують зворотного зв'язку спільноти.
Release	Випуск гри означає відсутність нових функцій, лише підтримку у вигляді виправлення багів. На момент випуску гра повинна з'явитися як у «Google Play», так і у «AppStore».

3.2 Планування людських ресурсів

Через коливання кількості активних членів команди, а також плаваючу можливість працювати над додатком – виникають великі труднощі із плануванням реалізації тих чи інших компонентів. Було вирішено починати планування після створення базової версії, яка включає лише генерацію комірок і пересування персонажа, щоб підвищити навички володіння Unity, а також встановити приблизну кількість людей, що залишилися працювати над додатком і далі.

На момент початку створення діаграми Ганта у 2023 році команда складалася з наступних людей:

- двох C# програмістів;
- музикального композитору.

Отриманий результат, розроблений враховуючи розміри і зайнятість команди зображено на рис. 3.1.

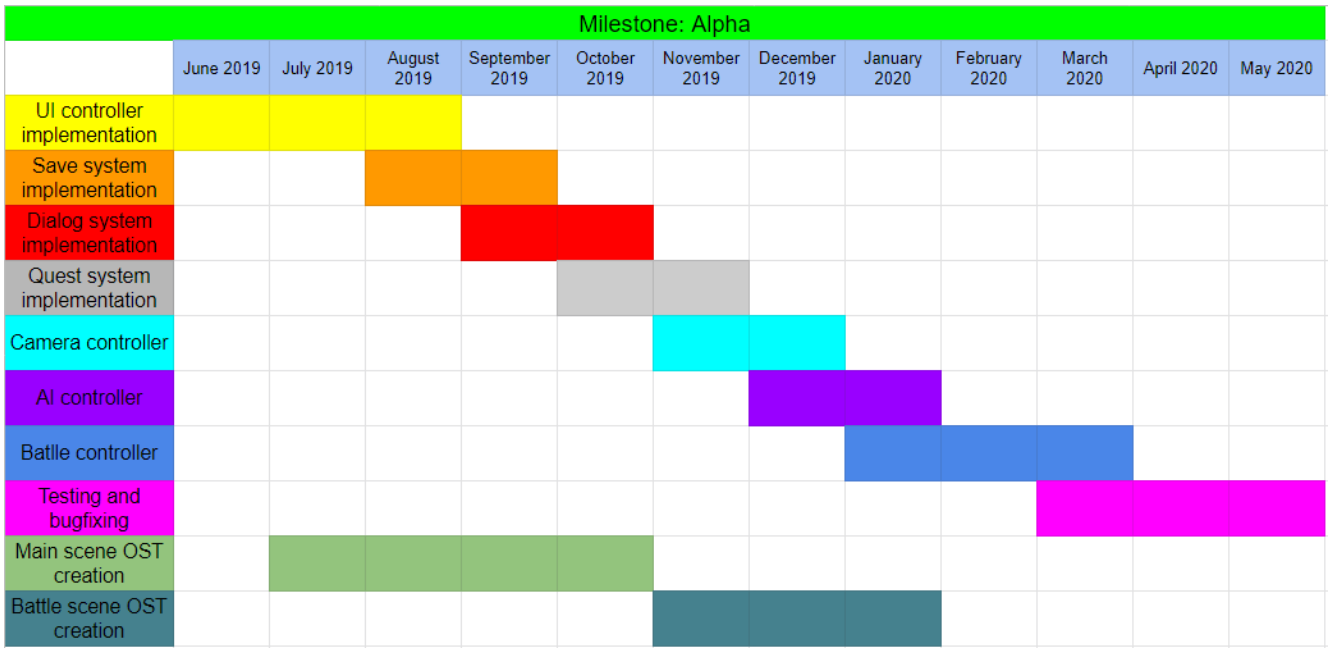


Рисунок 3.1 – Діаграма Ганта

3.3 Архітектура основних компонентів ігрового додатку

Проектування компонентів гри відбувається у тому ж порядку, як була запланована їх імплементація, тому почнемо з контролеру інтерфейсу користувача на головній ігровій сцені.

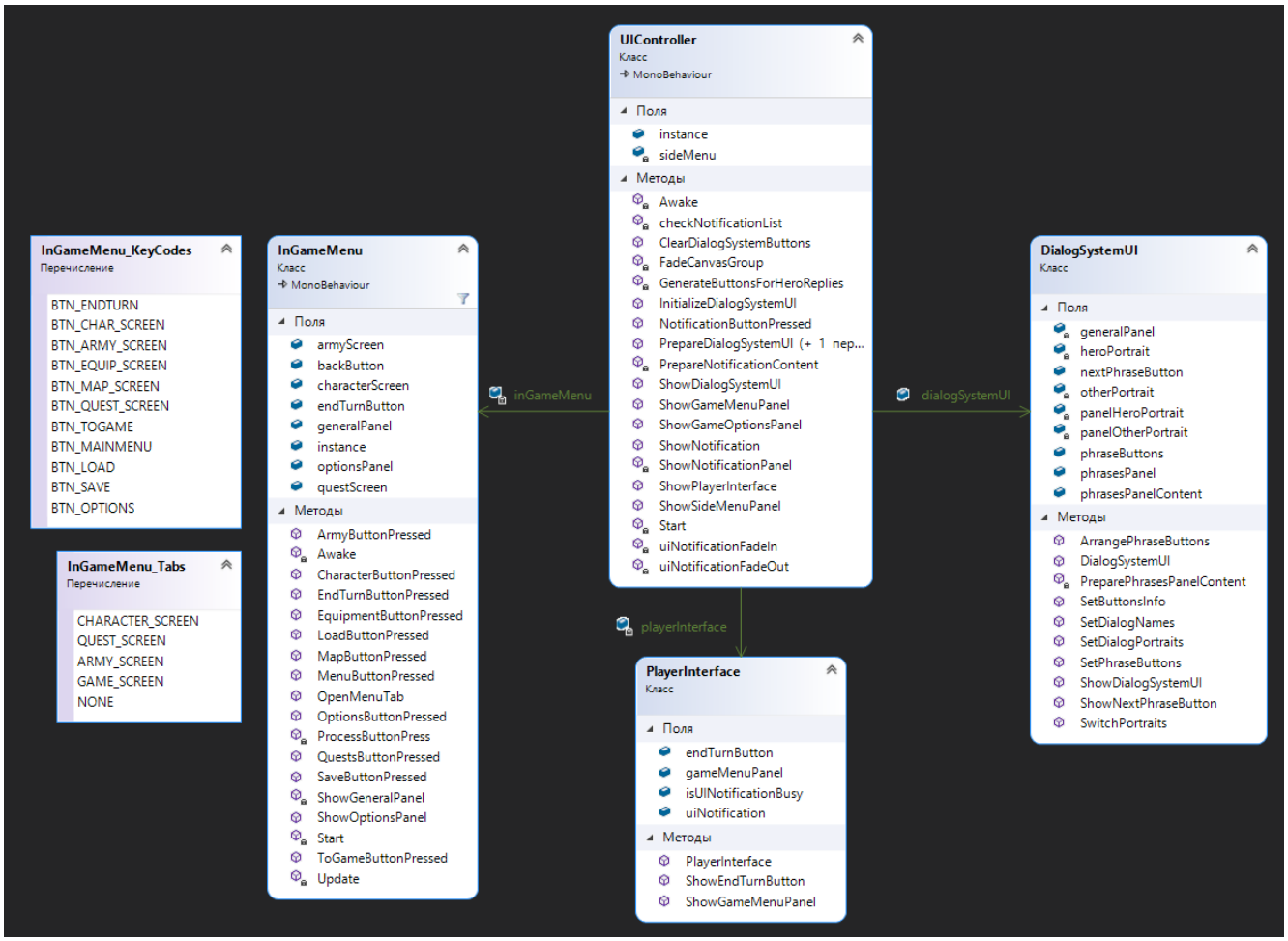


Рисунок 3.2 – Спрощена діаграма класів для контролеру інтерфейсу користувача

На рис. 3.2 зображено основний клас «UIController», який відповідає за загальну поведінку інтерфейсу користувача і має наступні допоміжні класи у якості полів:

- «InGameMenu», що відповідає за функціонал навігаційного меню;
- «DialogSystemUI», що контролює інтерфейс діалогової системи;
- «PlayerInterface», що контролює ігрові повідомлення, а також відповідає за те, чи увімкнутий інтерфейс користувача взагалі.

Наступною є система зберігання стану гри. Вона відповідає за створення файлу на файловій системі, що містить інформацію, яка допоможе відновити властивості усіх об’єктів гри. На рис. 3.3 зображено основний клас «SaveSystem»,

що має список типу «GameState», який містить усі необхідні дані про гру для запису в файл. «GameState» у свою чергу є композицією класів стану, що описують менші об’єкти, наприклад:

- «QuestState»;
- «CameraState»;
- «DialogState», тощо.

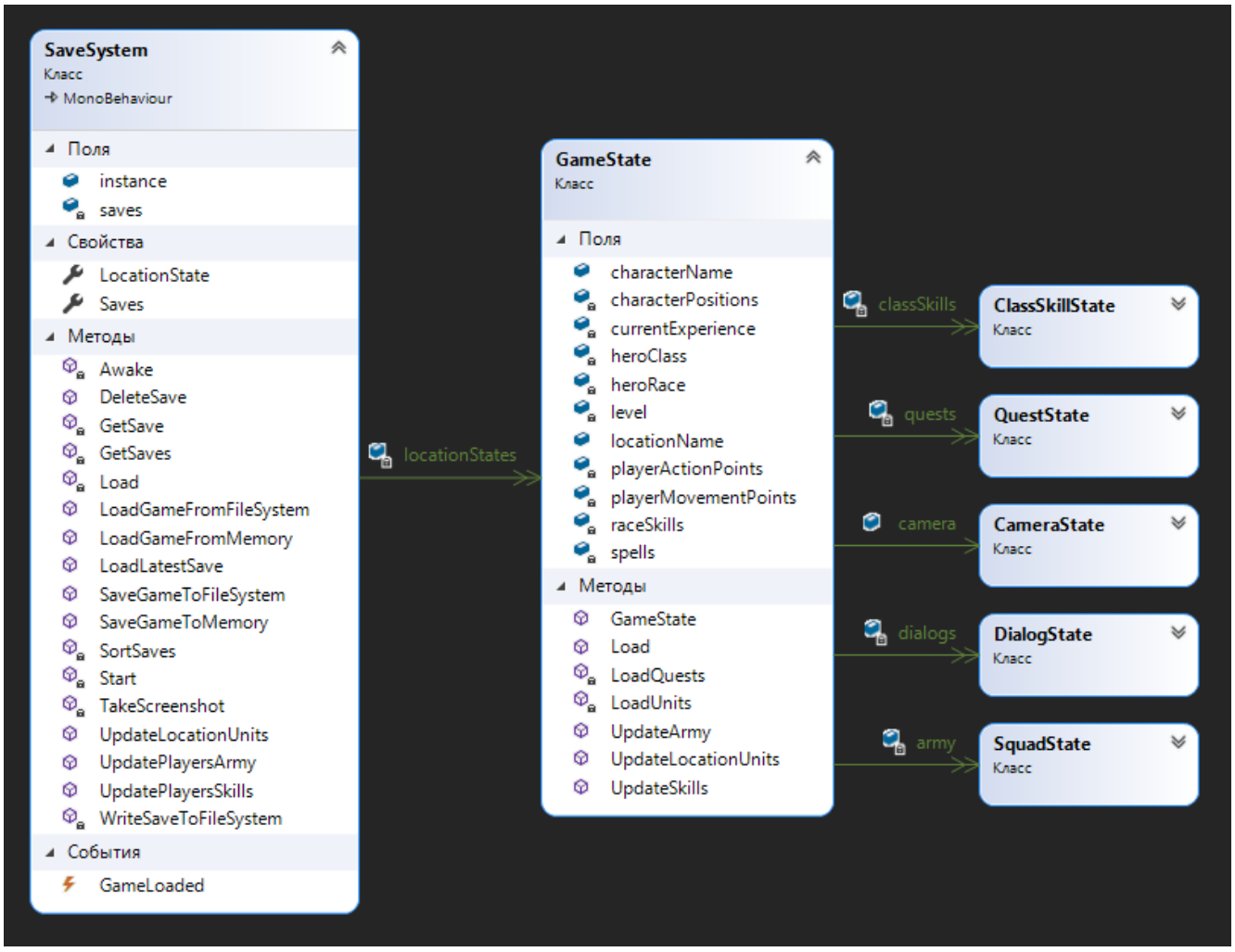


Рисунок 3.3 – Спрощена діаграма класів для системи зберігання прогресу

Далі йде діалогова система. Клас «DialogSystem», зображений на рис. 3.4, відповідає за можливість розпочати діалог, контролює активний діалог, а також сигналізує інтерфейсу користувача про необхідність створити певні об’єкти на екрані. В якості полів використовуються наступні класи:

- «Dialog», який містить ідентифікатор діалогу і усі доступні фрази. У даному класі використані допоміжний тип «Phrase», що зберігає у собі

ідентифікатор однієї фрази NPC та іншу інформацію, таку як текст фрази, і «HeroReplies», що зберігає список можливих фраз для головного героя;

- «DialogsXMLParser», який відповідає за вичитування інформації з xml файлу, що описує характеристики усіх доступних для сцени діалогів.

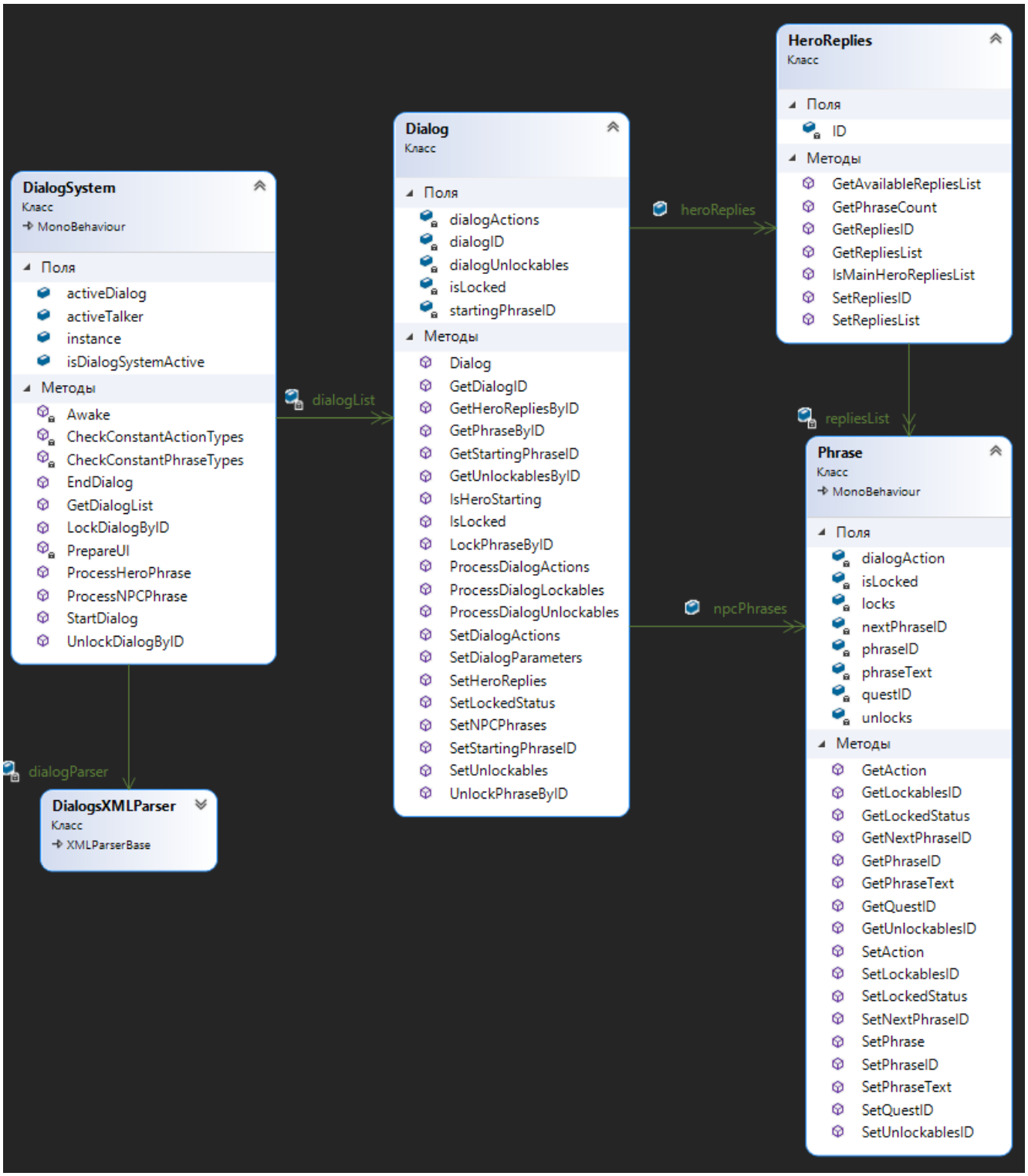


Рисунок 3.4 – Спрощена діаграма класів діалогової системи

Система завдань. «QuestSystem» - один з основних класів зображених на рис. 3.5. Основною його метою є загрузка інформації про квест і зберігання списку задач, що так чи інакше пов’язані с гравцем. Використовує допоміжний клас «QuestLoader», який відповідає за читання інформації з xml файлу, у якості поля. Іншим основним класом у системі виступає «Quest», що містить такі характеристики як стан завдання, виповнені задачі тощо. Використовує наступні допоміжні типи:

- «QuestData», що містить інформацію про задачі та нагороди завдання;
- «Task», що містить властивості конкретної задачі;

Рисунок 3.5 – Спрощена діаграма класів системи завдань

При роботі з Unity було виявлено, що для імплементації контролеру камери не потрібен окремий компонент – буде достатньо одного класу, що контролює

єдиний на сцені Unity об’єкт камери.

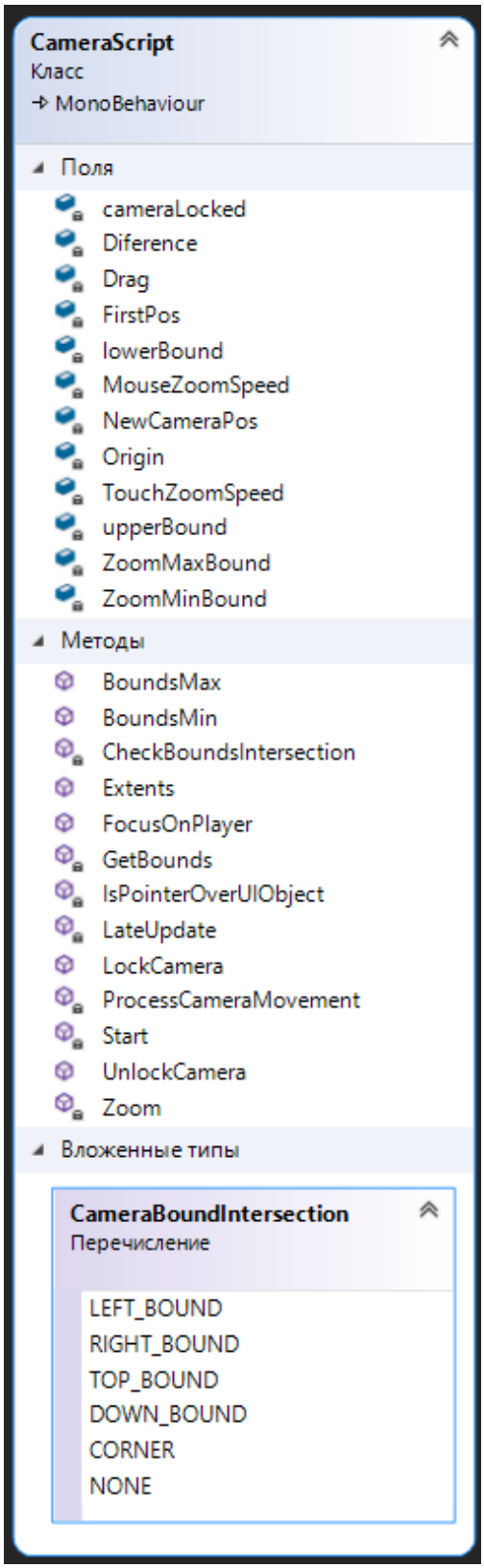


Рисунок 3.6 – Клас «CameraScript»

Штучний інтелект є одним із головних аспектів багатьох ігор, а тим паче покрокової стратегії. На даний момент для гри спроектовано простий штучний інтелект, що не стане сильним супротивником, але згодиться для демонстраційної версії. Основний клас «EasyLevelAI», що зображено на рис. 3.7, має невеликий список можливих дій, наприклад:

- атакувати найближчий ворожий загін;
- перемістити загін на комірку;
- спробувати відступити.

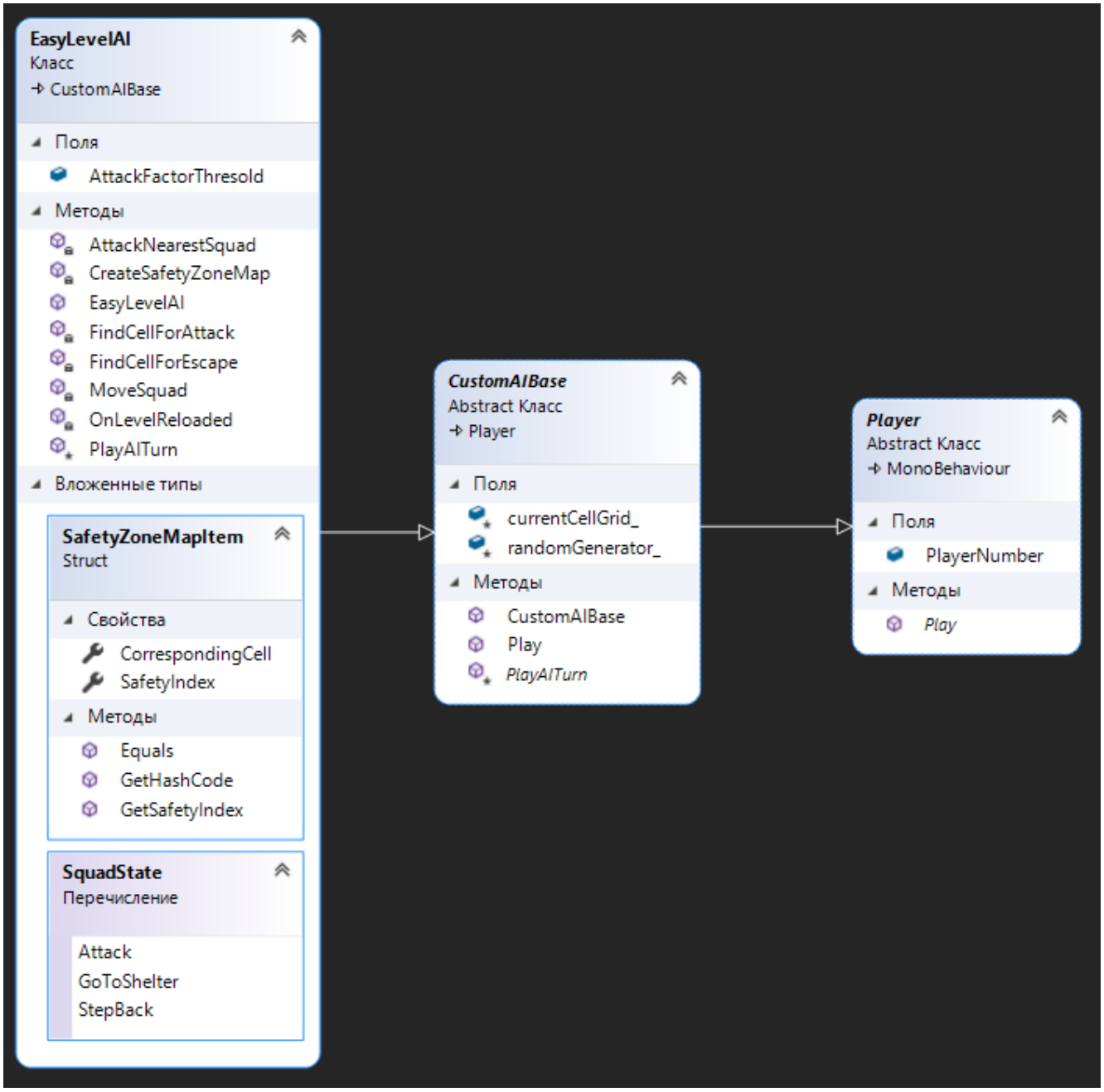


Рисунок 3.7 – Спрощена діаграма класів системи штучного інтелекту

Останнім, але не менш важливим компонентом для проектування є контролер бою. На рис. 3.8 зображено два основних класи. «BattleSystem» присутній на головній ігровій сцені і розпочинає бій за необхідністю. Далі відбувається загрузка сцени бою та ініціалізація «BattleController», який відповідає за стан поля бою і повинен відстежувати перемогу або програш гравця. Використовує класс «BattleUnit», який містить характеристики загону, наприклад:

- рівень здоров’я;
- силу атаки;

- навички, тощо.

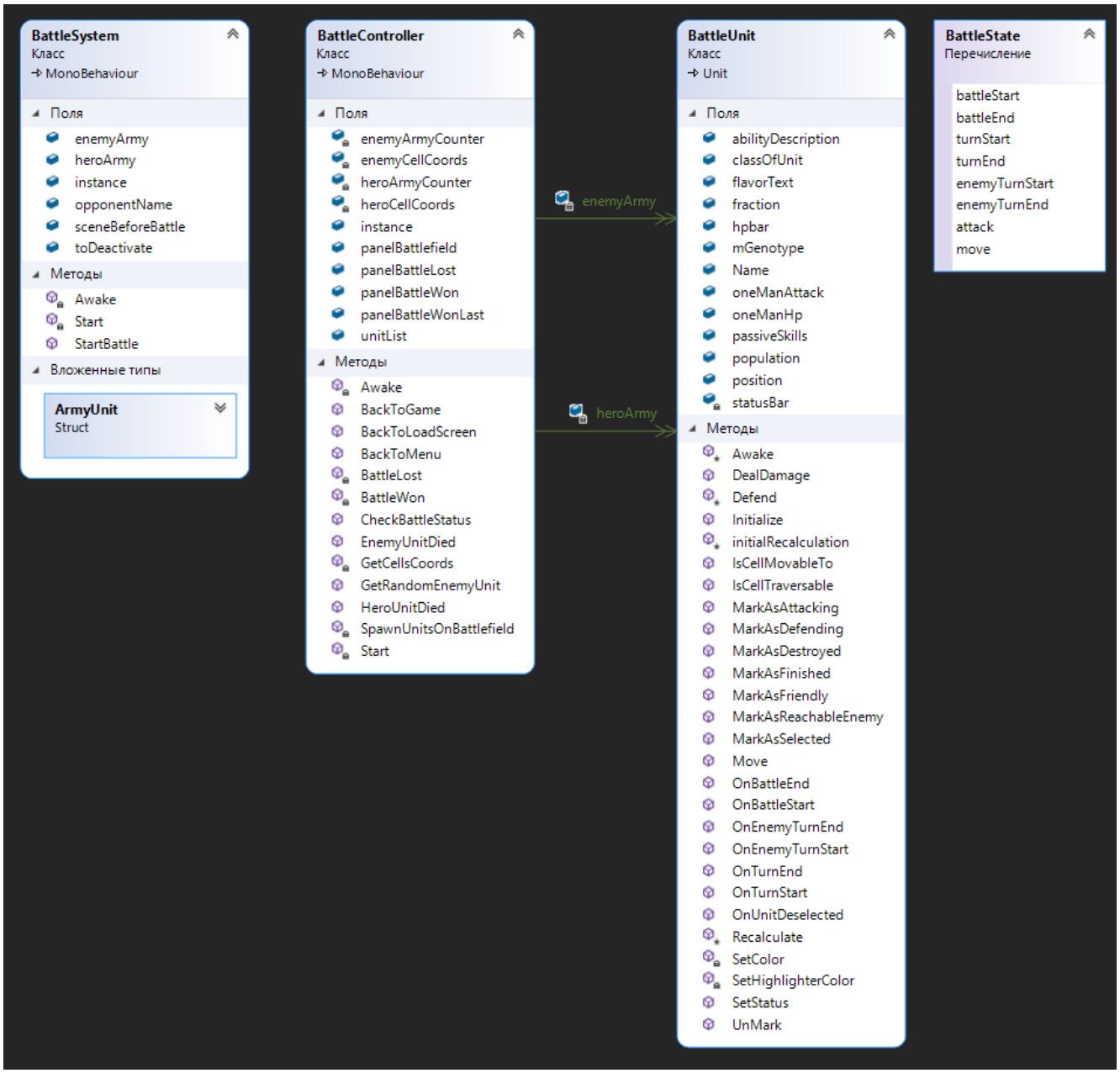


Рисунок 3.8 – Спрощена діаграма класів контролеру бою

Висновки до третього розділу

У цьому розділі були розглянуті звичні етапи розробки додатку, данні наведені у табл. 3.1. Етапи розробки «Edge of Devastation» результати у табл. 3.2.

При роботі з Unity було виявлено, що для імплементації контролеру камери не потрібен окремий компонент – буде достатньо одного класу, що контролює єдиний на сцені Unity об’єкт камери.

Штучний інтелект є одним із головних аспектів багатьох ігор, а тим паче покрокової стратегії. На даний момент для гри спроектовано простий штучний інтелект, що не стане сильним супротивником, але згодиться для демонстраційної версії.

4 РОЗРОБКА КОМПОНЕНТІВ ГРИ

У даному розділі буде розглянуто головні класи кожного зі спроектованих компонентів, а також додаткові інструменти, що було реалізовано, наприклад, для допомоги у тестуванні. Більшість уваги буде приділено методам, що містять велику частину бізнес-логіки.

4.1 Огляд основних класів і методів

Клас «UIController». Доступ до використання його інтерфейсів надається через статичний екземпляр. Така політика була запроваджена для більшості реалізованих систем. Розглянемо наступні методи класу:

- «PrepareDialogSystemUI»;
- «InitializeDialogSystemUI»;
- «GenerateButtonsForHeroReplies»;
- «ShowPlayerInterface».

```
Ссылок: 2
public void PrepareDialogSystemUI(HeroReplies replies)
{
    Debug.Log(replies.GetPhraseCount() + "PHRASE COUNT IN " + replies.GetRepliesID());
    dialogSystemUI.SetPhraseButtons(GenerateButtonsForHeroReplies(replies.GetPhraseCount()));

    dialogSystemUI.ArrangePhraseButtons();
    dialogSystemUI.SetButtonsInfo(replies.GetAvailableRepliesList());
    dialogSystemUI.ShowNextPhraseButton(false);
    dialogSystemUI.phrasesPanel.SetActive(true);

    dialogSystemUI.SwitchPortraits(true);
}


Ссылок: 2
public void PrepareDialogSystemUI(Phrase phrase)
{
    dialogSystemUI.ShowNextPhraseButton(true);
    dialogSystemUI.phrasesPanel.SetActive(false);

    dialogSystemUI.nextPhraseButton.GetComponent<Phrase>().SetPhrase(phrase);
    dialogSystemUI.nextPhraseButton.GetComponentInChildren<Text>().text = phrase.GetPhraseText();

    dialogSystemUI.SwitchPortraits(false);
}
```

Рис. 4.1 – Імплементація «PrepareDialogSystemUI» і його перевантаження

Метод «PrepareDialogSystemUI» визивається кожен раз, коли гравець натискає на фрази у діалогах. Задача методу – згенерувати необхідну кількість кнопок і показати їх на екрані. Перевантаження методу необхідне, щоб розлічати натискання на фрази NPC та героя.

«InitializeDialogSystemUI» визивається коли гравець розпочинає діалог із NPC. Задача методу – підготовити спрайти та імена персонажів, що беруть участь у діалозі.

«GenerateButtonsForHeroReplies» - допоміжний метод, але виконує одну з головних функцій інтерфейсу діалогової системи. Визивається всередині «PrepareDialogSystemUI» для генерації окремих кнопок для кожної фрази.

«ShowPlayerInterface» визивається, коли необхідно увімкнути або вимкнути навігаційне меню ті інші елементи інтерфейсу гравця, наприклад, коли починається і закінчується діалог.

Клас «SaveSystem». Доступ до використання його інтерфейсів надається через статичний екземпляр. Розглянемо наступні методи класу:

- «SaveGameToMemory»;
- «WriteSaveToFileSystem»;
- «LoadLatestSave».

```
ссылка: 1
private void WriteSaveToFileSystem(Texture2D screenshot)
{
    var save = new Save(screenshot);

    var formatter = new BinaryFormatter();
    if(!Directory.Exists(Application.persistentDataPath))
    {
        Directory.CreateDirectory(Application.persistentDataPath);
    }
    var file = File.Create(Application.persistentDataPath + "/" + save.data.saveName);
    formatter.Serialize(file, save);
    UnityEngine.Object.Destroy(screenshot);
    file.Close();
    saves.Add(save.data);
    SortSaves();
    var notification = UINotification.CreateGameSavedNotification(save.data.DisplayedName);
    ui.UIController.instance.ShowNotification(notification);
}
```

Рисунок 4.2 – Імплементація «WriteSaveToFileSystem»

Метод «SaveGameToMemory» необхідний для зберігання поточного стану гри у полі класу. Це необхідно, наприклад, щоб після сцени бою повернути гравця назад на глобальну карту.

«WriteSaveToFileSystem» виконує найголовнішу функцію системи – зберігає сейв-файл на файловій системі пристрою. Його імплементацію зображено на рис. 4.2.

Метод «LoadLatestSave» визивається, коли користувач натискає кнопку «Продовжити гру» у головному меню додатка. Він перевіряє, чи існує хоч один сейв-файл і виконує загрузку останнього створеного.

Клас «DialogSystem». Доступ до використання його інтерфейсів надається через статичний екземпляр. Розглянемо наступні методи класу:

- «StartDialog»;
- «EndDialog»;
- «ProcessNPCPhrase»;
- «ProcessHeroPhrase».

«StartDialog» відпрацьовує кожен раз, коли гравець взаємодіє із NPC, який має доступний діалог. Тут відбувається перевірка, чи існує діалог у системі, а також перша ініціалізація інтерфейсу діалогової системи. Реалізацію метода зображено на рис. 4.3.

«EndDialog» визивається після опрацювання останньої фрази у діалозі. Метод знищує усі елементи інтерфейсу діалогової системи і повертає звичний інтерфейс користувача глобальної карти.

Методи «ProcessNPCPhrase» і «ProcessHeroPhrase» викликають відповідне перевантаження метода «PrepareDialogSystemUI» і відпрацьовують при натисканні на фразу NPC або героя відповідно.

У контексті системи завдань буде обрано клас «Quest», так як у ньому зосереджена головна логіка завдань. Розглянемо наступні методи класу:

- «UpdateQuest»;
- «FailQuest»;
- «CompleteQuest»;
- «TaskCompleted».

Метод «UpdateQuest» використовується для просування гравця по задачам завдання. Визивається при виконанні вимог по завданню.

«FailQuest» використовується для того, щоб перевести завдання в стан «провалено» після відповідних дій гравця.

```
Ссылок: 4
public void StartDialog(UsableObject talker)
{
    Predicate<Dialog> requiredDialog = delegate(Dialog x)
    {
        return x.GetDialogID() == talker.dialogID;
    };
    if(!dialogList.Exists(requiredDialog))
    {
        return;
    }
    activeDialog = dialogList.Find(requiredDialog);
    activeTalker = talker;

    if (activeDialog.IsLocked())
    {
        Debug.Log(DialogConstants.LOG_PREFIX + "Dialog is locked!");
        return;
    }
    var UI = ui.UIController.instance;
    UI.ShowPlayerInterface(false, false, false);
    UI.ShowDialogSystemUI(true);

    var nextPhraseObject = UI.dialogSystemUI.nextPhraseButton;
    UnityAction processPhrase = delegate()
    {
        ProcessNPCPhrase(nextPhraseObject.GetComponent<Phrase>());
    };
    nextPhraseObject.GetComponent<Button>().onClick.AddListener(processPhrase);

    UI.InitializeDialogSystemUI(talker, activeDialog.IsHeroStarting());
    PrepareUI(activeDialog);
    isDialogSystemActive = true;
    Camera.main.GetComponent<CameraScript>().LockCamera();

    var cellGrid = TaskManager.instance.CellGrid;
    cellGrid.CellGridState = new CellGridStateWaitingForInput(cellGrid);
    Debug.Log(DialogConstants.LOG_PREFIX + "Started dialog!");
}
```

Рисунок 4.3 – Імплементація «StartDialog»

За допомогою метода «CompleteQuest» стан завдання переводиться у «виконано» і гравець отримує винагороду.

Метод «TaskCompleted» помічає окремі задачі завдання як виконані і активує наступну задачу.

```
ссылка: 1
protected void CompleteQuest()
{
    if(Owner is CPlayer)
    {
        State = State.Completed;
        _data._rewards.ForEach(
            reward =>
            {
                reward.ApplyReward(Owner as CPlayer);
            });
    }
}
```

Рисунок 4.4 – Імплементація «CompleteQuest»

Клас «CameraScript». У даному класі відсутній статичний екземпляр, так як уся логіка відбувається всередині і не потребує окремого статичного екземпляру. Розглянемо наступні методи класу:

- «CheckBoundsIntersection»;
- «ProcessCameraMovement»;
- «IsPointerOverUIObject».

Метод «CheckBoundsIntersection» перевіряє, чи досягла камера границь ігрового поля під час пересування. Якщо так, відповідний статус відправляється до «ProcessCameraMovement».

«ProcessCameraMovement» опрацьовує отриманий статус і переміщає камеру на необхідні координати сцени.

«IsPointerOverUIObject» дозволяє перевірити, чи пересікається тач користувача із будь-яким об’єктом інтерфейсу.

Від системи штучного інтелекту виберемо клас «EasyLevelAI», успадкований від «CustomAIBase». Розглянемо такі методи:

- «PlayAITurn»;
- «AttackNearestSquad»;

Головний метод «PlayAITurn» запускає переміщення загонів супротивника в сторону загонів героя, які потрібно атакувати. Якщо в радіусі атаки вже є армія героя – наказує атакувати її.

«AttackNearestSquad» шукає найслабший загін героя у радіусі атаки і наказує його атакувати.

```
ссылка: 1
private void ProcessCameraMovement(CameraBoundIntersection intersection)
{
    switch (intersection)
    {
        case CameraBoundIntersection.LEFT_BOUND:
        case CameraBoundIntersection.RIGHT_BOUND:

            Camera.main.transform.position = new Vector3(FirstPos.x, NewCameraPos.y, FirstPos.z);

            break;

        case CameraBoundIntersection.TOP_BOUND:
        case CameraBoundIntersection.DOWN_BOUND:

            Camera.main.transform.position = new Vector3(NewCameraPos.x, FirstPos.y, FirstPos.z);

            break;

        case CameraBoundIntersection.CORNER:


            break;

        case CameraBoundIntersection.NONE:

            Camera.main.transform.position = Origin - Diference;

            break;
    }
}
```

Рисунок 4.5 – Імплементація «ProcessCameraMovement»

```
Ссылок: 4
protected override async Task PlayAITurn()
{
    var timeForExecution = new List<float>();
    Debug.Log("Number of cells: " + currentCellGrid_.Cells.Count.ToString());

    foreach (var currentSquad in currentCellGrid_.Units.FindAll
                                 (
                                    x => x.PlayerNumber == PlayerNumber).OrderByDescending(x => x.AttackFactor)
                                 )
    {
        var enemySquads = currentCellGrid_.Units.FindAll
                          (
                               x => x.PlayerNumber != PlayerNumber
                          );
        if (enemySquads.Count != 0)
        {
            timeForExecution.Add(await MoveSquad(currentSquad, enemySquads));
            timeForExecution[timeForExecution.Count - 1] += await AttackNearestSquad(currentSquad, enemySquads);
        }
    }
    var statistic = new System.Text.StringBuilder();
    statistic.Append("Average time for execution: ");
    statistic.Append(timeForExecution.Average().ToString("0.0000"));
    statistic.Append(". Number of cells: ");
    statistic.Append(currentCellGrid_.Cells.Count);

    Debug.Log(statistic.ToString());
}
```

Рисунок 4.6 – Імплементація «PlayAITurn»

Клас «BattleController». Доступ до використання його інтерфейсів надається через статичний екземпляр. Розглянемо наступні методи:

- «SpawnUnitsOnBattlefield»;
- «BattleWon»;
- «BattleLost».

```
ссылка: 1
private void SpawnUnitsOnBattlefield()
{
    heroArmy.Clear();
    enemyArmy.Clear();

    for (int i = 0; i < BattleSystem.instance.heroArmy.Length; i++)
    {
        if (BattleSystem.instance.heroArmy[i].Unit != null)
        {
            GameObject newUnit = BattleSystem.instance.heroArmy[i].Unit;
            GameObject toSpawn = Instantiate(newUnit);
            BattleUnit spawnedUnit = toSpawn.GetComponent<BattleUnit>();
            spawnedUnit.PlayerNumber = 0;
            spawnedUnit.population = BattleSystem.instance.heroArmy[i].Population;
            spawnedUnit.position = BattleSystem.instance.heroArmy[i].position;
            heroArmy.Add(spawnedUnit);
            toSpawn.transform.position = heroCellCoords[i];
            toSpawn.transform.parent = unitList.transform;
            toSpawn.GetComponentInChildren<TextMesh>().color = new Color(0, 0, 1);
        }
    }

    for (int i = 0; i < BattleSystem.instance.enemyArmy.Length; i++)
    {
        if (BattleSystem.instance.enemyArmy[i].Unit != null)
        {
            GameObject newUnit = BattleSystem.instance.enemyArmy[i].Unit;
            GameObject toSpawn = Instantiate(newUnit);
            BattleUnit spawnedUnit = toSpawn.GetComponent<BattleUnit>();
            spawnedUnit.PlayerNumber = 1;
            spawnedUnit.population = BattleSystem.instance.enemyArmy[i].Population;
            enemyArmy.Add(spawnedUnit);
            toSpawn.transform.position = enemyCellCoords[i];
            toSpawn.transform.parent = unitList.transform;
            toSpawn.GetComponentInChildren<TextMesh>().color = new Color(1, 0, 0);
        }
    }
}
```

Рисунок 4.7 – Імплементація «SpawnUnitsOnBattlefield»

Метод «SpawnUnitsOnBattlefield» виконує одну з головних задач контролера бою – генерує загони героя і супротивника на полі. Для цього вираховуються координати кожної комірки, що буде використана і створюються екземпляри загонів.

«BattleWon» та «BattleLost» сигналізують системі про успішне або провальне завершення відповідно і відштовхуючись від результату показують діалогові вікна.

4.2 Розробка допоміжних засобів

Для тестування продуктивності гри необхідно мати легку можливість відстежувати FPS. Для цього було розроблено невеличкий скрипт, який використовує метод Update(), доступний для Unity класів. Справа у тому, що даний метод викликається двигуном у кінці кожного кадру, що дає можливість вирахувати різницю в часі і представити її у вигляді цілого числа FPS.

```
Сообщение Unity | Ссылок: 0
void Update()
{
    deltaTime += (Time.deltaTime - deltaTime) * 0.1f;
    float fps = 1.0f / deltaTime;
    fpsText.text = Mathf.Ceil(fps).ToString();
}
```

Рисунок 4.8 – Імплементація лічильника FPS

Отримані числа виводяться у простий текстовий об’єкт в кутку екрана.

Висновки до четвертого розділу

У даному розділі розглянуто головні класи кожного зі спроектованих компонентів, а також додаткові інструменти, які було реалізовано. Більшість уваги було приділено методам, що містять велику частину бізнес-логіки.

Метод «SpawnUnitsOnBattlefield» виконує одну з головних задач контролера бою – генерує загони героя і супротивника на полі.

Для тестування продуктивності гри необхідно мати легку можливість відстежувати FPS. Для цього було розроблено невеличкий скрипт, який використовує метод Update(), доступний для Unity класів.

5 ТЕХНІЧНЕ І ФУНКЦІОНАЛЬНЕ ТЕСТУВАННЯ ПРОДУКТУ

Технічне тестування передбачає перевірку працездатності продукту на якомога більшій кількості пристроїв. Його метою є збір інформації про продуктивність ПЗ на різних платформах для подальшого аналізу і виправлення недоліків.

Функціональне тестування проводиться для перевірки реалізованих у продукті фіч. Якщо один з варіантів використання не працює, або працює не так, як потрібно, необхідно повідомити про це відповідних розробників для розслідування проблеми.

5.1 Технічне тестування

У таб. 5.1 наведено характеристики пристроїв, що будуть використані для тестів. Необхідно охватити якомога більший спектр смартфонів за продуктивністю: від бюджетних до преміальних, тому було обрано наступні продукти:

- Samsung Galaxy Note 10, «high-end» пристрій;
- Google Pixel 4, «high-end» пристрій;
- Samsung Galaxy S8, «middle-end» пристрій;
- Meizu m3s, «low-end» пристрій.

Таблиця 5.1 – Характеристики смартфонів для тестування продукту

Назва	**Samsung Galaxy S8**	**Google Pixel 4**	**Samsung Galaxy Note 10**	**Meizu m3s**
SoC	Snapdragon 835 Octa-core (4x2.35 GHz Kryo & 4x1.9 GHz Kryo)	Snapdragon 855 Octa-core (1x2.84 GHz Kryo 485 & 3x2.42 GHz Kryo 485 & 4x1.78 GHz Kryo 485)	Snapdragon 855 Octa-core (1x2.84 GHz Kryo 485 & 3x2.42 GHz Kryo 485 & 4x1.78 GHz Kryo 485)	Mediatek MT6750 Octa-core (4x1.5 GHz Cortex-A53 & 4x1.0 GHz Cortex-A53)
GPU	Adreno 540	Adreno 640	Adreno 640	Mali-T860MP2

RAM	LPDDR4 4 GB	LPDDR4 6 GB	LPDDR4X 8 GB	LPDDR3 2 GB

Продовження таблиці 5.1

Накопичувач	UFS 2.0 64 GB	UFS 2.1 64 GB	UFS 3.0 256 GB	eMMC 5.1 16 GB
Роздільна здатність екрану	FULL HD+ 2280 x 1080 @60Hz	FULL HD+ 2280 x 1080 @90Hz	FULL HD+ 2280 x 1080 @60Hz	HD 1280 x 720 @60Hz
Ємність батареї	3000 mAh	2800 mAh	3500 mAh	3020 mAh

Першим було проведено стрес-тест батареї. Для цього на кожному пристрої гра була запущена протягом 20 хвилин з половиною максимальної яскравості дисплея. На початку і після тесту було зафіксовано графік використання батареї. Узагальнені результати наведено у таб. 5.2.

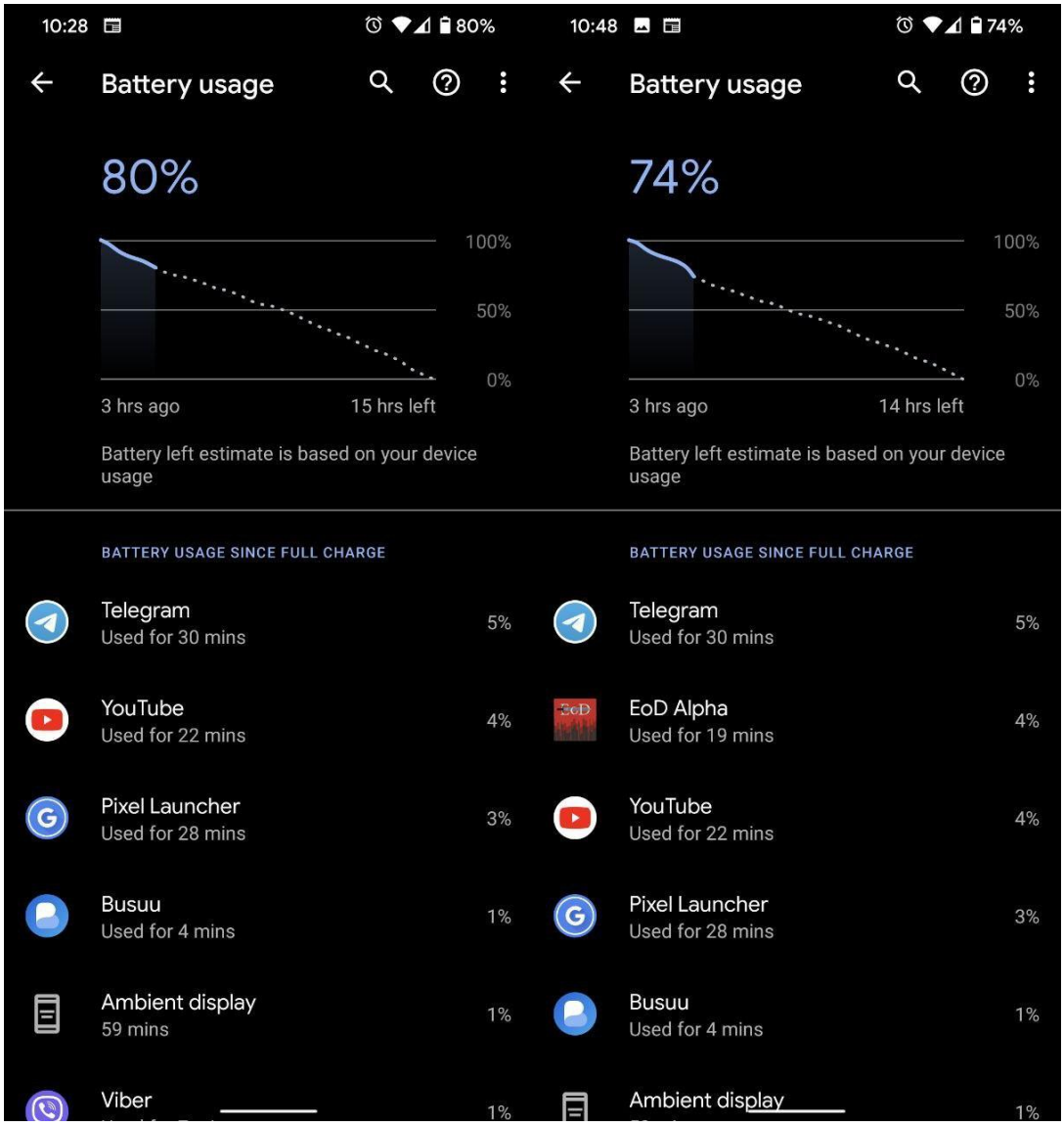


Рисунок 5.1 – Стан батареї Google Pixel 4 до та після тестування

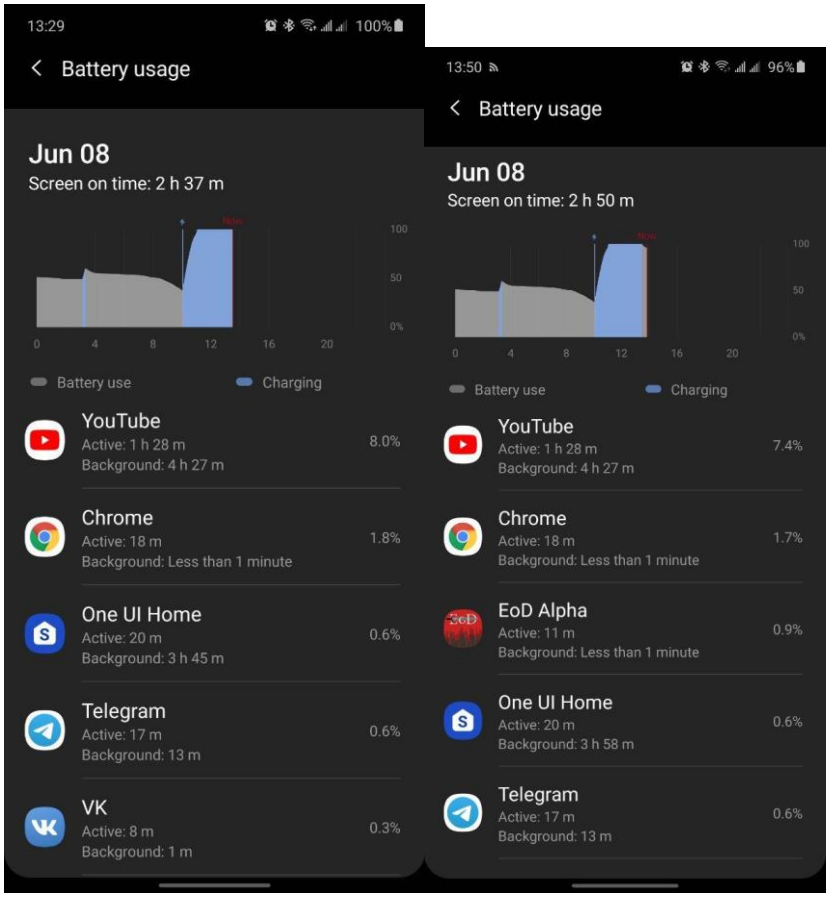


Рисунок 5.2 – Стан батареї Galaxy Note 10 до та після тестування

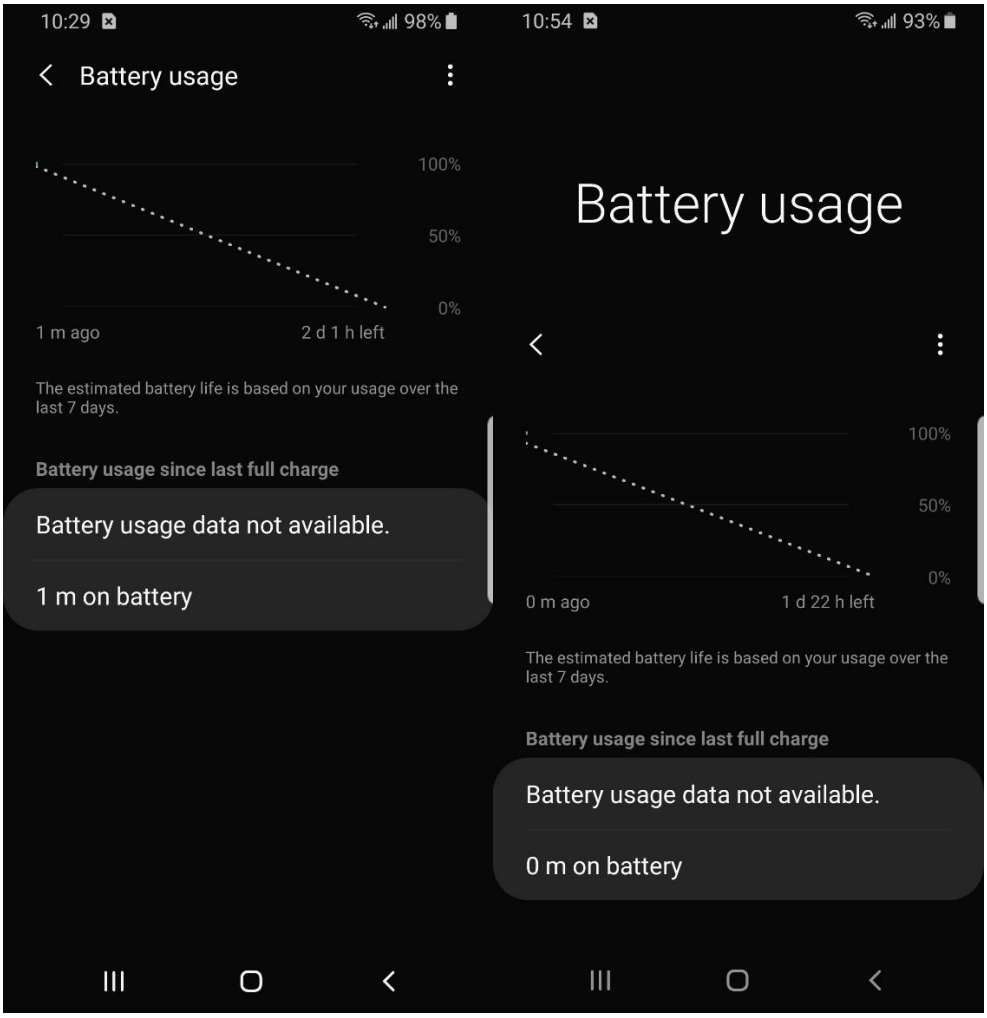


Рисунок 5.3 – Стан батареї Samsung Galaxy S8 до та після тестування

Таблиця 5.2 – Результати стрес-тесту батареї

Пристрій	**Рівень батареї до початку тесту**	**Рівень батареї після завершення тесту**	**Приблизно використано батареї грою**
Samsung Galaxy S8	98%	93%	~4%
Google Pixel 4	80%	74%	~5%
Samsung Galaxy Note 10	100%	96%	~3%
Meizu m3s	100%	98%	~1%

Від використаних грою відсотків віднято один, з ціллю зменшити вплив фонових задач смартфона на результати. Середній відсоток використання батареї

= 3,5%. В умовах тесту це означає, що за годину використання продукту заряд батареї буде знижено на ~11%, що є задовільним результатом.

Наступним етапом технічного тестування є перевірка рівня і стабільності FPS. Для цього буде використано реалізований у грі лічильник FPS, який активується у меню налаштувань. Далі, на усіх смартфонах виконується один і той самий сценарій. Було обрано відрізок ігрової карти з великою кількістю об’єктів.

Першим на тесті буде Meizu m3s, як представник бюджетного сегменту смартфонів.

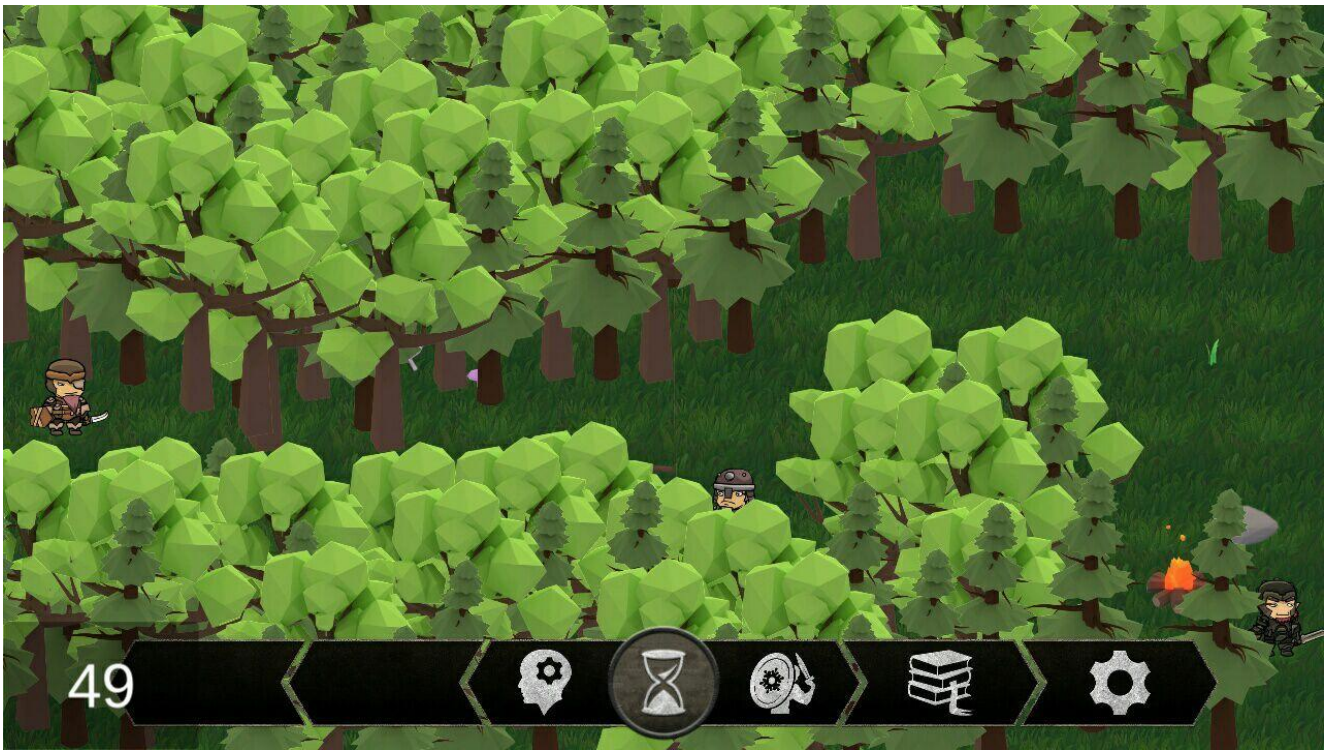


Рисунок 5.4 – Тест FPS на Meizu m3s

Результати тестування:

- мінімальна кількість кадрів за секунду: 45;
- середня кількість кадрів за секунду: 50.

Пристрій показує непогані результати, але отриманої частоти кадрів не достатньо для комфортного ігрового процесу.

Далі протестуємо пристрій середнього рівня. Результати тестування:

- мінімальна кількість кадрів за секунду: 58;
- середня кількість кадрів за секунду: 60.

Більшість часу спостерігається стабільна частота кадрів. Продуктивності гаджету достатньо для забезпечення плавного ігрового процесу.

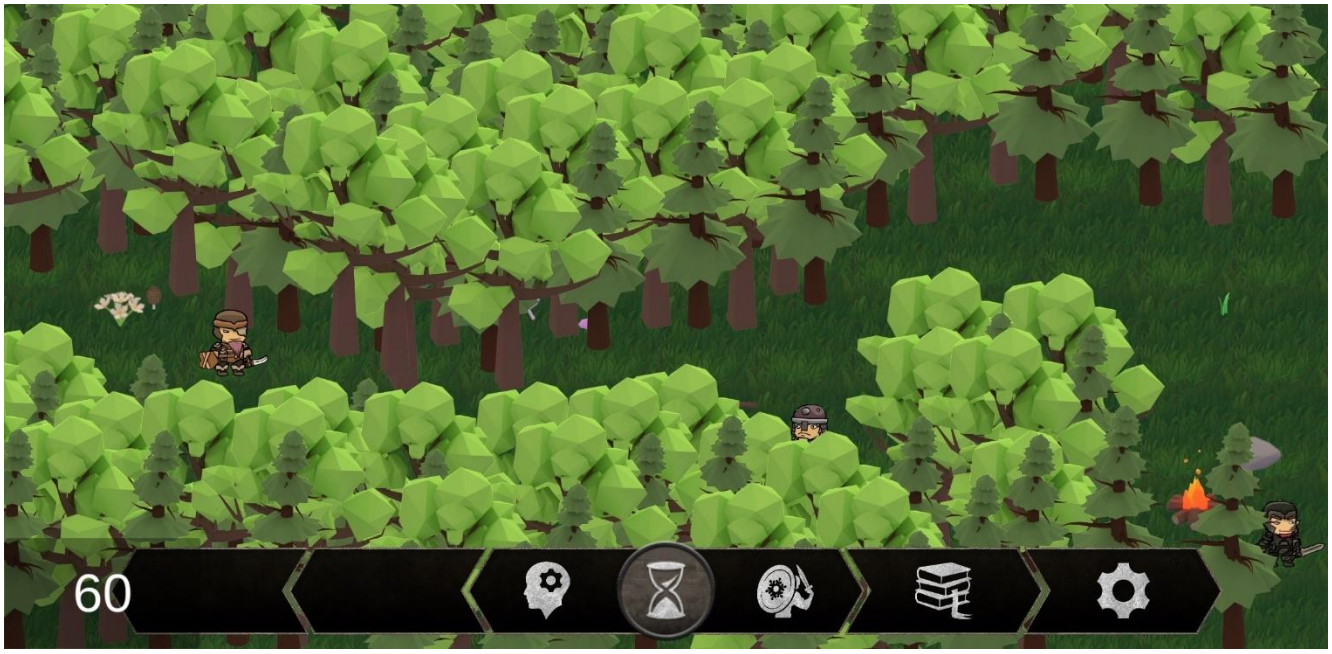


Рисунок 5.5 – Тест FPS на Samsung Galaxy S8

Останнім випробування проходить найпотужніша платформа. Судячи з результатів Galaxy S8 можна з легкістю передбачити стабільний FPS. Результати тестування:

- мінімальна кількість кадрів за секунду: 60;
- середня кількість кадрів за секунду: 60;

Google Pixel 4 покаже приблизно ті ж результати через однаковий SoC, тож у повторному тестуванні немає сенсу.

Рисунок 5.6 – Тест FPS на Samsung Galaxy Note 10

Аналіз результатів технічного тестування показує, що загальний рівень оптимізації продукту більш ніж задовільний. Пристрої середнього рівня починаючи з 2021 року випуску зможуть забезпечити необхідну потужність для комфортного використання продукту.

5.2 Функціональне тестування

Тестування розпочинається відразу після запуску гри. Гравець потрапляє до головного меню, де йому доступні кнопки:

- нова гра;
- продовжити гру;
- загрузити гру;
- налаштування.

Рисунок 5.7 – Головне меню

При натисканні кнопки налаштувань з’являється екран з усіма доступними опціями:

- зміна гучності музики;
- зміна гучності звуків;
- зміна мови;
- включення лічильника FPS.

Виявлено проблему зі слайдером гучності звуку – зміна положень не впливає на гучність.

Рисунок 5.8 – Меню налаштувань

При натисканні на кнопку загрузки гри відкривається екран з усіма доступними гравцю сейв-файлами. Є також можливість видалити будь-який сейв-файл.

Рисунок 5.9 – Список доступних сейв-файлів

Для цього перед користувачем з’явиться діалогове вікно зображене на рис. 5.11.

Рисунок 5.10 – Діалогове вікно видалення сейв-файлу

Після старту нової гри користувач опиняється на сцені вибору раси героя.

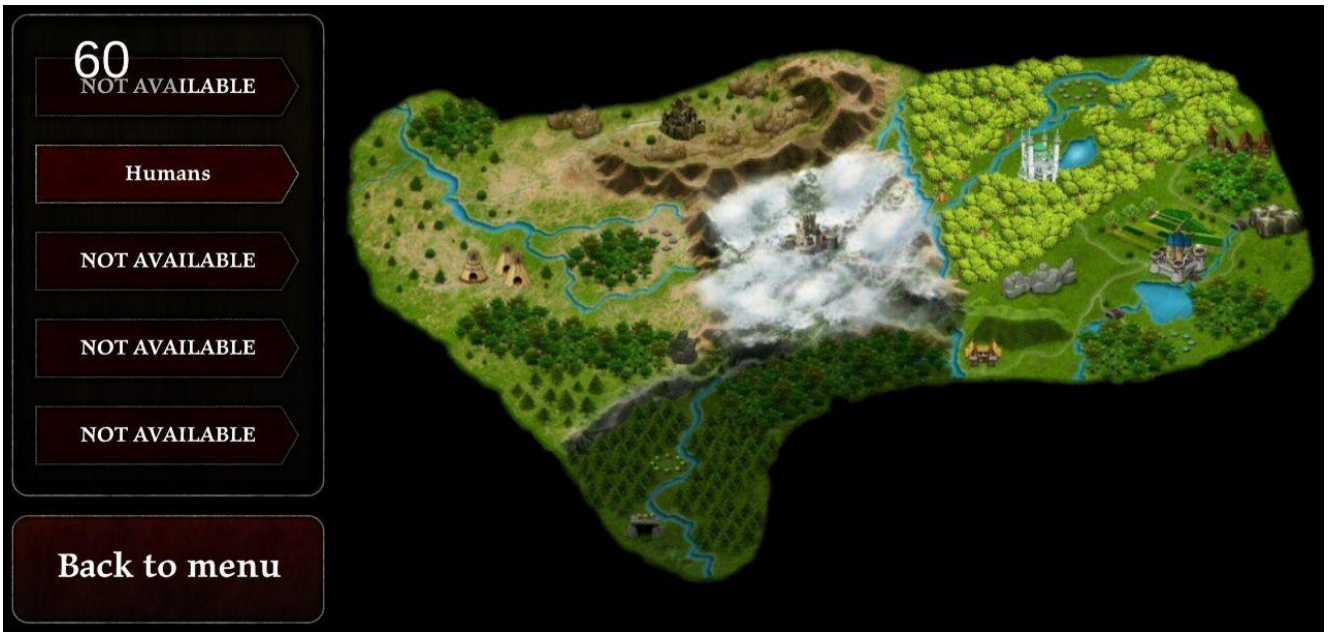


Рисунок 5.11 – Меню вибору раси героя

Після підтвердження раси, гравець вибирає клас майбутнього героя, а також вводить ім’я персонажу за допомогою клавіатури. Цей процес зображено на рис. 5.12 і 5.13 відповідно.

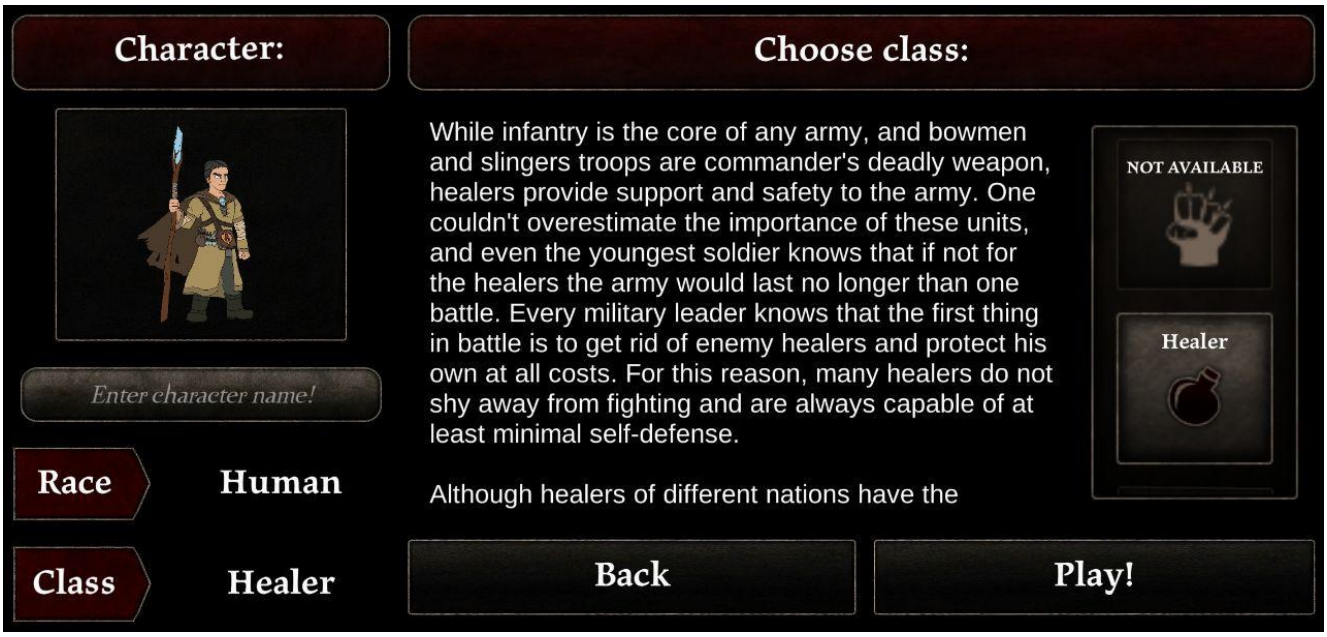


Рисунок 5.12 – Екран вибору класу персонажа

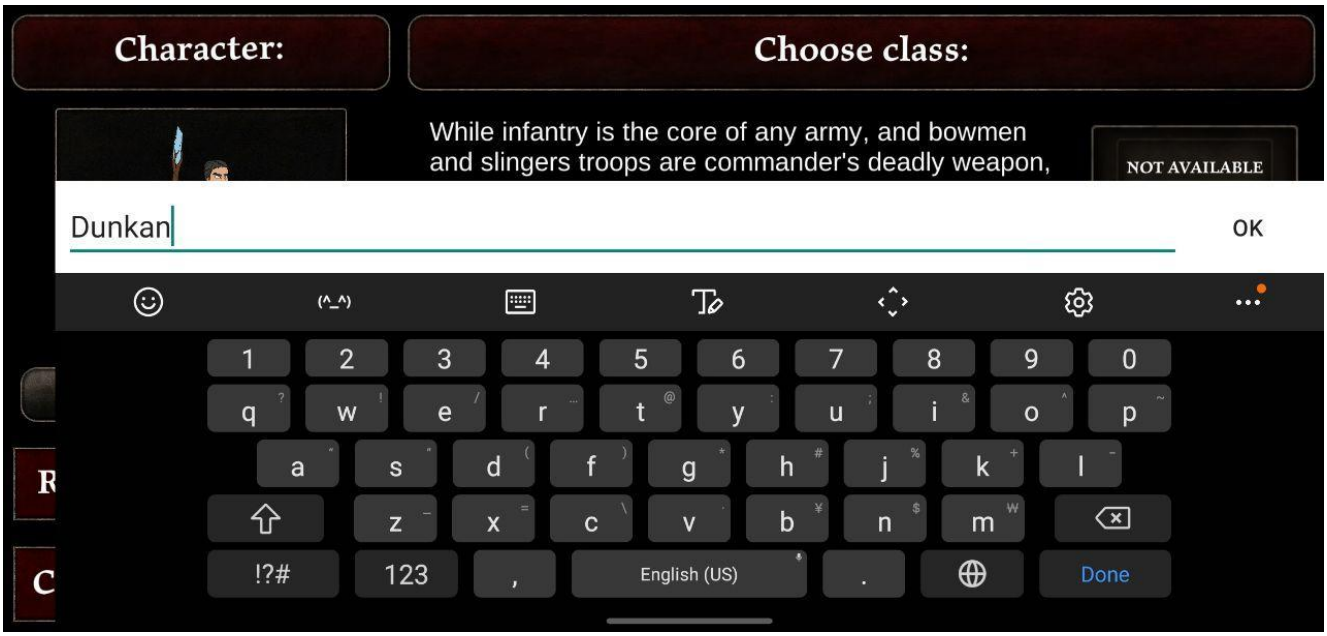


Рисунок 5.13 – Введення імені героя

Далі гравець опиняється на головній ігровій сцені, де для нього повинні бути можливі наступні дії:

- переміщення героя по коміркам;
- початок діалогу з NPC;
- переміщення камери;
- приближення/віддалення камери;

- використання навігаційного меню.

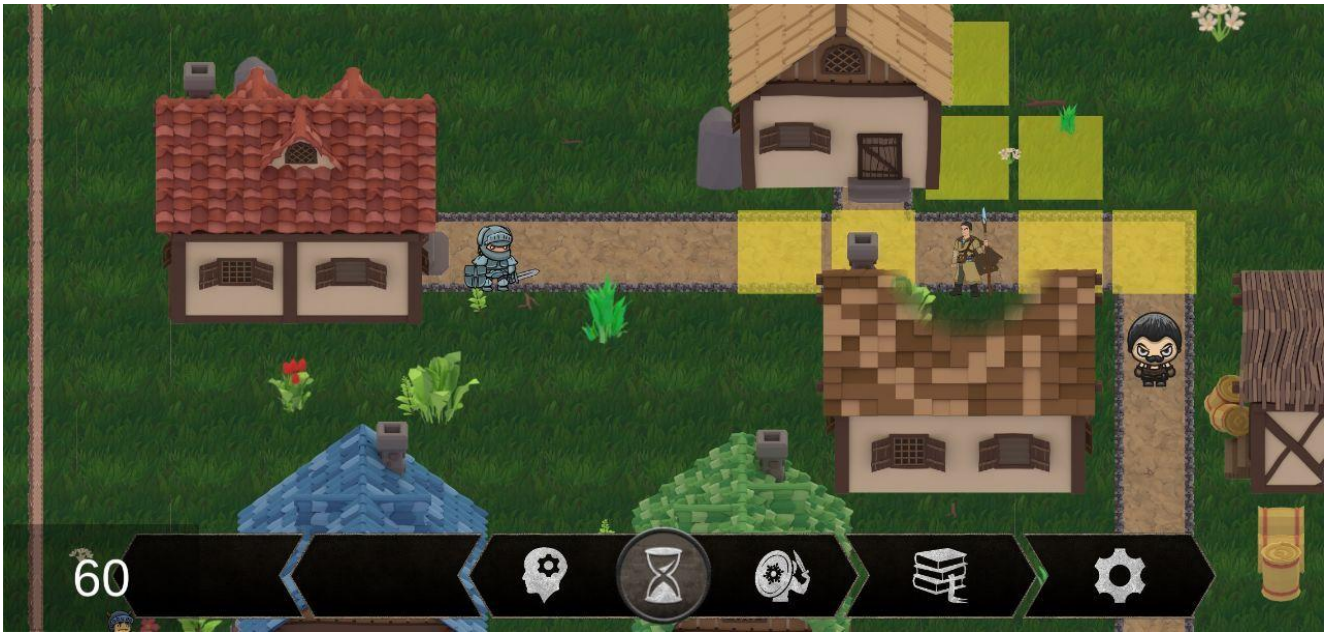


Рисунок 5.14 – Переміщення героя по коміркам

Рисунок 5.15 – Приближення камери

Тестування діалогової системи стартує автоматично після того, як NPC сам активує діалог з головним героєм, у якому жаліється на пропажу сина. Гравцю доступні дві фрази, що приводять до різних результатів.

Рисунок 5.16 – Доступні фрази героя під час діалогу

Під час використання навігаційного меню гравець має змогу:

- відкрити меню армії;
- відкрити меню персонажа;
- відкрити меню завдань;
- відкрити меню додаткових опцій;
- завершити хід.

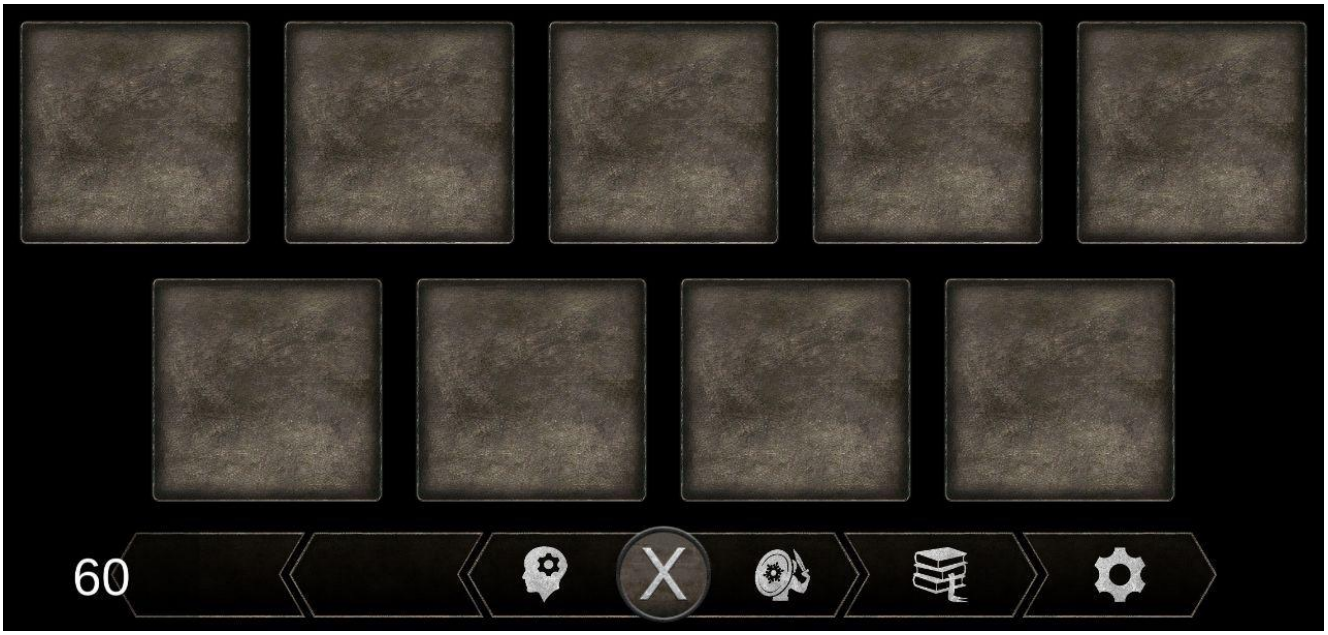


Рисунок 5.17 – Меню армії героя

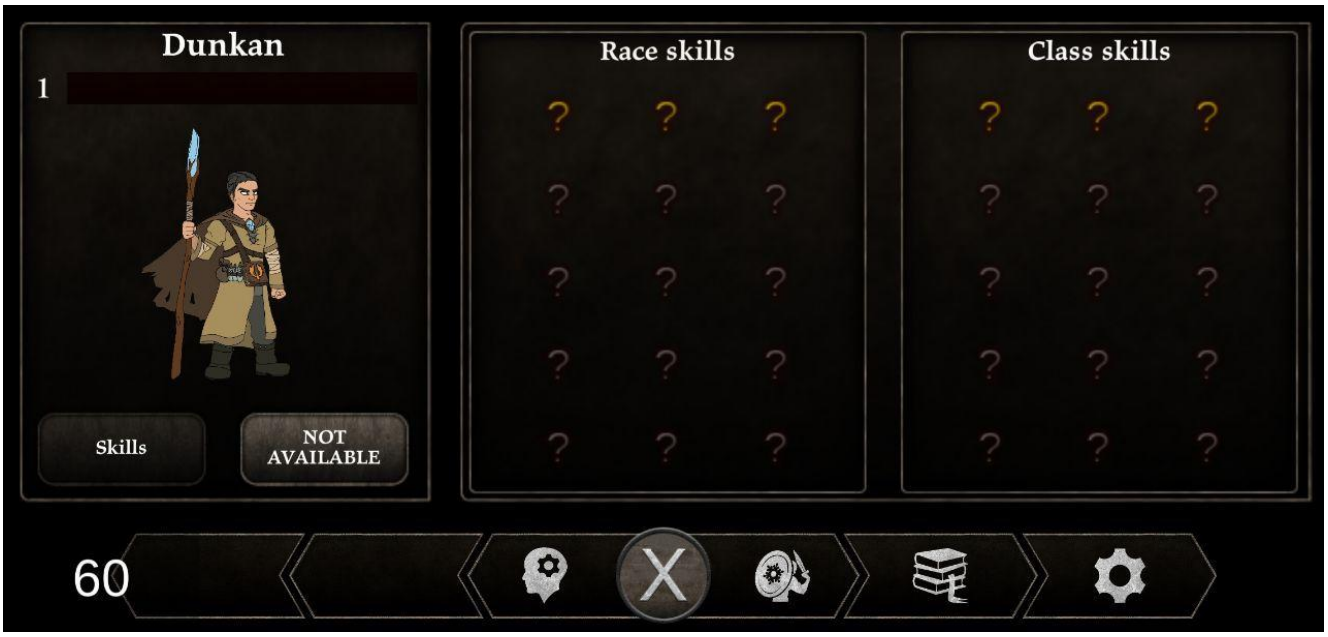


Рисунок 5.18 – Меню персонажа

Рисунок 5.19 – Меню завдань

Після відкриття меню додаткових опцій гравець отримує змогу виконати ще де-які дії знаходячись на глобальній карті, а саме:

- зберегти гру;
- загрузити гру;
- перейти до головного меню.

Рисунок 5.20 – Додаткове меню опцій

Як видно на рис. 5.21 – система повідомлень гравця працює справно.

Рисунок 5.21 – Повідомлення про успішне збереження гри

Більша кількість вимог щодо функціоналу на головній ігровій сцені виконано. Не було помічено багів або критичних помилок, які б заважали роботі способів використання.

Перевіряємо функціонал бойової сцени. Для цього необхідно атакувати бандитів на головній сцені самому, або підійти у радіус 2 комірок і завершити хід.

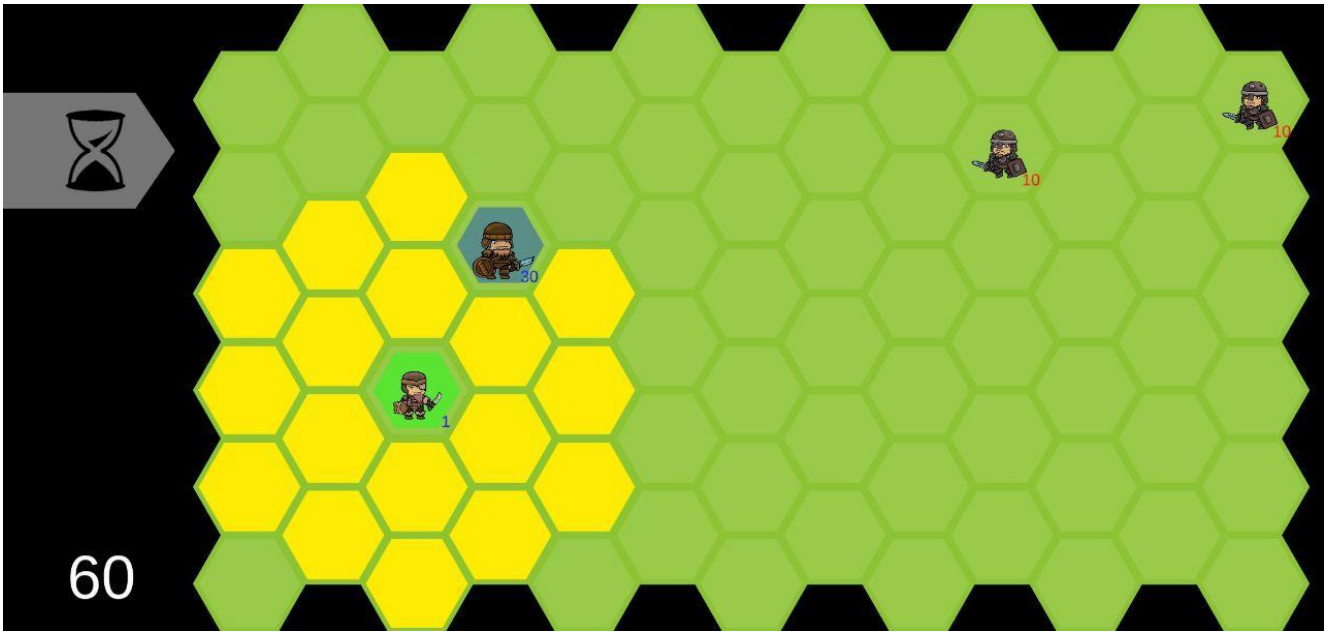


Рисунок 5.22 – Процес бою

Як видно на рис. 5.22 – ініціалізація поля бою пройшла успішно. У гравця є можливість керувати своїми загонами і атакувати армії ворога. У разі успішного завершення поєдинку гравець побачить відповідно діалогове вікно, що повідомляє результат.

Рисунок 5.23 – Діалогове вікно успішного завершення бою

У разі, якщо усі загони герою було знищено, або герой потрапив у бій не маючи армії, гра покаже діалогове вікно про негативний результат бою і запропонує вийти до головного меню, або загрузити один з сейв-файлів.

Рисунок 5.24 – Діалогове вікно провального завершення бою

Під час функціонального тестування було виявлено, що загальні вимоги до продукту виконано і більшість необхідних фіч працює. Але також біли знайдені проблеми:

- слайдер гучності звуку ні на що не впливає;
- немає можливості купляти предмети у NPC;
- відсутня система використання активних навичок героя або армії;

Причиною першої несправності є баг, який необхідно виправити. Щодо останніх двох проблем – вони виникли через недостачу або людських чил, або часу під час планування. Ці системи ще належить спроектувати і реалізувати для гри.

ВИСНОВКИ ТА РЕКОМЕНДАЦІЇ

Розробка ігрового Android-додатку у жанрі покрокової стратегії на базі Unity/C# - це перспективне завдання, яке об'єднує популярність жанру, технічні можливості мобільних платформ та ефективність гейм движка Unity. Такий додаток має потенційно високий попит серед гравців і може стати успішним, як з точки зору популярності серед гравців, так і з комерційного погляду для розробників [3]. В роботі висвітлені ключові аспекти процесу розробки ігрового додатку для мобільних платформ. Розглянуті питання вибору операційних систем, мов програмування та ігрового движка Unity/C#.

1. Було проведено аналіз великої кількості ресурсів, що стало основою для початку розробки додатку. Були розглянуті доступні платформи, операційні системи та їх технічні обмеження.
2. За результатами огляду конкурентних додатків на ринку виявлено, що ігри у жанрі покрокової стратегії продовжують збирати популярність серед геймерів. Це свідчить про те, що додаток "Edge of Devastation" може мати значні шанси на фінансовий успіх.
3. Планування та розробка додатку виявилися важливими завданнями. Під час тестування стало зрозуміло, що не всі системи і функції вдається реалізувати згідно з графіком. Таким чином, в процесі роботи важливо було виявляти гнучкість і готовність до внесення змін.
4. Додаток "Edge of Devastation" відповідає встановленим стандартам для розробки покрокових стратегій, однак він також пропонує інновації, такі як діалогова система та система завдань, щоб забезпечити оригінальність.
5. Отриманий продукт, хоч і потребує деяких доробок перед релізом, вже готовий для використання як демонстраційна версія.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1 Ігри – це мистецтво? // Оксана Волкова. - https://dtf/games/24412-igry-eto-iskusstvo

2 Історія розвитку та еволюція відеоігор у цифрах та картинках. // Олег Міхєєв. - https://hype.tech/@id103/istoriya-razvitiya-i-evolyuciya-videoigr-v-cifrah-i-kartinkah-w4e9feon

3 Середні ціни за CPI (інстал) на iOS та Android у 2022 році // https://useracquisition.ru/ru/srednie-ceny-za-cpi-install-na-ios-i-android-v-2022-godu /

4 Проектування покрокових онлайн ігор. // Є. У. Непомнящих. - https://cyberleninka/article/n/proektirovanie-poshagovyh-onlayn-igr/viewer

5 Порівняння продуктивності ПК та смартфонів, включаючи iPhone 11 // https://habr.com/ua/post/471018/

6 Загальнодоступна багатомовна універсальна інтернет-енциклопедія з вільним контентом // https://ua.wikipedia.org /.

7 What Is Game Development? // https://www.freecodecamp.org/news/what-is-game-development/

8 Швидкі діаграми Ганта + крутий шаблон GoogleDocs // https://medium.com/@Emel.in/швидкі-діаграми-ганта-крутий-шаблон-googledocs-3bd8c00aa6a

9 Unity User Manual (2019.4 LTS) // https://docs.unity3d.com/Manual/index.html

10 Тестування програмного забезпечення - основні поняття та визначення // http://www.protesting.ru/testing/

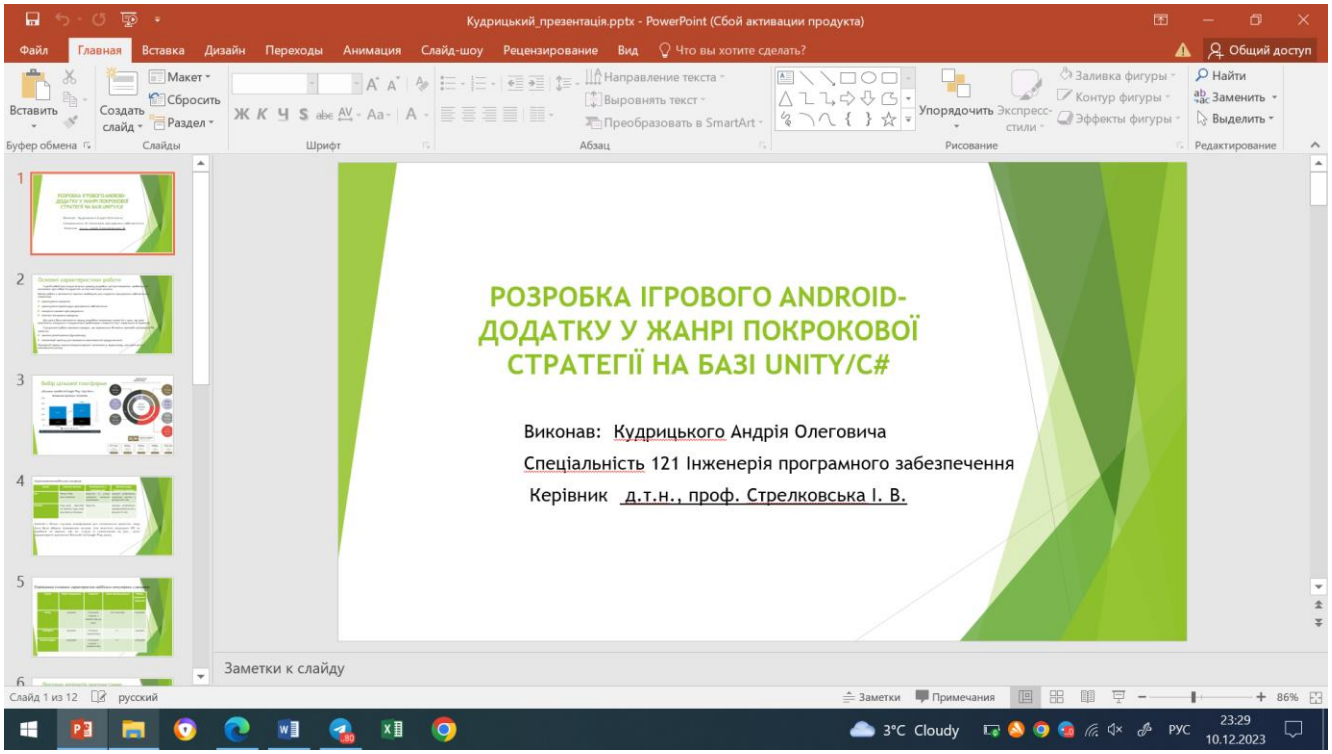
11 З нуля до розробника ігор: як створювати відеоігри, якщо у вас немає досвіду. Частина 1 // medium.com/nuances-of-programming /з-нуля-до-розробника-ігор-як-почати-створювати-відеоігри-якщо-у-вас-ні-досвіду-частина-1-a842af45ead15ead1

12 Сайт з інформацією про створення гри Unity з допомогою Visual Studio [Електронний ресурс] - URL: https://visualstudio.microsoft.com/ru/vs/unity-tools/

13 How to Download and Install Visual Studio for C# [Електронний ресурс] – Режим доступу: https://www.guru99.com/download-install-visual-studio.html

14 Гречко А. В., Захаров Н. В., Фалько М. О. Аналіз динаміки розвитку ринку відеоігор, джерел його фінансування та особливостей монетизації продукції в даній сфері. Ефективна економіка. 2021. № 5. – URL: http://www.economy.nayka.com.ua/?op=1&z=8871 DOI: 10.32702/2307-2105-2021.5.2

ДОДАТОК А
ПЕРЕЛІК КОПІЙ ДЕМОНСТРАЦІЙНОГО МАТЕРІАЛУ



Слайд 1 – Тема

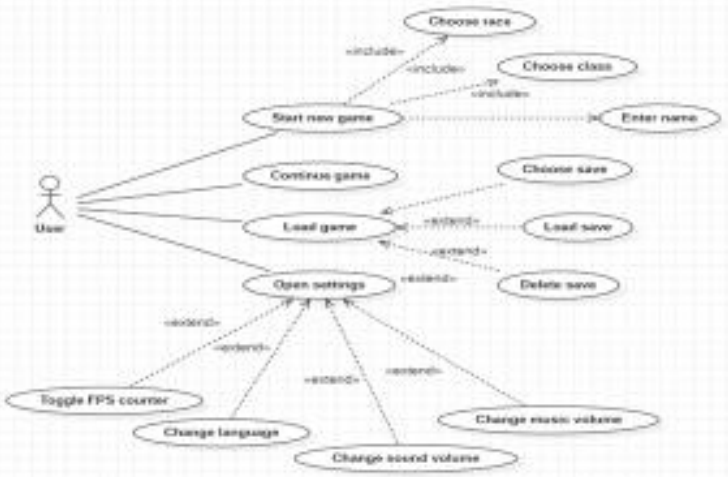
Слайд 3 – Вибір цільової платформи

Слайд 4 – Храктеристики мобільних платформ

)

Слайд 5 – Порівняння головних характеристик найбільш популярних «движків»

Діаграма варіантів використання головного меню



Слайд 6 – Діаграма варіантів використання головного меню

Етапи розробки «Edge of Devastation»

Назва етапу	Опис
Alpha	Загальний процес розробки, який диктує, що більшість функцій слід реалізувати тут. Поєднує ознаки закритого і відкритого альфа тестування, так як час від часу ком'юніті буде отримувати білди із новим функціоналом, що не був доведений до повноцінного робочого стану.
Closed Beta	Найбільша частина закритої бета-версії - це внутрішнє тестування, яке повинно дати нам відгук про всі ігрові системи та їх продуктивність. Тим не менш, деякі фічі можуть бути відкладені до цього етапу.
Open Beta	Ця частина призначена для відкритого тестування та виправлення багів, але все ж ми можемо спланувати тут декілька спірних фіч, які потребують зворотного зв'язку спільноти.
Release	Випуск гри означає відсутність нових функцій, лише підтримку у вигляді виправлення багів. На момент випуску гра повинна з'явитися як у «Google Play», так і у «AppStore».

ДОДАТОК Б
ЛІСТИНГ КОДУ

UIController.cs

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

namespace ui
{
    public class UIController : MonoBehaviour
    {
        public static UIController instance = null;

        public DialogSystemUI dialogSystemUI;
        private PlayerInterface playerInterface;
        private InGameMenu inGameMenu;
        private SideMenu sideMenu;

        void Awake()
        {
            if (instance == null)
            {
                instance = this;
            }
            else if (instance != this)
            {
                Destroy(gameObject);
            }

            dialogSystemUI = new DialogSystemUI();
            playerInterface = new PlayerInterface();
```

```
    }

    void Start()
    {
        inGameMenu = GameObject.Find("Panel_InGameMenuV2").GetComponentInChildren<InGameMenu>();

        dialogSystemUI.ShowDialogSystemUI(false);

        inGameMenu.OpenMenuTab(InGameMenu_Tabs.GAME_SCREEN);
    }

    public void ShowGameMenuPanel(bool flag)
    {
        playerInterface.ShowGameMenuPanel(flag);
    }

    public void ShowGameOptionsPanel(bool flag)
    {
        inGameMenu.ShowOptionsPanel(flag);
    }

    public void ShowDialogSystemUI(bool flag)
    {
        dialogSystemUI.ShowDialogSystemUI(flag);
    }

    public void ShowSideMenuPanel(bool flag)
    {
        //sideMenu.ShowSideMenu(flag);
    }
```

```
public void ShowPlayerInterface(bool gameMenu, bool optionsPanel, bool sideMenu)
{
    ShowGameMenuPanel(gameMenu);
    ShowGameOptionsPanel(optionsPanel);
    ShowSideMenuPanel(sideMenu);
}

public void PrepareDialogSystemUI(HeroReplies replies)
{
    Debug.Log(replies.GetPhraseCount() + "PHRASE COUNT IN " + replies.GetRepliesID());
    dialogSystemUI.SetPhraseButtons(GenerateButtonsForHeroReplies(replies.GetPhraseCount()));

    dialogSystemUI.ArrangePhraseButtons();
    dialogSystemUI.SetButtonsInfo(replies.GetAvailableRepliesList());
    dialogSystemUI.ShowNextPhraseButton(false);
    dialogSystemUI.phrasesPanel.SetActive(true);

    dialogSystemUI.SwitchPortraits(true);
}

public void PrepareDialogSystemUI(Phrase phrase)
{
    dialogSystemUI.ShowNextPhraseButton(true);
    dialogSystemUI.phrasesPanel.SetActive(false);

    dialogSystemUI.nextPhraseButton.GetComponent<Phrase>().SetPhrase(phrase);
    dialogSystemUI.nextPhraseButton.GetComponentInChildren<Text>().text = phrase.GetPhraseText();

    dialogSystemUI.SwitchPortraits(false);
}



public void InitializeDialogSystemUI(UsableObject other, bool isOtherStarting)
{
```

```
dialogSystemUI.SetDialogPortraits(TaskManager.instance.PlayerUnit.GetComponent<SpriteRenderer>(),
                                  other.gameObject.GetComponent<SpriteRenderer>());

        dialogSystemUI.SetDialogNames(other.ObjName);

        if (isOtherStarting == true)
        {
            dialogSystemUI.SwitchPortraits(false);
        }
    }

    public void ClearDialogSystemButtons()
    {
        dialogSystemUI.phraseButtons.Clear();

        foreach(Transform child in dialogSystemUI.phrasesPanelContent)
        {
            Destroy(child.gameObject);
        }
    }

    public void NotificationButtonPressed()
    {
        if ( DialogSystem.instance.isDialogSystemActive == false )
        {
            inGameMenu.OpenMenuTab(UINotification.tabToSwitch);
        }
    }



    public void ShowNotification(UINotification.Content content)
    {
        if (playerInterface.isUINotificationBusy == false)
```

```
    {
        PrepareNotificationContent(content.Title, content.Description, content.Icon);
        StartCoroutine(ShowNotificationPanel());
    }
    else
    {
        UINotification.notificationList.Add(content);
    }
}

private IEnumerator FadeCanvasGroup(CanvasGroup cg, float start, float end, float step = 0.5f)
{
    float timeStarted = Time.time;
    float timeSinceStarted;
    float percentageCompleted;

    while (cg.alpha != end)
    {
        timeSinceStarted = Time.time - timeStarted;
        percentageCompleted = timeSinceStarted / step;

        float currentValue = Mathf.Lerp(start, end, percentageCompleted);

        cg.alpha = currentValue;
        yield return null;
    }

    if (cg.alpha == 0)
    {
        playerInterface.isUINotificationBusy = false;
        checkNotificationList();
    }
}
```

```
private IEnumerator ShowNotificationPanel()
{
    playerInterface.isUINotificationBusy = true;
    uiNotificationFadeIn();
    yield return new WaitForSeconds(3f);
    uiNotificationFadeOut();
}

private void uiNotificationFadeIn()
{
    playerInterface.uiNotification.transform.Find("Button_OpenRelated").gameObject.SetActive(true);
    CanvasGroup cg = playerInterface.uiNotification.GetComponent<CanvasGroup>();
    cg.blocksRaycasts = true;
    StartCoroutine(FadeCanvasGroup(cg, cg.alpha, 1));
}

private void uiNotificationFadeOut()
{
    playerInterface.uiNotification.transform.Find("Button_OpenRelated").gameObject.SetActive(false);
    CanvasGroup cg = playerInterface.uiNotification.GetComponent<CanvasGroup>();
    cg.blocksRaycasts = false;
    StartCoroutine(FadeCanvasGroup(cg, cg.alpha, 0));
}



private void checkNotificationList()
{
    if (UINotification.notificationList.Count > 0)
    {
        ShowNotification(UINotification.notificationList[0]);
        UINotification.notificationList.RemoveAt(0);
    }
}
```

```
        private void PrepareNotificationContent(string title, string content, Sprite sprite)
        {
            playerInterface.uiNotification.transform.Find("Text_NotificationTitle").
                GetComponent<Text>().text = title;
            playerInterface.uiNotification.transform.Find("Text_NotificationContent").
                GetComponent<Text>().text = content;
            playerInterface.uiNotification.transform.Find("Image_NotificationIcon").
                GetComponent<Image>().sprite = sprite;
        }

        private List<GameObject> GenerateButtonsForHeroReplies(ushort phraseCount)
        {
            List<GameObject> phraseButtons = new List<GameObject>();

            if (phraseCount > 0)
            {
                for (int i = 0; i < phraseCount; i++)
                {
                    GameObject phraseButton = Instantiate(Resources.Load("Prefabs/Button_Phrase")) as GameObject;
                    phraseButtons.Add(phraseButton);
                }
            }

            return phraseButtons;
        }
    }
}
```

DialogSystem.cs

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

```
using UnityEngine.Events;

public class DialogSystem : MonoBehaviour {

    public static DialogSystem instance = null;

    private tools.DialogsXMLParser dialogParser;
    private List<Dialog> dialogList = new List<Dialog>();

    public Dialog activeDialog = new Dialog();
    public UsableObject activeTalker = new UsableObject();
    public bool isDialogSystemActive = false;

    void Awake()
    {
        if (instance == null)
        {
            instance = this;
        }
        else if (instance != this)
        {
            Destroy(gameObject);
        }
        DontDestroyOnLoad(gameObject);

        switch (Lean.Localization.LeanLocalization.CurrentLanguage)
        {
            case "English":

                dialogParser = new tools.DialogsXMLParser("Dialogs/Human_Start_01/English/DIALOGS");

                break;

            case "Russian":
```

```
            dialogParser = new tools.DialogsXMLParser("Dialogs/Human_Start_01/Russian/DIALOGS");

            break;
        }

        dialogParser.ReadDialogsFromXML(dialogList);

        var arr = dialogList.Find(x => x.GetDialogID() == "HUMAN_START_BLACKSMITH").heroReplies.Find(x => x.GetRepliesID() == "HR-HSB-0").GetRepliesList();

        Debug.Log(DialogConstants.LOG_PREFIX + "Initialization complete!");
    }


    public void StartDialog(UsableObject talker)
    {
        Predicate<Dialog> requiredDialog = delegate(Dialog x)
        {
            return x.GetDialogID() == talker.dialogID;
        };
        if(!dialogList.Exists(requiredDialog))
        {
            return;
        }
        activeDialog = dialogList.Find(requiredDialog);
        activeTalker = talker;

        if (activeDialog.IsLocked())
        {
            Debug.Log(DialogConstants.LOG_PREFIX + "Dialog is locked!");
            return;
        }
```

```
        var UI = ui.UIController.instance;
        UI.ShowPlayerInterface(false, false, false);
        UI.ShowDialogSystemUI(true);

        var nextPhraseObject = UI.dialogSystemUI.nextPhraseButton;
        UnityAction processPhrase = delegate()
        {
            ProcessNPCPhrase(nextPhraseObject.GetComponent<Phrase>());
        };
        nextPhraseObject.GetComponent<Button>().onClick.AddListener(processPhrase);

        UI.InitializeDialogSystemUI(talker, activeDialog.IsHeroStarting());
        PrepareUI(activeDialog);
        isDialogSystemActive = true;
        Camera.main.GetComponent<CameraScript>().LockCamera();

        var cellGrid = TaskManager.instance.CellGrid;
        cellGrid.CellGridState = new CellGridStateWaitingForInput(cellGrid);
        Debug.Log(DialogConstants.LOG_PREFIX + "Started dialog!");
    }



    public void ProcessNPCPhrase(Phrase phrase)
    {
        if (CheckConstantPhraseTypes(phrase) == false)
        {
            ui.UIController.instance.ClearDialogSystemButtons();

ui.UIController.instance.PrepareDialogSystemUI(activeDialog.GetHeroRepliesByID(phrase.GetNextPhraseID()
));
        }

        Debug.Log(DialogConstants.LOG_PREFIX + "Processed NPC phrase");
    }
```

```
    public void ProcessHeroPhrase(Phrase phrase)
    {
        if (CheckConstantPhraseTypes(phrase) == false)
        {

ui.UIController.instance.PrepareDialogSystemUI(activeDialog.GetPhraseByID(phrase.GetNextPhraseID()));
        }

        Debug.Log(DialogConstants.LOG_PREFIX + "Processed Hero phrase");
    }



    public void EndDialog()
    {

ui.UIController.instance.dialogSystemUI.nextPhraseButton.GetComponent<Button>().onClick.RemoveAllListe
ners();
        ui.UIController.instance.ClearDialogSystemButtons();

        ui.UIController.instance.ShowDialogSystemUI(false);
        ui.UIController.instance.ShowPlayerInterface(true, false, true);

        Camera.main.GetComponent<CameraScript>().UnlockCamera();

        CPlayer.instance.UpdateQuests(activeTalker._id, 1);

        isDialogSystemActive = false;

        Debug.Log(DialogConstants.LOG_PREFIX + "Ended dialog!");
    }



    private bool CheckConstantPhraseTypes(Phrase phrase)
```

```
{
    bool checkSuccess = false;

    if (phrase.GetUnlockablesID() != DialogConstants.NoneConstID)
    {
        activeDialog.ProcessDialogUnlockables(phrase.GetUnlockablesID());
    }

    if (phrase.GetLockablesID() != DialogConstants.NoneConstID)
    {
        activeDialog.ProcessDialogLockables(phrase.GetLockablesID());
    }

    if (phrase.GetAction() != DialogConstants.NoneConstID)
    {
        CheckConstantActionTypes(phrase.GetAction());
    }

    if (phrase.GetNextPhraseID() == DialogConstants.EndDialogConstID)
    {
        checkSuccess = true;
        EndDialog();
    }

    return checkSuccess;
}


private void CheckConstantActionTypes(string actionID)
{
    activeDialog.ProcessDialogActions(actionID);
}
```

```
    private void PrepareUI(Dialog activedialog)
    {
        var UI = ui.UIController.instance;
        if (activeDialog.IsHeroStarting() == true)
        {
            var heroReplies = activeDialog.GetHeroRepliesByID(activeDialog.GetStartingPhraseID());
            UI.PrepareDialogSystemUI(heroReplies);
        }
        else
        {
            var startingPhrase = activeDialog.GetPhraseByID(activeDialog.GetStartingPhraseID());
            UI.PrepareDialogSystemUI(startingPhrase);
        }
    }


    public void UnlockDialogByID(string id)
    {
        dialogList.Find(x => x.GetDialogID() == id).SetLockedStatus(false);
    }


    public void LockDialogByID(string id)
    {
        Debug.Log("LOCKING "+ id);
        dialogList.Find(x => x.GetDialogID() == id).SetLockedStatus(true);
    }


    public List<Dialog> GetDialogList()
    {
        return dialogList;
    }
}
```

SaveSystem.cs

```
using UnityEngine;
using UnityEngine.SceneManagement;
using System;
using System.IO;
using System.Collections;
using System.Collections.Generic;
using System.Runtime.Serialization.Formatters.Binary;

public class SaveSystem : MonoBehaviour
{
    private void Awake()
    {
        if(null == instance)
        {
            instance = this;
        }
        else if(this != instance)
        {
            Destroy(gameObject);
        }
        DontDestroyOnLoad(gameObject);
    }



    private void Start()
    {
        saves = new List<Save.MetaData>();
        locationStates = new Dictionary<string, GameState>();
        GetSaves();
    }
```

```
///
// <summary>
// Saves location state to memory. Has to be called before loading new scene.
// </summary>
///
public void SaveGameToMemory()
{
    Loader.playerLocation = SceneManager.GetActiveScene().name;
    var currentGameState = new GameState();
    locationStates[currentGameState.locationName] = currentGameState;
}


public void UpdatePlayersSkills()
{
    if(locationStates.ContainsKey(Loader.playerLocation))
    {
        locationStates[Loader.playerLocation].UpdateSkills();
    }
}


public void UpdatePlayersArmy()
{
    if(locationStates.ContainsKey(Loader.playerLocation))
    {
        locationStates[Loader.playerLocation].UpdateArmy();
    }
}


public void UpdateLocationUnits(List<Unit> newUnits)
{
    if(locationStates.ContainsKey(Loader.playerLocation))
```

```
    {
        locationStates[Loader.playerLocation].UpdateLocationUnits(newUnits);
    }
}


public void SaveGameToFileSystem()
{
    StartCoroutine(TakeScreenshot());
}


public void LoadGameFromMemory()
{
    StartCoroutine(Load(locationStates[Loader.playerLocation]));
}


public void LoadGameFromFileSystem(string saveName)
{
    var save = GetSave(saveName);
    StartCoroutine(Load(save.game));
}


public void DeleteSave(string saveName)
{
    var filePath = Application.persistentDataPath + "/" + saveName;
    File.Delete(filePath);
    var deletedSave = saves.Find(save => save.saveName == saveName);
    saves.Remove(deletedSave);
}
```

```
public void LoadLatestSave()
{
    var save = GetSave(saves[0].saveName);
    StartCoroutine(Load(save.game));
}



private void GetSaves()
{
    const string savePattern = "*^*^??.??.????^??@??@??.eod";
    saves.Clear();
    var saveFiles = Directory.EnumerateFiles(Application.persistentDataPath, savePattern);
    foreach(var filePath in saveFiles)
    {
        char[] pathSeparators = {'/', '\\'};
        var filePathElements = filePath.Split(pathSeparators);
        var file = filePathElements[filePathElements.Length - 1];
        var save = GetSave(file);
        if(save != null && save.data != null)
        {
            saves.Add(save.data);
        }
    }
    SortSaves();
}



private Save GetSave(string saveName)
{
    var formatter = new BinaryFormatter();
    var filePath = Application.persistentDataPath + "/" + saveName;
    var file = File.Open(filePath, FileMode.Open);
    var save = (Save)formatter.Deserialize(file);
    file.Close();
```

```
    return save;
}


private IEnumerator TakeScreenshot()
{
    yield return new WaitForEndOfFrame();
    var screenshot = new Texture2D(Screen.width, Screen.height, TextureFormat.RGB24, true);
    screenshot.ReadPixels(new Rect(0, 0, Screen.width, Screen.height), 0, 0);
    screenshot.Apply();
    WriteSaveToFileSystem(screenshot);
}


private void WriteSaveToFileSystem(Texture2D screenshot)
{
    var save = new Save(screenshot);

    var formatter = new BinaryFormatter();
    if(!Directory.Exists(Application.persistentDataPath))
    {
        Directory.CreateDirectory(Application.persistentDataPath);
    }
    var file = File.Create(Application.persistentDataPath + "/" + save.data.saveName);
    formatter.Serialize(file, save);
    UnityEngine.Object.Destroy(screenshot);
    file.Close();
    saves.Add(save.data);
    SortSaves();
    var notification = UINotification.CreateGameSavedNotification(save.data.DisplayedName);
    ui.UIController.instance.ShowNotification(notification);
}
```

```
private IEnumerator Load(GameState save)
{
    LevelManager.instance.LoadScene(save.locationName);
    while(LevelManager.instance.isSceneLoading)
    {
        yield return null;
    }
    save.Load();
    Loader.locationLoaded = true;
    Loader.gameStarted = true;
    if (GameLoaded != null)
    {
        GameLoaded.Invoke(this, new EventArgs());
    }
}



private void SortSaves()
{
    saves.Sort((first, second)=>-first.saveTime.CompareTo(second.saveTime));
}

public event EventHandler GameLoaded;

private Dictionary<string, GameState> locationStates;
private List<Save.MetaData> saves;
public static SaveSystem instance;

public List<Save.MetaData> Saves
{
    get
    {
        return saves;
    }
```

```
    }
    public Dictionary<string, GameState> LocationState
    {
        get => locationStates;
        private set => locationStates = value;
    }
}
```