Міністерство освіти і науки України
Національний технічний університет
«Дніпровська політехніка»

Інститут електроенергетики
(інститут)

Факультет інформаційних технологій
(факультет)

Кафедра Програмного забезпечення комп'ютерних систем
(повна назва)

ПОЯСНЮВАЛЬНА ЗАПИСКА
кваліфікаційної роботи ступеня

бакалавра
(назва освітньо-кваліфікаційного рівня)

студента *Тріска Антона Євгеновича*
(ПІБ)

академічної групи *122-18-2*
(шифр)

спеціальності *122 Комп'ютерні науки*
(код і назва спеціальності)

освітньої програми *Комп'ютерні науки*
(назва освітньої програми)

на тему: *Розробка мультиплатформеного ігрового додатку за допомогою Unity*

Керівники	**Прізвище, ініціали**	**Оцінка за шкалою**		**Підпис**
		рейтинговою	**інституційною**	
кваліфікаційної роботи	*доц. Приходченко С.Д.*			
розділів:				
спеціальний	*доц. Приходченко С.Д.*			
економічний	*доц. Касьяненко Л.В.*			
Рецензент				
Нормоконтролер	*доц. Гуліна І.Г.*			

Дніпро
2022

Міністерство освіти і науки України
НТУ «Дніпровська політехніка»

ЗАТВЕРДЖЕНО:
завідувач кафедри
програмного забезпечення комп'ютерних систем
(повна назва)

_____ І.М. Удовик
(підпис) (прізвище, ініціали)

« » _____ 2022 року

ЗАВДАННЯ
на кваліфікаційну роботу

бакалавра
(назва освітньо-кваліфікаційного рівня)

студента *122-18-2* (група) *Тріска Антона Євгеновича* (прізвище та ініціали)

тема кваліфікаційної роботи *Розробка мультиплатформеного ігрового додатку за допомогою Unity*

затверджена наказом ректора НТУ «ДП» від 18.05.2022 № 268-с

Розділ	Зміст виконання	Термін виконання
Спеціальний	***На основі матеріалів проєктно-технологічної практики та інших науково-технічних джерел провести аналіз стану рішення проблеми та постановку задачі. Обґрунтувати вибір та здійснити реалізацію методів вирішення проблеми***	***13.05.2022 р.***
Економічний	***Провести розрахунок трудомісткості розробки програмного забезпечення, витрат на створення ПЗ й тривалості його розробки***	***27.05.2022 р.***

Завдання видав _____ *доц. Приходченко С.Д.*
(підпис) (посада, прізвище, ініціали)

Завдання прийняв до виконання _____ *Тріско А.Є.*
(підпис) (прізвище, ініціали)

Дата видачі завдання: *14.01.2022 р.*

Термін подання кваліфікаційної роботи до ЕК: *13.06.2022 р.*

РЕФЕРАТ

Пояснювальна записка: 77 с., 15 рис., 3 дод., 20 джерел.

Об'єкт розробки: мультиплатформений ігровий додаток розроблений за допомогою ігрового двигуна Unity.

Мета кваліфікаційної роботи: розробити мультиплатформений ігровий додаток розроблений за допомогою ігрового двигуна Unity.

У вступі виконується аналіз сучасного стану проблеми, уточнюється постановка завдання, мета кваліфікаційної роботи та галузь її застосування, обґрунтовується актуальність теми.

У першому розділі проводиться дослідження предметної області та існуючих рішень, визначається актуальність завдання та призначення розробки, розроблюється постановка завдання.

У другому розділі обирається платформа для розробки, виконується проектування програми і її розробка, наводиться опис алгоритму і структури функціонування системи, визначаються вхідні і вихідні дані, наводяться характеристики складу параметрів технічних засобів, описується робота програми.

В економічному розділі визначається трудомісткість розробленого програмного продукту, проводиться підрахунок вартості роботи по створенню застосунку та розраховується час на його створення.

Практичне значення полягає у розробці ігрового додатку, що можна запускати на одній з декількох платформ та дозволяє користувачам використовувати його як у рекреаційних цілях і соціалізації при грі з іншою людиною, так і задля розвитку тактичного та стратегічного мислення.

Актуальність програмного продукту визначається значним ростом популярності комп’ютерних та мобільних ігор та загальним стабільним та потужним зростанням ігрової індустрії.

Список ключових слів: ГРА, ІГРОВИЙ ДОДАТОК, МУЛЬТИПЛАТФОРМА, ПЛАТФОРМА, UNITY, ІГРОВИЙ ДВИГУН, КОРИСТУВАЧ, ІНФОРМАЦІЙНА СИСТЕМА.

ABSTRACT

Explanatory note: 77 p., 15 figs., 3 appx., 20 sources.

Object of development:multiplatform game application developed using Unity game engine.

Purpose of the qualification work: Create a multiplatform game application using Unity game engine.

In the introduction it is analyzed the current state of the problem, clarified the problem, the purpose of the qualification work and the scope of its application, substantiated the relevance of the topic.

In the first section the research of the subject area and existing decisions is carried out, the urgency of the task and purpose of development is defined, the statement of the task is developed.

In the second section the platform for development is chosen, the program design and its development is carried out, the description of algorithm and structure of functioning of system is given, input and output data are defined, characteristics of structure of parameters of technical means are given, work of the program is described.

In the economic section it is determined the complexity of the developed software product, calculated the cost of work to create an application and calculated the time to create it.

Of practical importance is the development of a game application that can be run on one of several platforms and allows users to use it for recreational purposes and socialization when playing with another person, and to develop tactical and strategic thinking.

The relevance of the software product is determined by the significant growth in the popularity of computer and mobile games and the overall stable and strong growth of the gaming industry.

Keywords: GAME, GAME APPLICATION, MULTIPLATFORM, PLATFORM, GAME ENGINE, USER, INFORMATION SYSTEM.

.

ЗМІСТ

СПИСОК УМОВНИХ ПОЗНАЧЕНЬ

UNITY – багатоплатформовий інструмент для розробки відеоігор і застосунків;

C# – мова програмування;

JSON – текстовий формат обміну даними;

EXE – розширення виконуваного файлу, що застосовується в системах Microsoft Windows;

APK – формат архівних виконуваних файлів-програм для Android;

ОС – операційна система;

ПК – персональний комп’ютер;

GPU – графічний процесор;

CPU – центральний процесор.

ВСТУП

Відеоігрова індустрія є однією з найбільш передових, але при цьому і однією з найбільш мінливих гілок сфери інформаційних технологій. Кожна гра, що створюється на даний момент хоч і може виглядати подібно до чогось вже баченого стороннім глядачем, насправді, у самій своїй суті та методах реалізації буде справді унікальним продуктом, створеним на перехресті програмування та дизайну.

Відеоігри знаходяться на межі між мистецтвом і технологією, що в деякому сенсі було навряд чи можливо лише кілька десятиліть тому. Об'єднайте технологічний прогрес і той факт, що гра може бути, по суті, чим завгодно - від двовимірної головоломки на iPhone до живучої своїм життям RPG з відкритим світом і супер реалістичною графікою, - і тоді вас не шокуватиме відсутність єдиних стандартів створення комп'ютерних ігор.

Також індустрія з самого свого заснування купкою гіків ентузіастів з вищих навчальних закладів США продовжує стояти на чолі прогресу на полі комп’ютерних технологій, створюючи та розвиваючи справді інноваційні рішення як на рівні програмного забезпечення, так і на апаратному рівні, одночасно сприяючи розвитку вже існуючих технологій (мініатюризація та збільшення потужностей) та створюючи дещо абсолютно нове, що згодом виходить далеко поза межі індустрії відеоігор (віртуальна та аугоментована реальності).

Так, наприклад, сучасні персональні комп'ютери зобов'язані багатьом досягненням та інноваціям ігрової індустрії: звукові карти, графічні карти та прискорювачі тривимірної графіки, швидші процесори та спеціалізовані співпроцесори, такі як PhysX, — ось лише деякі з найпомітніших покращень. Звукові карти наприклад, спочатку були розроблені для додавання звуку цифрової якості в ігри, і пізніше вони були поліпшені для музичної індустрії. Графічні карти спочатку розроблялися задля забезпечення більшої кількості

кольорів на екрані, а потім для підтримки графічних інтерфейсів (GUI) та ігор. Це викликало необхідність більш високих дозволів і 3D-прискорення.

Тому завданням кваліфікаційної роботи було обрано “Розробка мультиплатформеного ігрового додатку за допомогою Unity”. Завдання даної кваліфікаційної роботи та об'єкт його діяльності безпосередньо пов'язані за напрямом підготовки «Комп’ютерні науки» та відповідає узагальненій тематиці кваліфікаційних робіт.

Метою даної кваліфікаційної роботи є поглиблене вивчення індустрії відеоігор та реалізація ігрового, додатку, що буде допомагати з покращенням рівня стресу, розвитку логічного та стратегічного мислення та може бути використаний у рекреаційних цілях.

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1. Загальні відомості з предметної галузі

Індустрія комп'ютерних ігор - сектор економіки, пов'язаний з розробкою, просуванням і продажем комп'ютерних ігор. До неї входить велика кількість спеціальностей, за якими працюють десятки тисяч людей по всьому світу.

Історія ігрової індустрії розпочалася у 1971 році із запуску аркадної гри Computer Space. Наступного року компанія Atari випустила першу комерційно успішну відеогру Pong. Оригінальна версія на аркадних автоматах було розпродано у кількості 19 тисяч штук. Протягом того ж року на ринок було випущено одну з перших в історії домашню ігрову консоль — Magnavox Odyssey. Поступово ринок як аркадних, так і домашніх ігор переповнився одноманітними клонами Pong, що призвело до кризи ринку відеоігор 1977 року. Ця криза була подолана вже наступного року, коли успіху досягла гра Space Invaders від Taito Corporation, що ознаменувало відновлення ринку відеоігор і зумовило наступ через деякий час так званої «золотої ери аркадних ігор».

Успіх цієї гри відкрив аркадним автоматам протягом «золотої ери» шлях у такі суспільно значущі місця як торгові центри, традиційні торгові зали, ресторани та цілодобові магазини. У всьому світі всього було продано понад 360 тисяч аркадних автоматів з грою Space Invaders, таким чином у 1982 році гра заробила 2 мільярди доларів монетами по 25 центів, що становить близько 6 мільярдів доларів за сучасним курсом.

На початку 1980-х «золота ера аркадних ігор» була у розквіті. Обсяг ринку аркадних автоматів США значно збільшився з $50 млн в 1978 до $900 млн в 1981, при цьому дохід всієї індустрії аркадних ігор потроївся, склавши $2,8 млрд в 1980. За один тільки 1981 індустрія аркадних ігор у США заробила $5 млрд, тобто $17,5 млрд у сучасних цінах. Піка аркадні ігри досягли 1982 року, заробивши $8 млрд ($28 млрд у цінах 2022), перевищивши сумарні продажі

поп-музики та кіноіндустрії США. Це також було більш ніж у 2 рази вище за продажі ринку домашніх ігрових систем, які в тому ж році склали $3,8 млрд.

Таким чином сумарний дохід ринків аркадних та домашніх ігор індустрії комп'ютерних ігор 1982 року становив $11,8 млрд(більше $41,3 млрд у цінах 2022). Ринок аркадних ігор продовжував приносити по $5 млрд продажів до 1985 року. Найбільш значущою грою того періоду була випущена в 1980 Pac-Man від Namco, розпродана в кількості більше 350 тисяч автоматів і заробила за рік більш ніж $1 млрд.

Початок цього періоду також збігся з появою багатьох домашніх комп'ютерів та ентузіастів-розробників ігор для них. Особливо сильний вплив вони вплинули в Європі (комп'ютер ZX Spectrum) та в Азії (NEC PC-8801 і MSX). Також у цей час з'явилися перші видання, присвячені комп'ютерним іграм, яких згодом стали додаватися різні носії даних.

У 1983 році ігрову індустрію в США потряс найпотужнішу кризу, викликану виробництвом великої кількості дуже погано розроблених ігор (що можна охарактеризувати як «кількість важливіша за якість»), через що в американській ігровій індустрії почався спад. Відновлення промисловості пов'язане з великим успіхом, якого досягла домашня ігрова приставка Nintendo Entertainment System, що призвело до захоплення цього ринку японськими компаніями.

Приблизно в той же час почала набувати форми і європейська ігрова індустрія з такими компаніями, як, наприклад, Ocean Software. До кінця десятиліття було випущено портативну ігрову систему GameBoy.

У 1987 році Nintendo програла судову справу проти Blockbuster LLC, що зробило законним такий спосіб поширення ігор як здавання напрокат подібно до кінофільмів.

У 1990-х стався подальший розвиток технологій, супутніх комп'ютерним іграм. Найбільш значущі:

1. Широке впровадження CD-ROM для поширення та зберігання даних.

2. Широке поширення операційних систем, що ґрунтуються на GUI, таких як AmigaOS, Microsoft Windows та Mac OS.

3. Суттєвий розвиток технологій тривимірної графіки та широке поширення 3D графічних процесорів, перехід до тривимірної графіки як до стандарту де-факто-візуалізації ігор.

4. Продовження покращення швидкодії CPU, всебічний розвиток архітектури.

5. Поява та поширення інтернету, внаслідок чого у другій половині десятиліття стала доступною спільна гра, що призвело до появи кіберспорту.

Поруч із розвитком технологій відбувався розвиток ринку ігор. У 1993 році продажі комп'ютерних ігор у світі склали $19,8 млрд($39.7 млрд у цінах 2022), $20,8 млрд у 1994 ($41 млрд у цінах 2022) та приблизно $30 млрд в 1998 ($ 53,1 млрд у цінах 2022). Продажі аркадних автоматів у США в 1994 склали $7 млрд ($14 млрд у цінах 2022), тоді як продажі ігор для домашніх консолей склали $6 млрд ($11.5 млрд у цінах 2022). Таким чином, сумарні продажі ігрової індустрії США більш ніж у 2,5 рази перевищили продажі кінематографа в США.

У 21 сторіччі ігри залишаються рушійною силою розвитку комп'ютерних технологій, які згодом використовуються для інших цілей. Наприклад, ігрові застосунки надали поштовху розвитку VR та AR технологіям, що тепер використовуються для тренування пілотів цивільної та військової авіації, бізнесом у маркетингових цілях і навіть у медицині задля лікування психосоматичних захворювань. Також зміцнів напрямок ігор для мобільних платформ. Мініатюризація апаратного забезпечення та масове поширення мобільних телефонів сприяло появі мобільних ігрових додатків; виник напрям створення ігор для соціальних мереж. Особливо відомий розробник ігор Zynga для соціальної мережі Facebook. Іншим прикладом успішних платформ для комп'ютерних ігор є iOS та Android.

Ігровий ринок лише в США має приблизну вартість у 95.4 мільярдів доларів, що втричі більше, у порівнянні з 2012 роком(31.23 мільярди). Майже половина доходу усієї індустрії приходиться на мобільний геймінг.

Загальновідомим хоч і хибним, або принаймні застарілим, ствердженням є те, що ігри призначені виключно для дітей, а отже не можуть бути визнанні як щось серйозне чи варте уваги. Доказ протилежного наведено на рис 1.1., наведеному нижче з детальною інформацією щодо складу гравців з України:

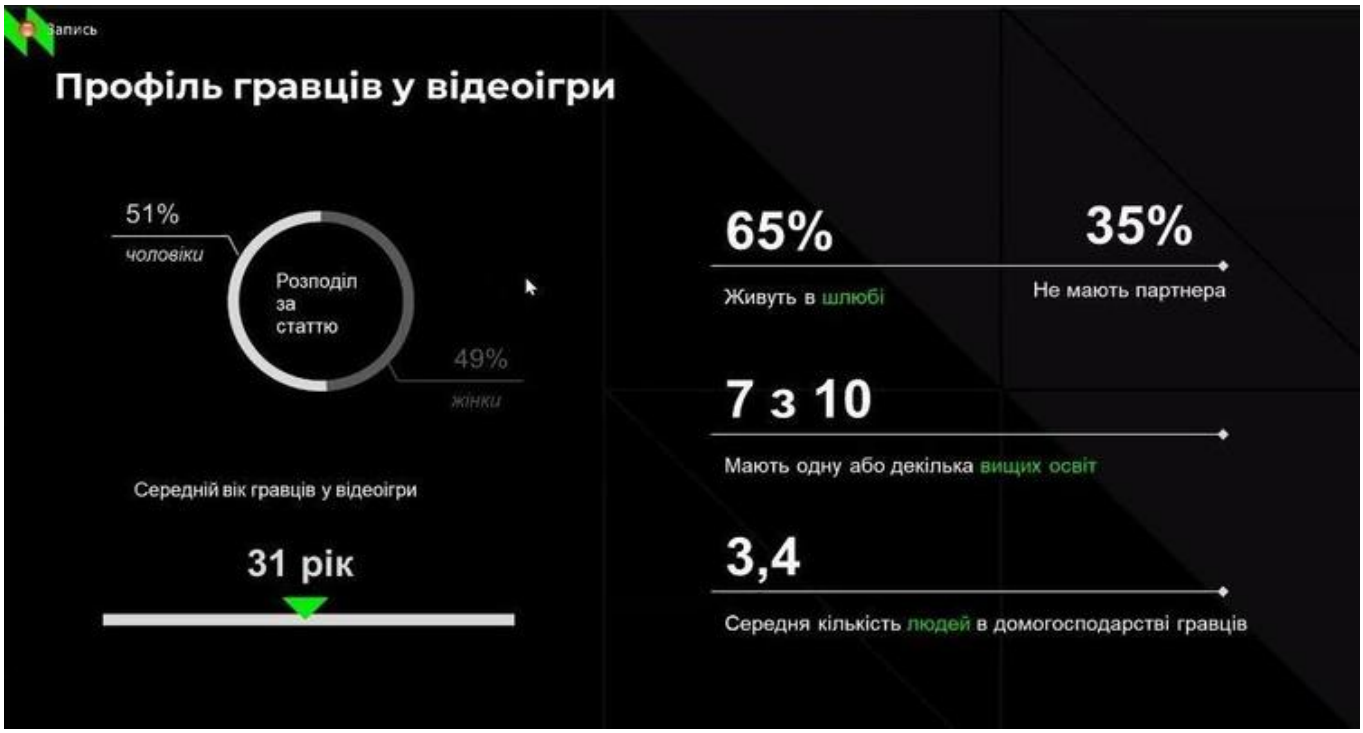

Рис.1.1. Профіль гравців у відеоігри

1.2. Призначення розробки та галузь застосування

Хоча ігрову індустрію і неодноразово звинувачували у безлічі проблем як соціального, так і фізіологічного характеру, але це зовсім не так. Насправді розумне проведення часу за відеоіграми не лише не завдає шкоди, а й приносить певну користь.

Так ігри з використанням контролера можуть бути чудовими для ваших рук. У дослідженні за участю групи хірургів дослідники виявили, що ті, хто грав у відеоігри, швидше виконували просунуті операції і робили на 37

відсотків менше помилок, ніж ті, хто цього не робив. Спеціальні відеоігри також використовувалися як фізіотерапія, щоб допомогти постраждалим від інсульту відновити контроль над своїми руками та зап’ястями.

Поки ви не дивитеся на екран протягом 10 годин поспіль, гра у відеоігри дійсно може покращити ваш зір. В одному дослідженні 10 студентів-чоловіків, які не були геймерами, навчалися протягом 30 годин в іграх від першої особи, а потім тестувалися проти 10 не-геймерів. Учні, які грали, змогли чіткіше бачити об’єкти в захараченому просторі завдяки покращеній просторовій роздільній здатності. Вони змогли натренувати свій мозок бачити дрібніші деталі, тому що в кожній грі ці деталі виявилися важливими.

Дослідження показали, що деякі відеоігри можуть підвищити настрій і покращити серцевий ритм — ознака того, що вони також можуть допомогти зняти стрес. Кореляція (а не причинно-наслідковий зв’язок) між відеоіграми та стресом знайшла відображення в численних не пов’язаних дослідженнях, тому відеоігри використовуються в терапії вже більше десяти років.

Метою даної кваліфікаційної роботи є реалізація ігрового, додатку, що буде допомагати з покращенням рівня стресу, розвитку логічного та стратегічного мислення та може бути використаний у рекреаційних цілях.

1.3. Підстави для розробки

Відповідно до освітньої програми, згідно навчального плану та графіків навчального процесу, в кінці навчання студент виконує кваліфікаційну роботу.

Тема роботи узгоджується з керівником проекту, випускаючою кафедрою, та затверджується наказом ректора.

Отже, підставами для розробки (виконання кваліфікаційної роботи) є:

- освітня програма 122 «Комп’ютерні науки»;
- навчальний план та графік навчального процесу;

– наказ ректора Національного технічного університету «Дніпровська політехніка» № 268-с від 18.05.2022 р;

– завдання на кваліфікаційну роботу на тему “Розробка мультиплатформеного ігрового додатку за допомогою Unity”

1.4. Постановка завдання

Завданням даної роботи є розробка мультиплатформеного ігрового додатку в жанрі “стратегія”. В результаті необхідно спроектувати та розробити гру для декількох платформ. Для прикладу будуть використані платформи Windows та Android.

Даний додаток призначений для використання двома гравцями, тож буде сприяти підвищеній соціалізації, розвитку стратегічного мислення та загального зниження стресу через ігрову форму.

1.5. Вимоги до програми або програмного виробу

1.5.1. Вимоги до функціональних характеристик

Кінцевий продукт повинен дотримуватися наступних функціональних вимог:

- зручний та зрозумілий користувацький інтерфейс;
- можливість керування як за допомогою клавіатури так і за допомогою сенсорного екрану;
- збереження налаштування та прогресу користувача між сесіями;
- додаток матиме два основні екрани: головне меню, що матиме функціонал для зміни налаштувань під користувача та ігровий екран, де буде проходити весь ігровий процес.

1.5.2. Вимоги до інформаційної безпеки

Для забезпечення інформаційної безпеки користувача повинні бути забезпечені такі умови:

- конфіденційність даних користувача;
- регулярні оновлення для виправлення можливих помилок у роботі та усунення небезпек;
- забезпечення неможливості передачі даних користувача третім сторонам.

1.5.3. Вимоги до складу та параметрів технічних засобів

Для роботи з додатком використовується його дистрибутив, що встановлюється в системі користувача.

Відповідні характеристики необхідні для роботи додатку на Windows :

- Операційна система: Windows 7 (SP1+) або вище
- ЦП: x86, x64 архітектура з підтримкою SSE2;
- відеоадаптер: з підтримкою DX10, DX11, DX12;
- відеопам’ять: 128 МБ;
- накопичувач: 100мб+ вільного місця на диску;
- оперативна пам’ять: 2 Гб.

Для MacOS:

- Операційна система: Sierra 10.12+
- ЦП: x64 архітектура з підтримкою SSE2;
- відеоадаптер: з підтримкою Metal;
- відеопам’ять: 128 МБ;
- накопичувач: 100мб+ вільного місця на диску;
- оперативна пам’ять: 2 Гб.

Для Android :

- Операційна система: 4.4 (API 19)+

- ЦП: ARMv7 з підтримкою Neon(32-bit) або ARM64
- відеоадаптер: з підтримкою OpenGL ES 2.0+, OpenGL ES 3.0+, Vulkan;
- відеопам’ять: 128 МБ;
- накопичувач: 100мб+ вільного місця на диску;
- оперативна пам’ять: 1 Гб.

Для iOS :

- Операційна система:iOS12+
- ЦП: A6/A6X SoC+;
- відеоадаптер: з підтримкою Metal;
- відеопам’ять: 128 МБ;
- накопичувач: 100мб+ вільного місця на диску;
- оперативна пам’ять: 1 Гб.

1.5.4. Вимоги до інформаційної та програмної сумісності

Ігровий додаток був розроблений на мові C# з використанням ігрового двигуна Unity, що забезпечує сумісність із майже усіма сучасними операційними системами, але при цьому необхідно дотримуватися умов, що були перелічені у розділі 1.5.3. В залежності від способу отримання дистрибутиву програми оновлення відбувається за допомогою стандартного функціоналу маркетплейсів Android/iOS, Windows, MacOs або шляхом перезавантаження та перевстановлення додатку у випадку отримання іншим шляхом, при цьому дані користувача втрачені не будуть.

РОЗДІЛ 2
ПРОЄКТУВАННЯ ТА РОЗРОБКА ІНФОРМАЦІЙНОЇ СИСТЕМИ

2.1. Загальні відомості з предметної галузі

Результатом даної кваліфікаційної роботи має бути ігровий мультиплатформений додаток, що дозволяє грати двом користувачам на будь-якій з підтримуваних платформ.

В ході роботи програми повинні підтримуватися правила гри, а також зрозумілий для гравців ігровий процес. По завершенню роботи програми повинні бути збережені усі налаштування користувача.

Основним призначенням додатку є використання у рекреаційних цілях. Також гра сприяє підвищеній соціалізації, розвитку стратегічного мислення та загального зниження стресу через ігрову форму.

2.2. Опис застосованих математичних методів

В даному ігровому додатку не використовуються складні математичні методи, а лише прості арифметичні операції, здебільшого з бібліотеки Mathf. При розробці додатку були використанні такі математичні методи: Pow (піднесення до степеня), Clamp (затискає значення між максимальним та мінімальним значеннями), Lerp (лінійна інтерполяція).

2.3. Опис використаних технологій та мов програмування

Unity — це багатоплатформовий ігровий двигун, розроблений Unity Technologies, вперше анонсований і випущений у червні 2005 року. З тих пір двигун поступово розширювався для підтримки різноманітних настільних, мобільних, консольних і віртуальних платформ. Він особливо популярний для

розробки мобільних ігор для iOS і Android і використовується для таких ігор, як Pokémon Go, Monument Valley, Call of Duty: Mobile, Beat Sabre і Cuphead. Він вважається простим у використанні для початківців розробників і популярним для розробки інді-ігор.

Unity дає користувачам можливість створювати ігри як у 2D, так і в 3D, а двигун пропонує основний API сценаріїв на C# за допомогою Mono, як для редактора Unity у формі плагінів, так і для самих ігор, а також для drag-and-drop функціональність. До того, як C# став основною мовою програмування, який використовується для двигуна, він раніше підтримував Boo, який було видалено з випуском Unity 5, та реалізацію JavaScript на основі Boo під назвою UnityScript, яка була замінена в серпні 2017 року на користь C#.

У 2D-іграх Unity дозволяє імпортувати спрайти та підтримує розширений рендер 2D світу. Для 3D-ігор Unity дозволяє специфікувати стиснення текстур, mip-maps і налаштування роздільної здатності для кожної платформи, яку підтримує ігровий двигун, та забезпечує підтримку відображення рельєфу, віддзеркалення, паралакса, екранного простору навколишнього середовища (SSAO), динамічні тіні за допомогою карт тіней, рендерингу до текстури та повноекранних ефектів постобробки.

Доступні два окремих пайплайну візуалізації, пайплайн візуалізації високої чіткості (HDRP) і універсальний пайплайн візуалізації (URP), на додаток до застарілого вбудованого.

Станом на 2020 рік програмне забезпечення, створене за допомогою ігрового двигуна Unity, працювало на понад 1,5 мільярда пристроїв. За даними Unity, додатки, створені за допомогою їхнього ігрового движка, становлять 50 відсотків усіх мобільних ігор і завантажуються понад 3 мільярди разів на місяць, і приблизно 15 000 нових проектів щодня запускаються з його програмним забезпеченням. Financial Times повідомила, що Unity «працює в деяких із найприбутковіших мобільних ігор у світі», таких як Pokémon Go і Call of Duty Mobile від Activision.

C# - це багатопарадигмова мова програмування загального призначення. C# охоплює дисципліни програмування зі статичною типізацією, жорсткою типізацією, лексичною, імперативною, декларативною, функціональною, загальною, об'єктно-орієнтованою (на основі класів) і компонентно-орієнтованою.

Стандарт Ecma перелічує ці цілі дизайну для C#:

- Ця мова має бути простою, сучасною, об’єктно-орієнтованою мовою програмування загального призначення.

- Мова та її реалізації повинні забезпечувати підтримку принципів розробки програмного забезпечення, таких як сильна перевірка типів, перевірка меж масиву, виявлення спроб використання неініціалізованих змінних і автоматичне збирання сміття. Надійність програмного забезпечення, довговічність і продуктивність програміста важливі.

- Ця мова призначена для використання при розробці програмних компонентів, придатних для розгортання в розподілених середовищах.

- Переносність дуже важлива для вихідного коду та програмістів, особливо тих, хто вже знайомий із C та C++.

- Підтримка інтернаціоналізації дуже важлива.

C# призначений для написання програм як для розміщених, так і для вбудованих систем, починаючи від дуже великих, які використовують складні операційні системи, і закінчуючи дуже малими, які мають спеціальні функції.

Незважаючи на те, що програми C# розроблені аби бути економними щодо пам’яті та потужності обробки, мова не була призначена для прямої конкуренції за продуктивністю та розміром із C або мовою асемблера.

За дизайном, C# є мовою програмування, яка найбільш безпосередньо відображає базову загальномовну інфраструктуру (CLI). Більшість його внутрішніх типів відповідають типам значень, реалізованим фреймворком CLI. Однак у специфікації мови не вказано вимоги компілятора до генерації коду: тобто не вказано, що компілятор C# повинен націлюватися на середовище виконання Common Language, або генерувати Common Intermediate Language

(CIL) або створювати будь-який інший специфічний формат. Теоретично компілятор C# може генерувати машинний код, як традиційні компілятори C++ або Fortran.

C# підтримує чіткі неявно введені оголошення змінних з ключовим словом var і неявно введені масиви з ключовим словом new[], за яким слідує ініціалізатор колекції.

C# підтримує строгий логічний тип даних, bool. Оператори, які приймають умови, наприклад while і if, вимагають виразу типу, який реалізує оператор істинності, наприклад булевий тип. Хоча C++ також має булевий тип, його можна вільно перетворювати в цілі числа та з них, а такі вирази, як if (a), вимагають лише конвертації a в bool, дозволяючи a бути int або вказівником. C# забороняє цей підхід «цілочисельне значення істина чи хибна» на тій підставі, що примушування програмістів використовувати вирази, які повертають точно bool, може запобігти певним типам помилок програмування, наприклад, якщо (a = b) (використання присвоєння = замість рівності ==).

C# є більш безпечним для типів, ніж C++. Єдиними неявними перетвореннями за замовчуванням є ті, які вважаються безпечними, наприклад розширення цілих чисел. Це застосовується під час компіляції, під час JIT-компіляції і, в деяких випадках, під час виконання. Ніяких неявних перетворень між булевими та цілими числами, а також між членами перерахування та цілими числами не відбувається (за винятком літералу 0, який можна неявно перетворити на будь-який перерахований тип). Будь-яке визначене користувачем перетворення має бути явно позначено як явне чи неявне, на відміну від конструкторів копіювання C++ та операторів перетворення, які за замовчуванням є неявними.

C# має явну підтримку коваріації та контрваріантності в загальних типах, на відміну від C++, який має певний ступінь підтримки контраваріантності просто через семантику типів повернення у віртуальних методах.

Мова C# не дозволяє використовувати глобальні змінні або функції. Усі методи та члени мають бути оголошені всередині класів. Статичні члени

відкритих класів можуть замінювати глобальні змінні та функції. Члени перерахування розміщуються у власній області дії.

Метод у C# — це член класу, який можна викликати як функцію (послідовність інструкцій), а не просто здатність властивості класу зберігати значення. Як і в інших синтаксично подібних мовах, таких як C++ і ANSI C, сигнатурою методу є оголошення, що містить у порядку: будь-які додаткові ключові слова доступності (наприклад, private), явну специфікацію його типу повернення (наприклад, int або ключове слово void, якщо значення не повертається), ім'я методу і, нарешті, послідовність у круглих дужках специфікацій параметрів, розділених комами, кожна з яких складається з типу параметра, його формальної назви та, за бажанням, значення за замовчуванням, яке буде використовуватися, коли немає жодного надано. Певні конкретні типи методів, наприклад ті, які просто отримують або встановлюють властивість класу шляхом повернення значення або присвоєння, не вимагають повного підпису, але в загальному випадку визначення класу включає повне оголошення підпису його методів.

Як і C++, і на відміну від Java, програмісти C# повинні використовувати ключове слово модифікатора області virtual, щоб дозволити методам перевизначати підкласи.

Методи розширення в C# дозволяють програмістам використовувати статичні методи, як ніби вони є методами з таблиці методів класу, дозволяючи додавати методи до об’єкта, який, на думку розробника, має існувати на цьому об’єкті та його похідних.

Тип dynamic дозволяє прив’язувати метод під час виконання, дозволяючи виклики методів, подібних до JavaScript, і композицію об’єктів під час виконання.

C# підтримує строго типізовані покажчики функцій через ключове слово делегат. Подібно до сигналу та слоту псевдо-C++ фреймворку Qt, C# має семантику, що стосується подій стилю публікації-підписки, хоча C# використовує для цього делегатів.

C# пропонує синхронізовані виклики методів, подібні до Java, через атрибут [MethodImpl(MethodImplOptions.Synchronized)] і підтримує взаємовиключні блокування за допомогою ключового слова lock.

2.4. Опис структури системи та алгоритмів її функціонування

Для демонстрації варіантів сценаріїв використання користувачем програмного додатку та послідовностей ігрового процесу нижче наведені ULM діаграми варіантів використання (Use Case):

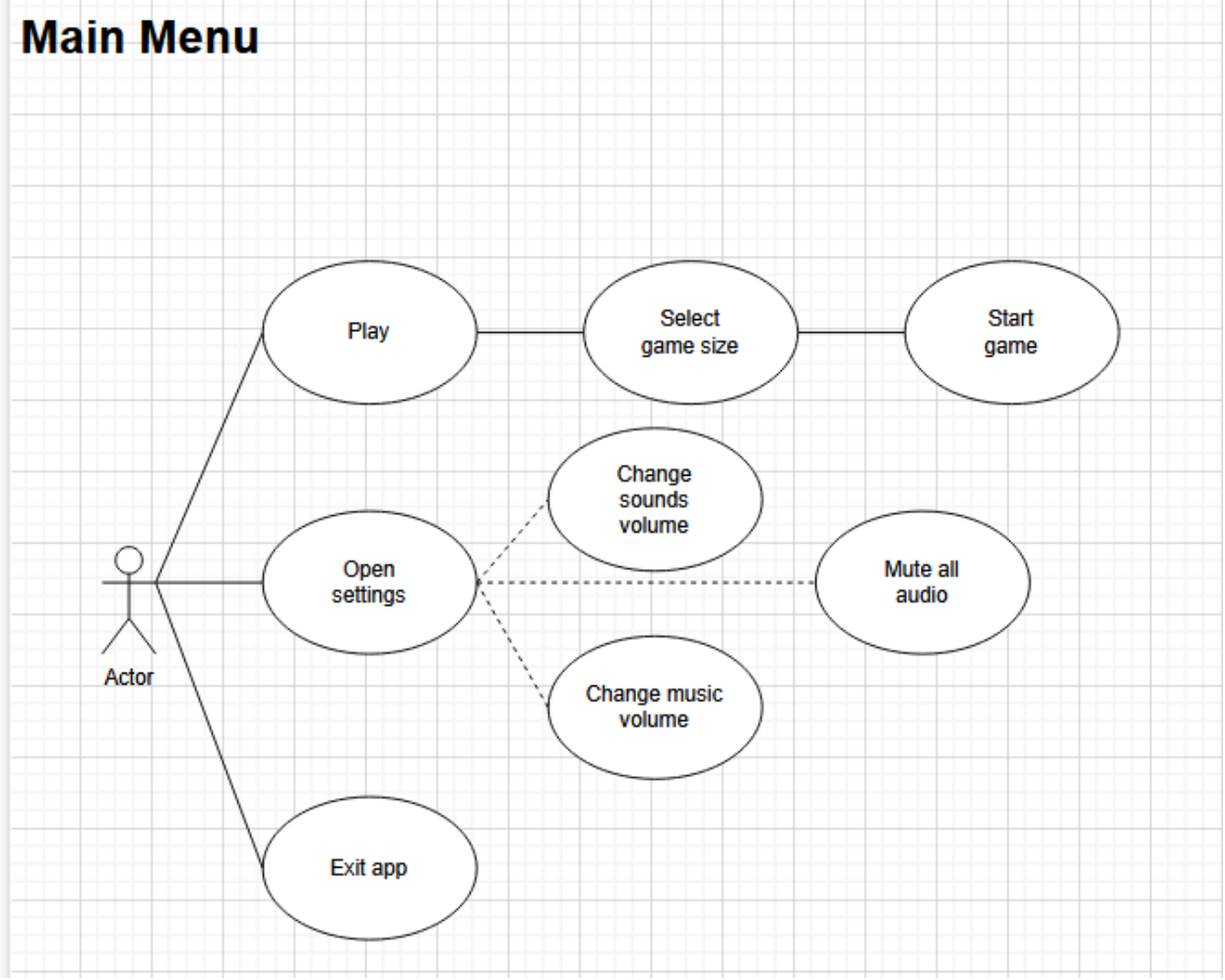

Рис. 2.1. Діаграма варіантів використання для головного меню

В головному меню користувач може розпочати гру, додатково обравши її розмір, відкрити вікно налаштувань та відредагувати значення під себе або зачинити, дивіться додаток рис. 2.1.

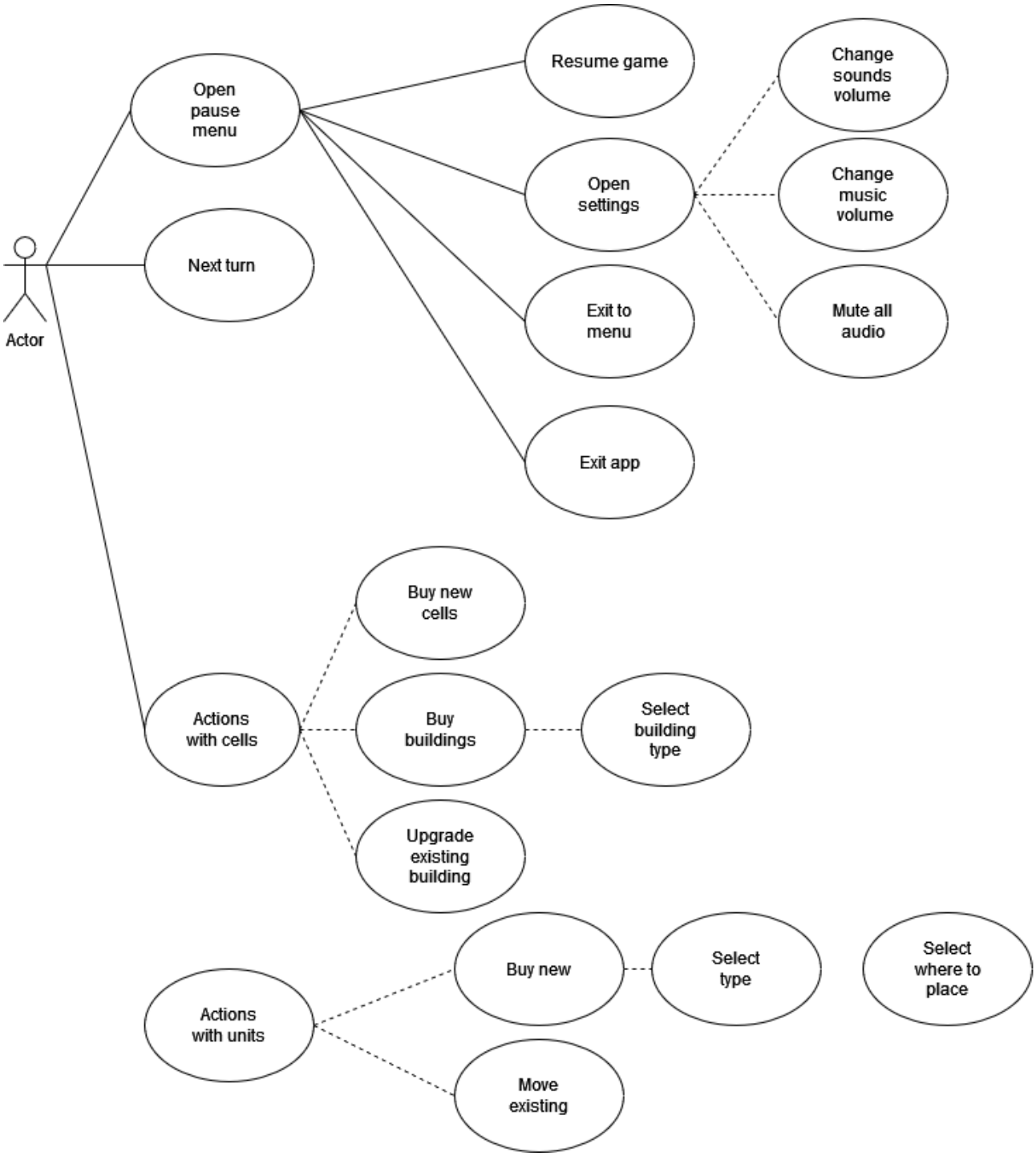

Рис. 2.2. Діаграма варіантів використання для основного ігрового екрану

На основному ігровому екрані, рис. 2.1. та рис. 2.2., користувач може:

- відкрити меню паузи, де може вийти в меню або з гри, відкрити вікно налаштувань;

- передати хід наступному гравцю;

- проводити дії з клітинами ігрового поля: купувати собі у власність нові, на існуючих будувати будівлі різних типів, а якщо будівля вже існує її можна поліпшити;

- проводити дії з юнітами гравця: купувати нових та розміщати їх або рухати по полю вже існуючих.

При розробці ігрового додатку мною були прийняті такі патерни проектування для створення функціоналу програми:

Сінглтон - це породжувальний патерн проектування, який гарантує, що клас має лише один екземпляр, і надає до нього глобальну точку доступу. Всі реалізації сінглтона зводяться до того, щоб приховати конструктор за замовчуванням і створити публічний статичний метод, який і контролюватиме життєвий цикл об'єкта-сінглтона. Якщо у вас є доступ до класу сінглтона, значить, буде доступ і до цього статичного методу. З якої точки коду ви б його не викликали, він завжди віддаватиме той самий об'єкт.

Посередник - це поведінковий патерн проектування, який дозволяє зменшити зв'язаність безлічі класів між собою завдяки переміщенню цих зв'язків в один клас-посередник. Патерн посередник змушує об'єкти спілкуватися не безпосередньо один з одним, а через окремий об'єкт-посередник, який знає, кому потрібно перенаправити той чи інший запит. Завдяки цьому компоненти системи залежатимуть тільки від посередника, а не від десятків інших компонентів.

Спостерігач – це поведінковий патерн проектування, який створює механізм підписки, що дозволяє одним об'єктам стежити та реагувати на події, що відбуваються в інших об'єктах. Патерн спостерігач пропонує зберігати в об'єкті видавця список посилань на об'єкти передплатників, причому видавець не повинен вести список підписки самостійно. Він надасть методи, за допомогою яких передплатники могли б додавати або прибирати себе зі списку.

Оскільки додаток написаний за допомогою ігрового двигуна Unity в ньому використовуються вбудовані методи івентів, що мають заздалегідь визначений порядок викликів, дивіться рис. 2.3. та рис. 2.4.:

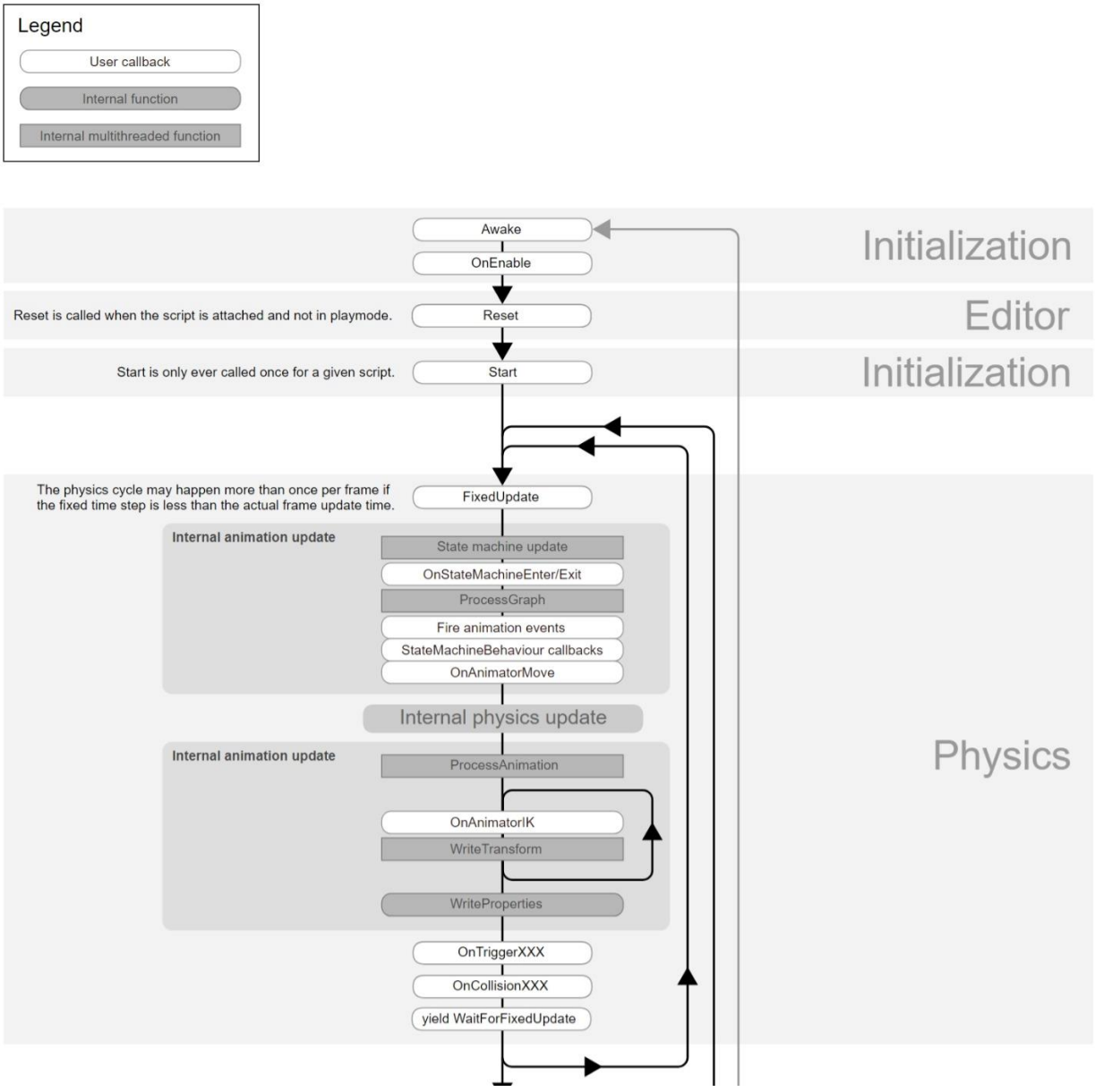

Рис. 2.3. Порядок викликів вбудованих методів івентів частина 1

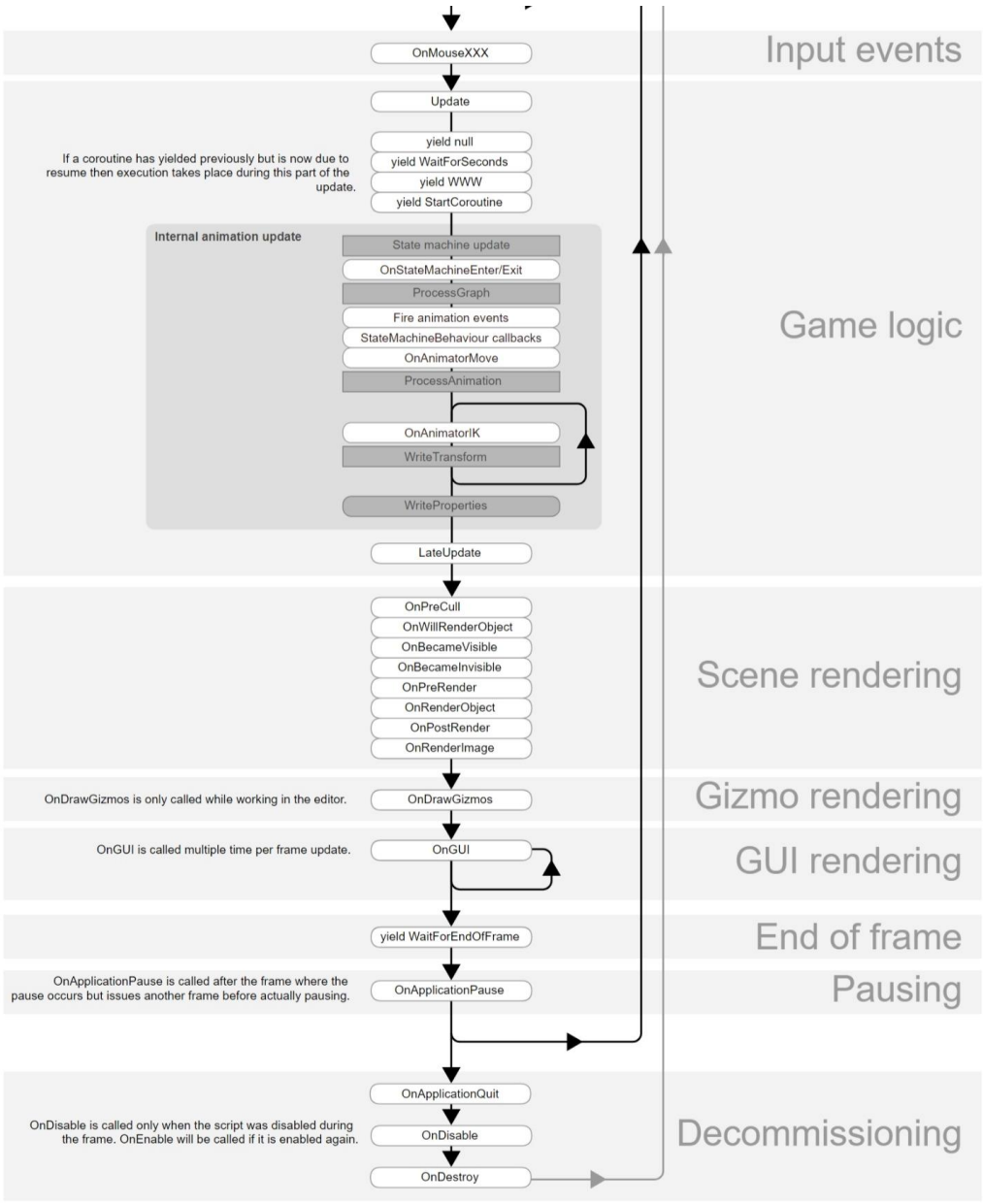

Рис. 2.4. Порядок викликів вбудованих методів івентів частина 2

2.5. Обґрунтування та організація вхідних та вихідних даних програми

В якості вхідних параметрів ігровий додаток приймає дані пристроїв вводу для управління ігровим процесом, а точніше натиснення клавіш клавіатури, позицію та натиснення кнопок миші, а також аналоги вищеназваних з використанням сенсорного екрану мобільних пристроїв. З вихідних даних програма видає лише файл, що зберігає у собі дані користувача, такі як налаштування та персоналізація.

2.6. Опис розробленої системи

2.6.1. Використані технічні засоби

Для розробки була використана персональна електронно-обчислювальна машина (ЕОМ) з наступними характеристиками:

- Операційна система: Windows 10 Home
- ЦП: Intel(R) Core(TM) i5-8250U CPU @ 1.60GHz 1.80 GHz
- відеоадаптер: Intel Uhd Graphics
- аудіокарта: Realtek High Defenition Audio
- накопичувач: SATA SSD 256 GB
- оперативна пам’ять: 8 Гб.

А також засоби вводу -виведення: монітор з роздільною здатністю 1920*1080, клавіатура, комп’ютерна миша.

2.6.2. Використані програмні засоби

Unity - багатоплатформове середовище розробки комп'ютерних ігор, розроблене американською компанією Unity Technologies. Unity дозволяє створювати програми, що працюють на більш ніж 25 різних платформах, що включають персональні комп'ютери, ігрові консолі, мобільні пристрої, інтернет-програми та інші. Випуск Unity відбувся у 2005 році і з того часу триває постійний розвиток.

Microsoft Visual Studio - інтегроване середовище розробки (IDE) від Microsoft. Воно використовується для розробки комп’ютерних програм, а також веб-сайтів, веб-додатків, веб-сервісів та мобільних додатків. Visual Studio використовує платформи розробки програмного забезпечення Microsoft, такі як Windows API, Windows Forms, Windows Presentation Foundation, Windows Store і Microsoft Silverlight. Може створювати як рідний код, так і керований код.

GitHub - постачальник Інтернет-хостингу для розробки програмного забезпечення та контролю версій за допомогою Git. Він пропонує розподілений контроль версій і управління вихідним кодом (SCM) Git, а також власні функції. Забезпечує контроль доступу та декілька функцій спільної роботи, таких як відстеження помилок, запити функцій, керування завданнями, безперервна інтеграція та вікі для кожного проекту. Станом на листопад 2021 року GitHub повідомляє, що має понад 73 мільйони розробників і понад 200 мільйонів репозиторіїв (включаючи принаймні 28 мільйонів загальнодоступних сховищ). Це найбільший хост вихідного коду станом на листопад 2021 року.

Microsoft Paint - простий растровий графічний редактор, який входить до складу всіх версій Microsoft Windows. Програма відкриває та зберігає файли у форматах Windows bitmap (BMP), JPEG, GIF, PNG та односторінкових TIFF. Програма може бути в кольоровому або двоколірному чорно-білому режимі, але немає режиму відтінків сірого. Завдяки своїй простоті та тому, що вона входить до складу Windows, вона швидко стала однією з найбільш використовуваних програм у ранніх версіях Windows, вперше познайомивши багатьох із малюванням на комп'ютері. Він досі широко використовується для простих завдань маніпулювання зображеннями.

Blender - безкоштовний набір програмних засобів для комп'ютерної 3D-графіки з відкритим вихідним кодом, який використовується для створення анімаційних фільмів, візуальних ефектів, мистецтва, 3D-друкованих моделей, анімаційної графіки, інтерактивних 3D-додатків, віртуальної реальності та, раніше, відеоігор. Функції Blender включають 3D-моделювання, УФ-картографування, текстурування, цифрове малювання, редагування растрової графіки, такелаж і зняття шкіри, моделювання рідини та диму, моделювання частинок, моделювання м'якого тіла, скульптування, анімацію, переміщення матчів, рендеринг, графіку руху, редагування відео та композитинг. Blender популярний серед спільноти FOSS і початківців завдяки своїй доступності та безкоштовності.

2.6.3. Виклик та завантаження програми

Для комп’ютерів на базі ОС Windows:

Архів з додатком завантажується з мережі інтернет та розпаковується за допомогою програм архіваторів(наприклад, 7ZIP). Після розпаковки додаток можна запускати через .exe файл, дивіться рис. 2.5.

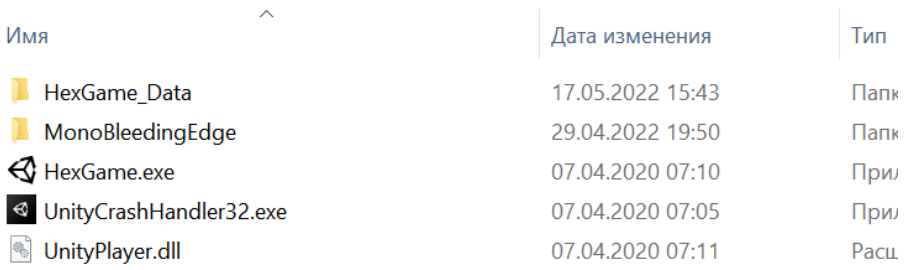

Рис. 2.5. Приклад папки з додатком після розпакування архіву

Для мобільних пристроїв на базі Android:

Необхідно завантажити .apk файл та запустити його. Далі ви побачите стандартне вікно встановлення додатків, дивіться рис. 2.6. (перед цим може знадобитися дозволити встановлення програм з невідомих джерел). Після встановлення .apk файлу додаток запускається за допомогою іконки, що може бути знайдена на головному екрані або у меню з рештою програм.

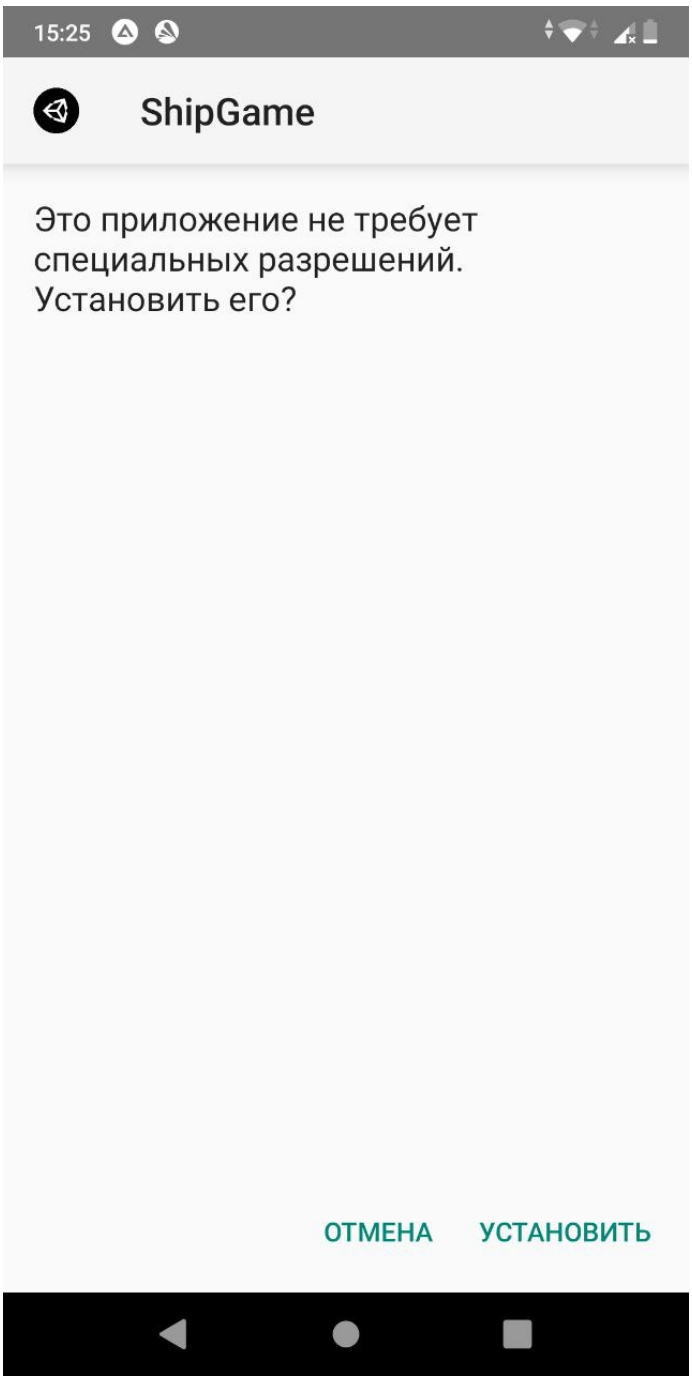

Рис. 2.6. Приклад інтерфейсу вікна встановлення додатку на Android

2.6.4. Опис інтерфейсу користувача

Після запуску ігрового додатку першим екраном, що бачить користувач є головне меню. Тут він може запустити гру, вибрати її розмір, зайти в екран налаштувань або вийти з гри, дивіться рис. 2.7. - рис. 2.9.

Рис. 2.7. Головне меню

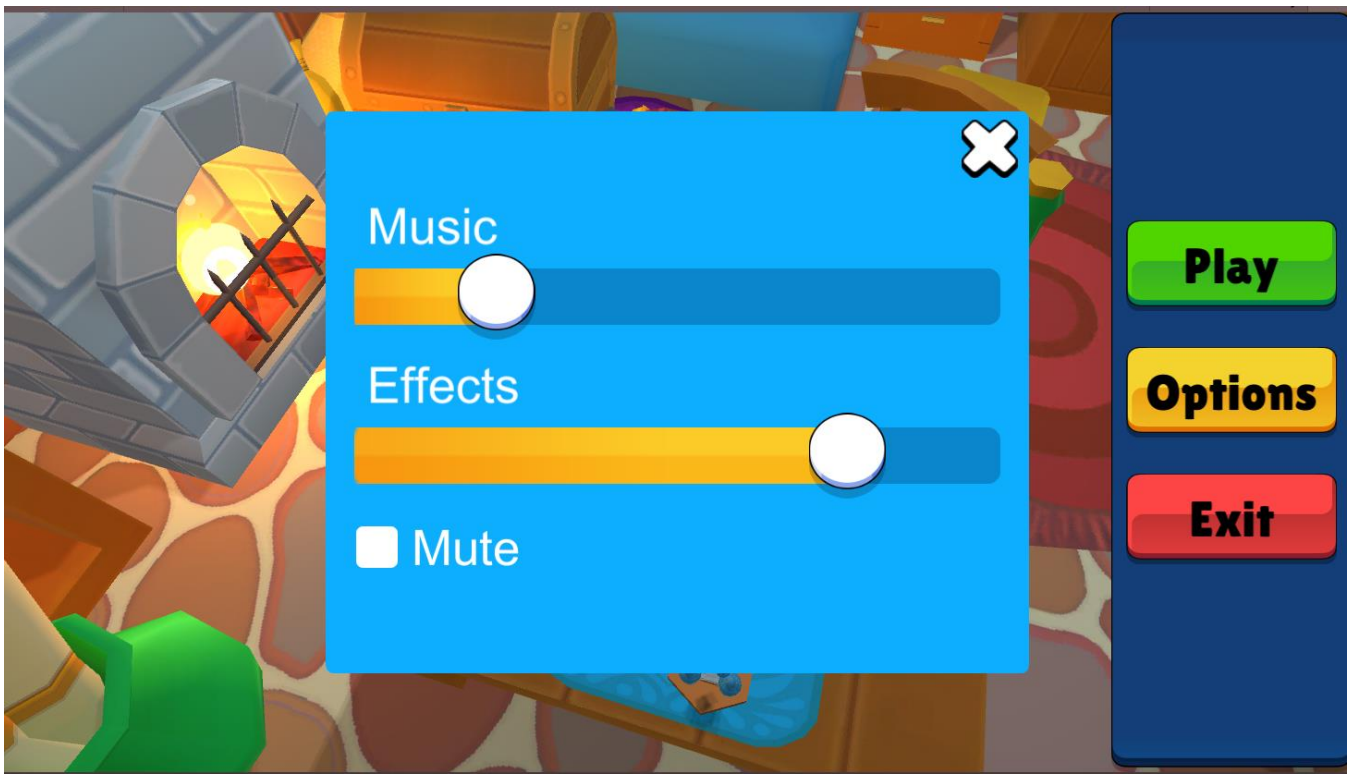

Рис. 2.8. Екран налаштувань

Рис. 2.9. Екран вибору розміру гри

Після вибору розміру гри користувач потрапляє на основний ігровий екран, де він може оперувати кнопками, що викликають зрушення в ігровому процесі, нові екрани з детальною інформацією або контекстними діями, викликати екран паузи, що дозволяє перейти до налаштувань, повернутися в меню, вийти з гри або продовжити гру, дивіться рис. 2.10. - рис. 2.14.

Рис. 2.10. Основний ігровий екран

Рис. 2.11. Екран паузи

Рис. 2.12. – Екран будівництва.

Рис. 2.13. – Екран апгрейду існуючої будівлі.

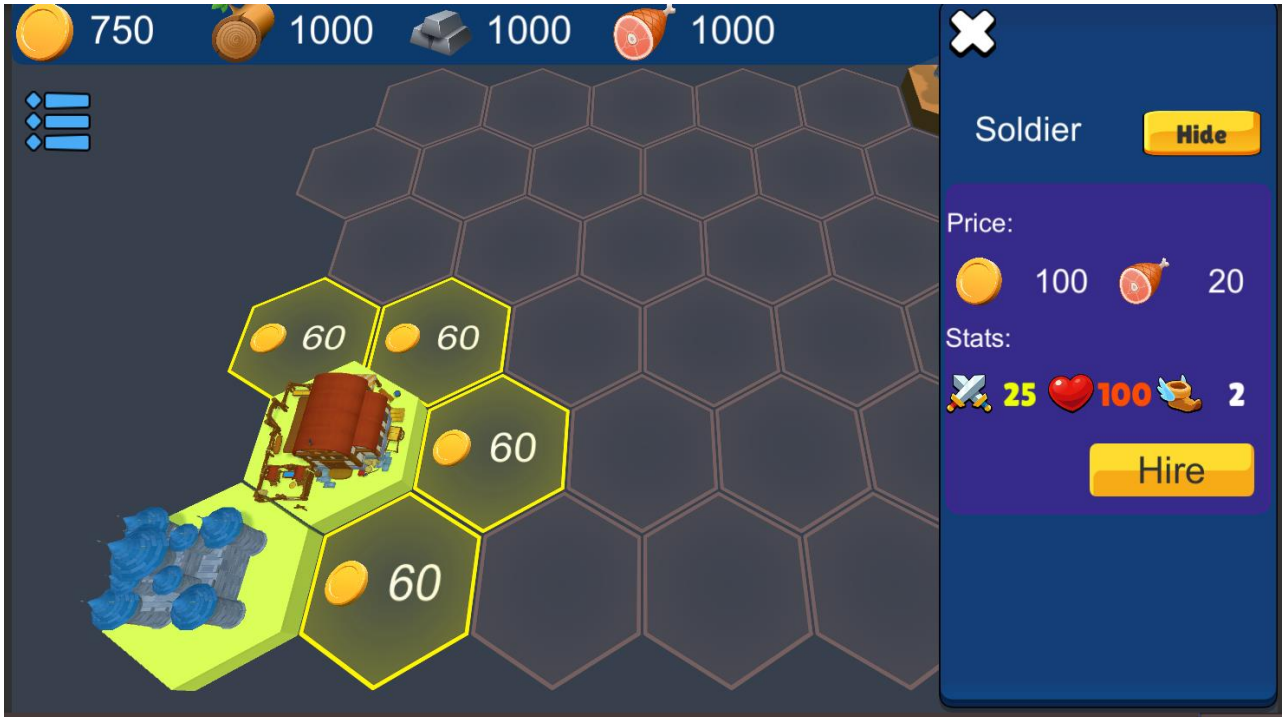

Рис. 2.14. – Екран найму юнітів.

РОЗДІЛ 3
ЕКОНОМІЧНІЙ РОЗДІЛ

3.1. Розрахунок трудомісткості та вартості розробки програмного продукту

Початкові дані:

1. передбачуване число операторів програми – 2500;
2. коефіцієнт складності програми – 1,4;
3. коефіцієнт корекції програми в ході її розробки – 0,06;
4. годинна заробітна плата програміста – 147 грн/год;
5. коефіцієнт збільшення витрат праці внаслідок недостатнього опису задачі – 1,2;
6. Коефіцієнт кваліфікації програміста, обумовлений від стажу роботи з даної спеціальності – 1,3;
7. вартість машино-години ЕОМ – 16 грн/год.

Нормування праці в процесі створення ПЗ істотно ускладнено в силу творчого характеру праці програміста. Тому трудомісткість розробки ПЗ може бути розрахована на основі системи моделей з різною точністю оцінки.

Трудомісткість розробки ПЗ можна розрахувати за формулою:

$$t = t_o + t_u + t_a + t_n + t_{отл} + t_д \text{, людино-годин,} \qquad (3.1)$$

де t_o – витрати праці на підготовку й опис поставленої задачі (приймається 50);

t_u – витрати праці на дослідження алгоритму рішення задачі;

t_a – витрати праці на розробку блок-схеми алгоритму;

t_n – витрати праці на програмування по готовій блок-схемі;

$t_{отл}$ – витрати праці на налагодження програми на ЕОМ;

$t_д$ – витрати праці на підготовку документації.

Складові витрати праці визначаються через умовне число операторів у ПЗ, яке розробляється.

Умовне число операторів (підпрограм):

$$Q = q \cdot C \cdot (1 + p), \qquad (3.2)$$

де q – передбачуване число операторів (2500);

C – коефіцієнт складності програми (1,4);

p – коефіцієнт кореляції програми в ході її розробки (0,06).

$$Q = 2500 \cdot 1{,}4 \cdot (1 + 0{,}06) = 3710;$$

Витрати праці на вивчення опису задачі ti визначається з урахуванням уточнення опису і кваліфікації програміста:

$$t_u = \frac{Q \cdot B}{(75 \ldots 85) \cdot K}, \text{ людино-годин} \qquad (3.3)$$

де B – коефіцієнт збільшення витрат праці внаслідок недостатнього опису задачі (1,2);

K – коефіцієнт кваліфікації програміста, обумовлений стажем роботи з даної спеціальності (1,1);

$$t_u = 3710 \cdot 1{,}2/(85 \cdot 1{,}3) = 40{,}3, \text{ людино-годин.}$$

Витрати праці на розробку алгоритму рішення задачі:

$$t_a = \frac{Q}{(20 \ldots 25) \cdot K}; \qquad (3.4)$$

$$t_a = 3710/(20 \cdot 1{,}3) = 142{,}7, \text{ людино-годин.}$$

Витрати на складання програми по готовій блок-схемі:

$$t_n = \frac{Q}{(20...25)\cdot K}; \quad (3.5)$$

$$t_n = 3710/(25 \cdot 1{,}3) = 114{,}2, \text{ людино-годин.}$$

Витрати праці на налагодження програми на ЕОМ:

– за умови автономного налагодження одного завдання:

$$t_{\text{отл}} = \frac{Q}{(4...5) \cdot K}; \quad (3.6)$$

$$t_{\text{отл}} = 3710/(5 \cdot 1{,}3) = 570{,}8, \text{ людино-годин,}$$

– за умови комплексного налагодження завдання:

$$t_{\text{отл}}^{\text{к}} = 1{,}2 \cdot t_{\text{отл}}; \quad (3.7)$$

$$t_{\text{отл}}^{\text{к}} = 1{,}2 \cdot 570{,}8 = 684{,}9, \text{ людино-годин,}$$

Витрати праці на підготовку документації:

$$t_{\partial} = t_{\partial p} + t_{\partial o}; \quad (3.8)$$

де $t_{\partial p}$ – трудомісткість підготовки матеріалів і рукопису;

$$t_{\partial} = \frac{Q}{(15...20)\cdot K}; \quad (3.9)$$

$$t_{\partial} = 3710/(20 \cdot 1{,}3) = 142{,}7, \text{ людино-годин,}$$

де $t_{\partial o}$ – трудомісткість редагування, печатки й оформлення документації;

$$t_{до} = 0{,}75 \cdot t_{др}; \qquad (3.10)$$

$$t_{до} = 0{,}75 \cdot 142{,}7 = 107{,}3, \text{ людино-годин.}$$

$$t_{д} = 142{,}7 + 107{,}3 = 250, \text{ людино-годин.}$$

Отримаємо трудомісткість розробки програмного забезпечення:

$$t = 50 + 40{,}3 + 142{,}7 + 114{,}2 + 570{,}8 + 250 = 1168, \text{ людино-годин.}$$

У результаті ми розрахували, що в загальній складності необхідно 1168 людино-годин для розробки даного програмного забезпечення.

3.2. Рахунок витрат на створення програми

Витрати на створення ПЗ Кпо включають витрати на заробітну плату виконавця програми ЗЗП і витрат машинного часу, необхідного на налагодження програми на ЕОМ.

$$К_{ПО} = З_{ЗП} + З_{МВ}, \text{ грн,} \qquad (3.11)$$

$З_{ЗП}$ – заробітна плата виконавців, яка визначається за формулою:

$$З_{ЗП} = t \cdot С_{ПР}, \text{ грн,} \qquad (3.12)$$

де t – загальна трудомісткість, людино-годин;

$С_{ПР}$ – середня годинна заробітна плата програміста, грн/година.

З урахуванням того, що середня годинна зарплата програміста становить 147 грн/год, то отримаємо:

$$З_{ЗП} = 1168 \cdot 147 = 171696, \text{ грн.}$$

Вартість машинного часу ЗМВ, необхідного для налагодження програми на ЕОМ, визначається за формулою:

$$З_{МВ} = t_{отл} \cdot C_{М}, \text{ грн,} \qquad (3.13)$$

де $t_{отл}$ – трудомісткість налагодження програми на ЕОМ, год;

$C_{МЧ}$ – вартість машино-години ЕОМ, грн/год.

$$З_{МВ} = 570{,}8 \cdot 16 = 9138{,}8 \text{ грн.}$$

Звідси витрати на створення програмного продукту:

$$К_{ПО} = 171696 + 9138{,}8 = 180828{,}8 \text{ грн.}$$

Очікуваний період створення ПЗ:

$$T = \frac{t}{B_k \cdot F_p}, \text{ мес.} \qquad (3.14)$$

де B_k – число виконавців;

$F_р$ – місячний фонд робочого часу (при 40 годинному робочому тижні Fр=176 годин).

Витрати на створення програмного продукту:

$$T = 1168/(1 \cdot 176) = 6{,}6 \text{ міс.}$$

Висновки. Мобільний додаток має вартість 180828,8 грн. Ймовірний очікуваний час розробки – 6,6 місяці при стандартному 40-годинному робочому

тижні і 178-годинному робочому місяці. Цей термін пов’язаний з кількістю операторів і включає в себе час для дослідження та розробку алгоритму розв’язання задачі, розробку дизайну і створення документації. На розробку мобільного додатку буде витрачено 1168 людино-годин.

ВИСНОВКИ

Під час виконання кваліфікаційної роботи був розроблений мультиплатформений ігровий додаток, використовуючи сучасний ігровий двигун Unity.

Основним призначенням додатку є використання у рекреаційних цілях у вільний час. Також гра сприяє підвищеній соціалізації завдяки необхідності у живому супротивнику, шо додатково сприяє виникненню ефекту суперництва та покращенню мислительних здатностей, розвитку стратегічного мислення та загального зниження стресу через ігрову форму.

При створенні додатку був використані сучасні підходи до об’єктно орієнтованого програмування, а саме структура програми підводилася під принципи SOLID, запропоновані Майклом Фаєрсом на початку двохтисячних, а саме: принцип єдиної відповідальності(single responsibility principle),принцип відкритості/закритості(open-closed principle), принцип підстановки Лісков(Liskov substitution principle),принцип розділення інтерфейсів(interface segregation principle), принцип інверсії залежностей(dependency inversion principle). Усе це сприяє зрозумілості структури та відкритості додатку до підтримки існуючих та подальшого впровадження нових функцій.

Також під час виконання кваліфікаційної роботи у економічному розділі були проведені підрахунки для визначення трудомісткості розробленої інформаційної системи 1168 людино-годин, вартості роботи по її створенню 180828,8 грн та часу витраченого на створення 6.6 міс.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. .Zackariasson P. and Wilson T. The Video Game Industry: Formation, Present State, and Future. New York: Routledge, 2014, 282 p.

2. Luenendonk M. "The Gaming Industry – An Introduction". URL: https://www.cleverism.com/gaming-industry-introduction/(дата звернення: 15.04.2022)

3. Bloomberg Businessweek: magazine. Bloomberg L.P. 1994. No. 3392–3405. P. 58.

4. Arrington M. Zynga Takes $180 Million Venture Round From DST. URL:https://www.businessinsider.com/zynga-takes-180-million-venture-round-from-dst-others-cue-russian-mafia-jokes-2009-12(дата звернення: 17.04.2022)

5. Riddell, D. ESports: Global revenue expected to smash $1 billion by 2019. URL:https://edition.cnn.com/2016/05/29/sport/esports-revolution-revenue-audience-growth/index.html(дата звернення: 19.04.2022)

6. Witkowski W. Videogames are a bigger industry than movies and North American sports combined, thanks to the pandemic. URL:https://www.marketwatch.com/story/videogames-are-a-bigger-industry-than-sports-and-movies-combined-thanks-to-the-pandemic-11608654990(дата звернення: 23.04.2022)

7. Playing Video Games Offers Learning Across Life Span, Say Studies. URL:https://www.sciencedaily.com/releases/2008/08/080817223442.htm(дата звернення: 25.04.2022)

8. Playing Video Games Improves Eyesight. URL:https://www.livescience.com/1382-study-playing-video-games-improves-eyesight.html(дата звернення: 25.04.2022)

9. Gaming well: links between videogames and flourishing mental health.URL:https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3978245/(дата звернення: 25.04.2022)

10. System requirements for Unity. URL:https://docs.unity3d.com/Manual/system-requirements.html(дата звернення: 26.04.2022)

11. Dealessandri M. What is the best game engine: is Unity right for you?. URL: https://www.gamesindustry.biz/articles/2020-01-16-what-is-the-best-game-engine-is-unity-the-right-game-engine-for-you(дата звернення:27.04.2022)

12. Axon S. Unity at 10: For better—or worse—game development has never been easier. URL:https://arstechnica.com/gaming/2016/09/unity-at-10-for-better-or-worse-game-development-has-never-been-easier/(дата звернення: 27.04.2022)

13. C# Language Specification (5th ed.). Режим доступу:https://www.ecma-international.org/publications-and-standards/standards/ecma-334/

14. Baskar R. Design Goals of C#. URL:https://www.java-samples.com/showtutorial.php?tutorialid=1425(дата звернення:28.04.2022)

15. virtual (C# Reference)". URL:https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/virtual(дата звернення: 30.04.2022)

16. Singleton pattern. URL:https://refactoring.guru/design-patterns/singleton(дата звернення: 1.05.2022)

17. Mediator pattern. URL: https://refactoring.guru/design-patterns/mediator(дата звернення: 1.05.2022)

18. Observer pattern. URL: https://refactoring.guru/design-patterns/observer(дата звернення: 1.05.2022)

19. Order of execution for event functions. URL:https://docs.unity3d.com/Manual/ExecutionOrder.html(дата звернення: 3.05.2022)

20. Martin R. Clean Architecture: A Craftsman's Guide to Software Structure and Design. Pearson,2017. 432 p

ДОДАТОК А

КОД ПРОГРАМИ

Лістинг AudioManager.cs:

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

public class AudioManager : MonoBehaviour//Менеджер відповідальний за музику
{
    public static AudioManager Instance;
    public string musicKey;
    public string effectsKey;
    public string muteKey;
    private bool _isMuted;
    [SerializeField]
    private AudioMixer _audioMixer;
    [SerializeField]
    private AudioSource _musicSource;
    [SerializeField]
    private AudioSource _effectsSource;
    [SerializeField]
    private List<AudioClip> _audioClips;
    [SerializeField]
    private List<AudioClip> _musicList;

    private void OnEnable()
    {
        if (Instance != null)
        {
            Destroy(Instance.gameObject);
        }
        Instance = this;
    }

    private void Start()
    {
        SetAudio();
        DontDestroyOnLoad(this);
    }

    private void SetAudio()//Виставляє гучність відповідно до збереженних значень
    {
        _audioMixer.SetFloat(musicKey, GetSavedVolume(musicKey));
        _audioMixer.SetFloat(effectsKey, GetSavedVolume(effectsKey));
        _isMuted = Convert.ToBoolean(PlayerPrefs.GetInt(muteKey, 0));
        CheckForMute();
```

```
    }

    public float GetSavedVolume(string key)//Повертає збережені значення
    {
        float sound = 0;
        if (PlayerPrefs.HasKey(key))
        {
            sound = PlayerPrefs.GetFloat(key, 0);
        }
        return sound;
    }

    public void SetVolume(string key,float value)//Виставляє нове значення відповідно до
значення слайдера
    {
        Debug.LogError(value);
        _audioMixer.SetFloat(key, Mathf.Log10(value) * 20);
        PlayerPrefs.SetFloat(key, Mathf.Log10(value) * 20);
        PlayerPrefs.Save();
    }

    public void SetMuteState(bool isMuted)//Змінює статус мьюта
    {
        _isMuted = isMuted;
        PlayerPrefs.SetInt(muteKey, Convert.ToInt32(isMuted));
        PlayerPrefs.Save();
        CheckForMute();
    }

    private void CheckForMute()//Мьютить звуки або повертає до збереженних значень
    {
        if (_isMuted)
        {
            _audioMixer.SetFloat(musicKey, -80);
            _audioMixer.SetFloat(effectsKey, -80);
        }
        else
        {
            _audioMixer.SetFloat(musicKey, GetSavedVolume(musicKey));
            _audioMixer.SetFloat(effectsKey, GetSavedVolume(effectsKey));
        }
    }

    public bool IsMuted()//Повертає значення мьюту
    {
        return _isMuted;
    }

    public void PlaySound(Sounds sound)//Програє обраний звук один раз
    {
        _effectsSource.PlayOneShot(_audioClips[(int)sound]);
    }
```

```
    public void PlayMusic(Music music)//Починає грати вибрану музику
    {
        _musicSource.clip = _musicList[(int)music];
        _musicSource.Play();
    }

    public AudioClip GetAudioClip(Sounds sound)//Повертає усі звуки
    {
        return _audioClips[(int)sound];
    }
}

public enum Sounds//Звуки що є в проєкті
{
    Button,
    Slider,
    CloseWindow,
    OpenWindow,
    BuyCell,
    Build,
    Upgrade,
    Hire,
    Castle,
    Farm,
    Lambermill,
    Mine,
    Smith,
    NextTurn
}

public enum Music//Музика що є в проєкті
{
    MenuMusic,
    GameMusic
}
```

Лістинг CellsManager.cs:

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

public class CellsManager : MonoBehaviour
{
    public static CellsManager Instance;
    private HexCell _currentHex;
    [SerializeField]
    private List<HexCell> _player1HexCells;
    [SerializeField]
    private List<HexCell> _player2HexCells;
```

```
    private Dictionary<ResourceType, int> _player1Resources;
    private Dictionary<ResourceType, int> _player2Resources;
    private HexGrid _hexGrid;
    private int _player1CellPrice = 50;
    private int _player2CellPrice = 50;
    public int cellPrice = 50;

    private void OnEnable()
    {
        if (Instance)
        {
            Destroy(Instance.gameObject);
        }
        Instance = this;
        SetResources();
        _hexGrid = FindObjectOfType<HexGrid>();
        cellPrice = _player1CellPrice;
    }

    private void SetResources()//Заповнення словників з ресурсами гравців
    {
        _player1Resources =
            Enum.GetValues(typeof(ResourceType)).Cast<ResourceType>().ToDictionary(t => t, t =>
0);
        _player2Resources =
            Enum.GetValues(typeof(ResourceType)).Cast<ResourceType>().ToDictionary(t => t, t =>
0);
    }

    private void Start()
    {
        SetAvailablility(_player2HexCells, false);
        SetAvailablility(_player1HexCells, true);
        _hexGrid.TriangulateAll();
        EventManager.NextTurn += NextTurn;
    }

    private void NextTurn()//Наступний хід
    {
        GetCellsProduction(PlayerManager.Instance.currentPlayer);
        PlayerManager.Instance.ChangePlayer();
        ChangePlayers();
        _hexGrid.TriangulateAll();
        EventManager.UpdateResources?.Invoke();
    }

    private void ChangePlayers()// Хід переходить до нового гравця
    {
        if (PlayerManager.Instance.currentPlayer.Equals(Player.Player1))
        {
            SetAvailablility(_player2HexCells, false);
            SetAvailablility(_player1HexCells, true);
```

```
            _player1CellPrice = cellPrice;
            cellPrice = _player2CellPrice;
        }
        else
        {
            SetAvailablility(_player1HexCells, false);
            SetAvailablility(_player2HexCells, true);
            _player2CellPrice = cellPrice;
            cellPrice = _player1CellPrice;
        }
    }

    public void GetCellsProduction(Player player)//Збирає дохід обраного гравця
    {
        for (int i = 0; i < Enum.GetValues(typeof(ResourceType)).Length; i++)
        {
            if (player.Equals(Player.Player1))
            {
                _player1Resources[(ResourceType)i] += GetProductionIncome((ResourceType)i,
PlayerManager.Instance.currentPlayer);
            }
            else
            {
                _player2Resources[(ResourceType)i] += GetProductionIncome((ResourceType)i,
PlayerManager.Instance.currentPlayer);
            }
        }
        EventManager.UpdateResources?.Invoke();
    }

    public int GetProductionIncome(ResourceType resource, Player player)//Повертає дохіл
певного ресурсу
    {
        int income = 0;
        if (player.Equals(Player.Player1))
        {
            for (int i = 0; i < _player1HexCells.Count; i++)
            {
                income += _player1HexCells[i].production[resource];
            }
        }
        else
        {
            for (int i = 0; i < _player2HexCells.Count; i++)
            {
                income += _player2HexCells[i].production[resource];
            }
        }
        return income;
    }
```

```
    public void AddResource(ResourceType resourceType, int amount, Player player)//Додає ресурс до “гаманця” гравця
    {
        if (player.Equals(Player.Player1))
        {
            _player1Resources[resourceType] += amount;
        }
        else
        {
            _player2Resources[resourceType] += amount;
        }
    }

    public int GetProductionAmount(ResourceType production, Player player)//Повертає кількість певного ресурсу
    {
        if (player.Equals(Player.Player1))
        {
            return _player1Resources[production];
        }
        else
        {
            return _player2Resources[production];
        }
    }

    public void AddNewCell(HexCell hexCell, Player player)//Додає одну клітину гравцю
    {
        if (player.Equals(Player.Player1))
        {
            _player1HexCells.Add(hexCell);
        }
        else
        {
            _player2HexCells.Add(hexCell);
        }
    }

    public void ChangeOwner(Player player, HexCell cell)//Передача контролю для клітиною
    {
        if (player.Equals(Player.Player1))
        {
            _player2HexCells.Remove(cell);
            _player1HexCells.Add(cell);
        }
        else
        {
            _player1HexCells.Remove(cell);
            _player2HexCells.Add(cell);
        }
        CheckForGameEnd();
    }
```

```
    private void CheckForGameEnd()
    {
        if (_player1HexCells.Count == 0)
        {
            PlayerManager.Instance.SetWinner(Player.Player2);
        }
        else if (_player2HexCells.Count == 0)
        {
            PlayerManager.Instance.SetWinner(Player.Player1);
        }
    }

    public void SetCurrentHex(HexCell hexCell)//Вибір клітини
    {
        _currentHex = hexCell;
    }

    public HexCell GetCurrentHex()//Повертає обрану клітину
    {
        return _currentHex;
    }

    public bool CanBuy(Dictionary<ResourceType, int> price, int multiplier)//Перевіряє чи
достатньо ресурсів для покупки
    {
        foreach (KeyValuePair<ResourceType, int> pair in price)
        {
            if (GetProductionAmount(pair.Key, PlayerManager.Instance.currentPlayer) < pair.Value *
multiplier)
            {
                return false;
            }
        }
        return true;
    }

    public void BuySomething(Dictionary<ResourceType, int> price, int multiplier)//Витрачає
ресурси на покупку
    {
        foreach (KeyValuePair<ResourceType, int> pair in price)
        {
            AddResource(pair.Key, -pair.Value * multiplier, PlayerManager.Instance.currentPlayer);
        }
        EventManager.UpdateResources?.Invoke();
    }

    public void SetAvailablility(List<HexCell> cells, bool availability)//Змінює статус клітини на
доступний
    {
        for (int i = 0; i < cells.Count; i++)
        {
```

```
            cells[i].SetNeighborsAvailable(availability);
        }
    }

    public void HighlightCells(bool isActive)//Підсвічує підходящі клітини
    {
        List<HexCell> cells;
        if (PlayerManager.Instance.currentPlayer.Equals(Player.Player1))
        {
            cells = _player1HexCells;
        }
        else
        {
            cells = _player2HexCells;
        }
        for (int i = 0; i < cells.Count; i++)
        {
            cells[i].ChangeWalkableSprite(isActive);
        }
    }
}
```

Лістинг EventManager.cs:

```
using System;

public class EventManager
{
    public static Action UpdateResources;
    public static Action NextTurn;
    public static Action<HexCell> CellSelected;
    public static Action<HexCell> PlaceUnit;
}
```

Лістинг ParticlestManager.cs:

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

public class ParticlesManager : MonoBehaviour
{
    public static ParticlesManager Instance;
    [SerializeField]
    private List<ParticleSystem> _particles;

    private void OnEnable()
    {
        if (Instance != null)
        {
            Destroy(Instance.gameObject);
        }
```

```
        Instance = this;
    }

    public void PlayParticle(Particles particle, Vector3 position)
    {
        ParticleSystem particleSystem = _particles[(int)particle];
        particleSystem.transform.position = position;
        particleSystem.Play();
    }
}

public enum Particles
{
    Build,
    Death
}
```

Лістинг PlayerManager.cs:

```
using UnityEngine;

public class PlayerManager : MonoBehaviour
{
    public static PlayerManager Instance;
    public Player currentPlayer;
    public bool isMovingUnit;
    public UnitScreen unitScreen;

    private void OnEnable()
    {
        if (Instance)
        {
            Destroy(Instance.gameObject);
        }
        Instance = this;
    }

    private void Start()
    {
        AudioManager.Instance.PlayMusic(Music.GameMusic);
    }

    public void ChangePlayer()
    {
        if (currentPlayer.Equals(Player.Player1))
        {
            currentPlayer = Player.Player2;
        }
        else
        {
            currentPlayer = Player.Player1;
```

```
        }
        Debug.LogError(currentPlayer);
    }

    public void SetWinner(Player player)
    {
        Debug.LogError("Winner: " + player);
    }
}

public enum Player
{
    Player1,
    Player2
}
```

Лістинг PlayerManager.cs:

```
using DG.Tweening;
using UnityEngine;
using UnityEngine.UI;

public class UIManager : MonoBehaviour
{
    [SerializeField]
    private Button _menuButton;
    [SerializeField]
    private Button _nextTurnButton;
    [SerializeField]
    private Button _hireButton;
    [SerializeField]
    private Text _turnText;
    private Transform _turnTransform;
    private bool _isAnimating;

    private void Start()
    {
        _menuButton.onClick.AddListener(ShowMenu);
        _nextTurnButton.onClick.AddListener(NextTurn);
        _hireButton.onClick.AddListener(OpenHiringWindow);
        _turnTransform = _turnText.transform.parent;
        ShowWhichTurn();
    }

    private void ShowMenu()
    {
        if (!WindowManager.Instance.WindowIsOpen())
        {
            WindowManager.Instance.OpenWindow(WindowType.PauseWindow);
        }
    }
```

```
    private void NextTurn()
    {
        if (!_isAnimating)
        {
            EventManager.NextTurn?.Invoke();
            AudioManager.Instance.PlaySound(Sounds.NextTurn);
            ShowWhichTurn();
        }
    }

    private void ShowWhichTurn()
    {
        ChangeAnimatingStatus();
        Vector3 pos = _turnTransform.position;
        _turnText.text = PlayerManager.Instance.currentPlayer + "`s Turn";
        Tween tween = _turnTransform.DOMoveY(pos.y - 150, 1);
        tween.OnComplete(() => ResetTurnTextPosition());
        tween.SetEase(Ease.InOutBack);

    }

    private void ResetTurnTextPosition()
    {
        Tween tween = _turnTransform.DOMoveY(_turnTransform.position.y + 150, 1);
        tween.SetEase(Ease.InOutBack);
        tween.SetDelay(0.5f);
        tween.OnComplete(() => ChangeAnimatingStatus());
    }

    private void ChangeAnimatingStatus()
    {
        _isAnimating = !_isAnimating;
    }

    private void OpenHiringWindow()
    {
        if (!WindowManager.Instance.WindowIsOpen())
        {
            WindowManager.Instance.OpenWindow(WindowType.HiringWindow);
        }
    }

    private void Update()
    {
        if (Input.GetKeyDown(KeyCode.Space))
        {
            NextTurn();
        }
    }
}
```

Лістинг WindowManager.cs:

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

public class WindowManager : MonoBehaviour
{
    public static WindowManager Instance;
    private BasicWindow _currentWindow;
    [SerializeField]
    private PriceElement _priceElement;
    [SerializeField]
    private BasicWindow _buildingWindow;
    [SerializeField]
    private UpgradeWindow _upgradeWindow;
    [SerializeField]
    private SettingsWindow _settingseWindow;
    [SerializeField]
    private PauseWindow _pauseWindow;
    [SerializeField]
    private HiringWindow _hiringWindow;
    [SerializeField]
    private StartWindow _startWindow;
    public bool isHiring;


    private void OnEnable()
    {
        if (Instance)
        {
            Destroy(Instance.gameObject);
        }
        Instance = this;
    }

    private void Update()
    {
        if (Input.GetKeyDown(KeyCode.Escape))
        {
            if (!_currentWindow)
            {
                OpenWindow(WindowType.PauseWindow);
            }
        }
    }

    public bool WindowIsOpen()
    {
        if (_currentWindow)
        {
            return true;
```

```
    }
    return false;
}

public void OpenWindow(WindowType windowType)
{
    switch (windowType)
    {
        case WindowType.BuildingWindow:
            _currentWindow = _buildingWindow;
            break;
        case WindowType.UpgradeWindow:
            _currentWindow = _upgradeWindow;
            break;
        case WindowType.SettingsWindow:
            _currentWindow = _settingseWindow;
            break;
        case WindowType.PauseWindow:
            _currentWindow = _pauseWindow;
            break;
        case WindowType.HiringWindow:
            _currentWindow = _hiringWindow;
            break;
        case WindowType.StartWindow:
            _currentWindow = _startWindow;
            break;
    }
    _currentWindow?.OpenWindow();
}

public void CloseWindow()
{
    if (!_currentWindow)
    {
        if (_pauseWindow.gameObject.activeInHierarchy)
        {
            _currentWindow = _pauseWindow;
        }
    }
    _currentWindow.gameObject.SetActive(false);
    _currentWindow = null;
}

public void SetUpLayout(Dictionary<ResourceType, int> dict, Transform layout, int multiplier)
{
    foreach (KeyValuePair<ResourceType, int> pair in dict)
    {
        if (pair.Value != 0)
        {
            PriceElement priceElement = Instantiate(_priceElement, layout);
            priceElement.SetUpElement(pair.Key, pair.Value * multiplier);
        }
```

```
        }
    }

    public void SetUpSeparatedLayout(Dictionary<ResourceType, int> dict, Transform layout1,
Transform layout2)
    {
        foreach (KeyValuePair<ResourceType, int> pair in dict)
        {
            if (pair.Value > 0)
            {
                PriceElement priceElement = Instantiate(_priceElement, layout1);
                priceElement.SetUpElement(pair.Key, pair.Value);
            }
            else if (pair.Value < 0)
            {
                PriceElement priceElement = Instantiate(_priceElement, layout2);
                priceElement.SetUpElement(pair.Key, pair.Value);
            }
        }
    }
}

public enum WindowType
{
    BuildingWindow,
    UpgradeWindow,
    SettingsWindow,
    PauseWindow,
    HiringWindow,
    StartWindow
}
```

Лістинг HexCell.cs:

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

public class HexCell : MonoBehaviour
{
    public HexCoordinates coordinates;
    public UnitScript unitStanding;
    public Dictionary<ResourceType, int> production;
    public int level = 1;
    [SerializeField]
    public GameObject _player1Hex;
    [SerializeField]
    public GameObject _player2Hex;
    [SerializeField]
```

```
HexCell[] neighbors;
public CellStatus cellStatus;
private BuildingType _buildingType;
[SerializeField]
private Transform _humanBuildings;
[SerializeField]
private Transform _undeadBuildings;
[SerializeField]
private List<GameObject> _playerBuildings;
[SerializeField]
private SpriteRenderer _sprite;
[SerializeField]
private SpriteRenderer _walkableSprite;
[SerializeField]
private Text _priceText;

private Color _color;
[SerializeField]
private Color _unavaliableColor;
[SerializeField]
private Color _avaliableColor;
[SerializeField]
private Color _claimedColor;

private void OnEnable()
{
   production = BuildingsInfo.zeroProduction;
}

public HexCell GetNeighbor(HexDirection direction)
{
   return neighbors[(int)direction];
}

public void SetNeighbor(HexDirection direction, HexCell cell)
{
   neighbors[(int)direction] = cell;
   cell.neighbors[(int)direction.Opposite()] = this;
}

public Color GetCurrentColor()
{
   switch (cellStatus)
   {
      case CellStatus.Unavaliable:
         _color = _unavaliableColor;
         break;
      case CellStatus.Available:
         _color = _avaliableColor;
         break;
      case CellStatus.ClaimedPlayer1:
         _color = _claimedColor;
```

```
            break;
        }
        _sprite.color = _color;
        return _color;
    }

    public void SetNeighborsAvailable(bool avaliable)
    {
        for (int i = 0; i < neighbors.Length; i++)
        {
            if (neighbors[i])
            {
                if (neighbors[i].cellStatus == CellStatus.Unavaliable && avaliable)
                {
                    neighbors[i].cellStatus = CellStatus.Available;
                }
                else if (neighbors[i].cellStatus == CellStatus.Available && !avaliable)
                {
                    neighbors[i].cellStatus = CellStatus.Unavaliable;
                }
                neighbors[i].ChangePriceText(avaliable);
            }
        }
    }

    public void ChangePriceText(bool isActive)
    {
        _priceText.gameObject.transform.parent.gameObject.SetActive(isActive);
    }

    public void SetPrice()
    {
        _priceText.text = CellsManager.Instance.cellPrice.ToString();
    }

    public BuildingType GetBuildingType()
    {
        return _buildingType;
    }

    public void SetProduction()
    {
        switch (_buildingType)
        {
            case BuildingType.Castle:
                production = BuildingsInfo.castleProduction;
                break;
            case BuildingType.Farm:
                production = BuildingsInfo.farmProduction;
                break;
            case BuildingType.Lambermill:
                production = BuildingsInfo.lambermillProduction;
```

```
            break;
        case BuildingType.Smith:
            production = BuildingsInfo.smithProduction;
            break;
        case BuildingType.Mine:
            production = BuildingsInfo.mineProduction;
            break;
    }
}

public void Build(BuildingType buildingType)
{
    _buildingType = buildingType;
    GameObject building;
    GetOwnersBuildings();
    building = _playerBuildings[(int)_buildingType - 1];
    AnimateConstruction(building);
    ParticlesManager.Instance.PlayParticle(Particles.Build, transform.position);
    SetProduction();
}

private void AnimateConstruction(GameObject building)
{
    Vector3 scale = building.transform.localScale;
    building.transform.localScale = Vector3.zero;
    building.SetActive(true);
    building.transform.DOScale(scale, 0.5f);
}

public void RemoveBuilding()
{
    GetOwnersBuildings();
    _playerBuildings[(int)_buildingType - 1].SetActive(false);
    _buildingType = BuildingType.None;
    production = BuildingsInfo.zeroProduction;
}

public void ClaimCell(Player player)
{
    if (cellStatus.Equals(CellStatus.Available))
    {
        AnimateCell(player);
    }
    if (player.Equals(Player.Player1))
    {
        _player1Hex.SetActive(true);
        _player2Hex.SetActive(false);
        cellStatus = CellStatus.ClaimedPlayer1;
    }
    else
    {
        _player1Hex.SetActive(false);
```

```
            _player2Hex.SetActive(true);
            cellStatus = CellStatus.ClaimedPlayer2;
        }
        if (!_buildingType.Equals(BuildingType.None))
        {
            ChangeBuildingStyle(player);
        }
        ChangePriceText(false);
        SetNeighborsAvailable(true);
    }

    private void AnimateCell(Player player)
    {
        GameObject hex = new GameObject();
        if (player.Equals(Player.Player1))
        {
            hex = _player1Hex;
        }
        else
        {
            hex = _player2Hex;
        }
        Vector3 scale = hex.transform.localScale;
        hex.transform.localScale = Vector3.zero;
        hex.SetActive(true);
        hex.transform.DOScale(scale, 0.5f).SetEase(Ease.OutBack);
        for (int i = 0; i < neighbors.Length; i++)
        {
            if (neighbors[i])
            {
                if (!neighbors[i].cellStatus.Equals(CellStatus.Available) &&
!neighbors[i].cellStatus.Equals(CellStatus.Unavaliable))
                {
                    neighbors[i].transform.DOShakeScale(0.5f, 0.1f);//.DOPunchScale(Vector3.one * 0.1f,
0.5f)
                }
            }
        }
    }

    public void ChangeWalkableSprite(bool isActive)
    {
        _walkableSprite.gameObject.SetActive(isActive);
    }

    public void ShowUpgrade()
    {
        PlayBuildingSound();
        GameObject building = new GameObject();
        GetOwnersBuildings();
        building = _playerBuildings[(int)_buildingType - 1];
```

```
        building.transform.DOShakeScale(0.5f, 0.1f).OnComplete(() =>
WindowManager.Instance.OpenWindow(WindowType.UpgradeWindow));
    }

    public void UpgradeBuilding()
    {
        Invoke(nameof(AnimateUpgrade), 0.1f);
    }

    private void AnimateUpgrade()
    {
        _playerBuildings[(int)_buildingType - 1].SetActive(false);
        level++;
        GetOwnersBuildings();
        GameObject building = _playerBuildings[(int)_buildingType - 1];
        building.SetActive(true);
        Vector3 scale = building.transform.localScale;
        building.transform.localScale = Vector3.zero;
        building.transform.DOScale(scale, 0.5f);
        ParticlesManager.Instance.PlayParticle(Particles.Build, transform.position);
    }

    private void PlayBuildingSound()
    {
        string name = _buildingType.ToString();
        Sounds sound = (Sounds)Enum.Parse(typeof(Sounds), name);
        AudioManager.Instance.PlaySound(sound);
    }

    private void GetOwnersBuildings()
    {
        if (cellStatus.Equals(CellStatus.ClaimedPlayer1))
        {
            GetBuildingsList(Player.Player1);

        }
        else if (cellStatus.Equals(CellStatus.ClaimedPlayer2))
        {
            GetBuildingsList(Player.Player2);
        }
    }

    private void GetBuildingsList(Player player)
    {
        List<GameObject> buildings = new List<GameObject>();
        Transform buildingsParent;
        if (player.Equals(Player.Player1))
        {
            buildingsParent = _humanBuildings.GetChild(level - 1);
        }
        else
        {
```

```
            buildingsParent = _undeadBuildings.GetChild(level - 1);
        }
        for (int i = 0; i < buildingsParent.childCount; i++)
        {
            buildings.Add(buildingsParent.GetChild(i).gameObject);
        }
        _playerBuildings = buildings;
    }

    private void ChangeBuildingStyle(Player newOwner)
    {
        if (newOwner.Equals(Player.Player1))
        {
            GetBuildingsList(Player.Player2);
            _playerBuildings[(int)_buildingType - 1].SetActive(false);
            ParticlesManager.Instance.PlayParticle(Particles.Build, transform.position);
            GetBuildingsList(Player.Player1);
            _playerBuildings[(int)_buildingType - 1].SetActive(true);
        }
        else
        {
            GetBuildingsList(Player.Player1);
            _playerBuildings[(int)_buildingType - 1].SetActive(false);
            ParticlesManager.Instance.PlayParticle(Particles.Build, transform.position);
            GetBuildingsList(Player.Player2);
            _playerBuildings[(int)_buildingType - 1].SetActive(true);
        }
    }
}

public enum CellStatus
{
    Unavaliable,
    Available,
    ClaimedPlayer1,
    ClaimedPlayer2
}

public enum BuildingType
{
    None,
    Farm,
    Lambermill,
    Smith,
    Castle,
    Mine
}

public enum ResourceType
{
    Gold,
    Wood,
```

```
    Ore,
    Food
}
```

Лістинг CameraScript.cs:

```
using UnityEngine;

public class CameraScript : MonoBehaviour
{
    [SerializeField]
    private float _mainSpeed = 100.0f;
    [SerializeField]
    private float _scrollSpeed;
    [SerializeField]
    private float _minY;
    [SerializeField]
    private float _maxY;
    private float _totalRun = 1.0f;

    void Update()
    {
        if (WindowManager.Instance.WindowIsOpen())
        {
            return;
        }
        if (Application.platform.Equals(RuntimePlatform.Android))
        {
            MoveWithTouch();
            ZoomWithTouch();
        }
        else
        {
            MoveWithWASD();
            ZoomWithMouse();
        }
    }

    private void MoveWithTouch()
    {
        if (Input.touchCount == 1)
        {
            Touch touch = Input.GetTouch(0);
            Vector3 delta = touch.deltaPosition * 0.1f;
            transform.position += new Vector3(-delta.x, 0, -delta.y);
        }
    }

    private void ZoomWithTouch()
    {
        if (Input.touchCount == 2)
        {
```

```
        Touch touch0 = Input.GetTouch(0);
        Touch touch1 = Input.GetTouch(1);
        Vector2 prevPos0 = touch0.position - touch0.deltaPosition;
        Vector2 prevPos1 = touch1.position - touch1.deltaPosition;
        float oldMagnitude = (prevPos0 - prevPos1).magnitude;
        float newMagnitude = (touch0.position - touch1.position).magnitude;
        float diff = newMagnitude - oldMagnitude;
        Zoom(diff * -0.1f);
    }
}

private void ZoomWithMouse()
{
    if (Input.mouseScrollDelta != Vector2.zero)
    {
        Zoom(Input.mouseScrollDelta.y * -_scrollSpeed);
    }
}

private void Zoom(float zoom)
{
    Vector3 newPos = transform.position;
    float posY = newPos.y;
    posY += zoom;
    posY = Mathf.Clamp(posY, _minY, _maxY);
    newPos.y = posY;
    transform.position = newPos;
}

private void MoveWithWASD()
{
    Vector3 p = GetBaseInput();
    if (p.sqrMagnitude > 0)
    {
        _totalRun = Mathf.Clamp(_totalRun * 0.5f, 1f, 1000f);
       p = p * _mainSpeed;
        p = p * Time.deltaTime;
        Vector3 newPosition = transform.position;
        transform.Translate(p);
        newPosition.x = transform.position.x;
        newPosition.z = transform.position.z;
        transform.position = newPosition;
    }
}

private Vector3 GetBaseInput()
{
    Vector3 p_Velocity = new Vector3();
    if (Input.GetKey(KeyCode.W))
    {
        p_Velocity += new Vector3(0, 0, 1);
    }
```

```
        if (Input.GetKey(KeyCode.S))
        {
            p_Velocity += new Vector3(0, 0, -1);
        }
        if (Input.GetKey(KeyCode.A))
        {
            p_Velocity += new Vector3(-1, 0, 0);
        }
        if (Input.GetKey(KeyCode.D))
        {
            p_Velocity += new Vector3(1, 0, 0);
        }
        return p_Velocity;
    }
}
```

Лістинг UnitScreen.cs:

```
using UnityEngine;
using UnityEngine.UI;

public class UnitScreen : MonoBehaviour
{
    [SerializeField]
    private Text _nameText;
    [SerializeField]
    private Text _strengthText;
    [SerializeField]
    private Text _healthText;
    [SerializeField]
    private Text _movementText;

    public void SetName(string name)
    {
        _nameText.text = name;
    }

    public void SetHealth(int health)
    {
        _healthText.text = health.ToString();
    }

    public void SetStrength(int strength)
    {
        _strengthText.text = strength.ToString();
    }

    public void SetMovement(int currentMovement, int maxMovement)
    {
        _movementText.text = currentMovement + "/" + maxMovement;
    }
}
```

Лістинг UnitScript.cs:

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

public class UnitScript : MonoBehaviour
{
    [SerializeField]
    private ParticleSystem _selectionParticles;
    private HexCell _currentCell;
    private bool _isSelected;
    [SerializeField]
    private int _strength;
    [SerializeField]
    private int _health;
    [SerializeField]
    private int _movement;
    [SerializeField]
    private Image _healthbar;
    private bool _isRecentlySelected;
    [SerializeField]
    private float _speed;
    private UnitScreen _unitScreen;
    private int _movementLeft;
    private bool _isMoving;
    private int _currentHealth;
    public Player owner;

    private void Start()
    {
        EventManager.NextTurn += NewTurn;
        EventManager.CellSelected += SetCurrentHex;
        SetUp();
    }

    private void SetUp()
    {
        _unitScreen = PlayerManager.Instance.unitScreen;
        _movementLeft = _movement;
        _currentHealth = _health;
        _healthbar.fillAmount = 1;
    }

    private void NewTurn()
    {
        if (_isSelected)
        {
            Deselect();
```

```
        }
        _movementLeft = _movement;
    }

    private void Update()
    {
        if (Input.GetMouseButtonDown(0) && _isSelected && !_isRecentlySelected)
        {
            Deselect();
        }
    }

    private void SetCurrentHex(HexCell hexCell)
    {
        if (!_isMoving && _isSelected && _movementLeft > 0)
        {
            HexDirection[] hexDirection =
Enum.GetValues(typeof(HexDirection)).Cast<HexDirection>().ToArray();
            for(int i = 0;i < hexDirection.Length;i++)
            {
                if (hexCell.Equals(_currentCell.GetNeighbor(hexDirection[i])) &&
CellIsWalkable(hexCell))
                {
                    SetWalkableCells(false);
                    if (!hexCell.unitStanding)
                    {
                        GetNewCell(hexCell);
                    }
                    else if(hexCell.unitStanding.owner != owner)
                    {
                        Attack(hexCell);
                    }
                }
            }
        }
    }

    private void GetNewCell(HexCell hexCell)
    {
        _currentCell.unitStanding = null;
        hexCell.unitStanding = this;
        _currentCell = hexCell;
        _currentCell.unitStanding = this;
        StartCoroutine(Move());
        UpdateMovement(_movementLeft - 1);
    }

    IEnumerator Move()
    {
        float time = 0;
        _isMoving = true;
        while (Vector3.Distance(transform.position,AdjustedPosition()) > 0.1f)
```

```
        {
            transform.position = Vector3.Lerp(transform.position, AdjustedPosition(), time);
            Vector3 targetDirection = AdjustedPosition() - transform.position;
            transform.rotation =
Quaternion.Slerp(transform.rotation,Quaternion.LookRotation(targetDirection),Time.deltaTime);
            time += _speed *3 * Time.deltaTime;
            yield return null;
        }
        SetWalkableCells(true);
        Conquer();
        _isMoving = false;
    }

    private void SetWalkableCells(bool isActive)
    {
        HexDirection[] hexDirection =
Enum.GetValues(typeof(HexDirection)).Cast<HexDirection>().ToArray();
        for (int i = 0; i < hexDirection.Length; i++)
        {
            HexCell cell = _currentCell.GetNeighbor(hexDirection[i]);
            if (cell && CellIsWalkable(cell))
            {
                cell.ChangeWalkableSprite(isActive);
            }
        }
    }

    private void UpdateMovement(int newValue)
    {
        _movementLeft = newValue;
        _unitScreen.SetMovement(_movementLeft, _movement);
    }

    private Vector3 AdjustedPosition()
    {
        return _currentCell.transform.position + Vector3.up * 5;
    }

    private void Attack(HexCell cell)
    {
        StartCoroutine(AnimateAttack(cell));
    }

    IEnumerator AnimateAttack(HexCell cell)
    {
        _isMoving = true;
        float time = 0;
        Vector3 target = cell.unitStanding.transform.position;
        while (Vector3.Distance(transform.position, target) > 0.1f)
        {
            transform.position = Vector3.Lerp(transform.position,target, time);
            Vector3 targetDirection = target - transform.position;
```

```
            transform.rotation = Quaternion.Slerp(transform.rotation,
Quaternion.LookRotation(targetDirection), Time.deltaTime);
            time += _speed * 3 * Time.deltaTime;
            yield return null;
        }
        _isMoving = false;
        cell.unitStanding.TakeDamage(_strength, this);
        if (!cell.unitStanding)
        {
            GetNewCell(cell);
        }
        _movementLeft = 0;
        if (_isSelected)
        {
            StartCoroutine(Move());
            _unitScreen.SetMovement(_movementLeft, _movement);
        }
    }

    private void OnMouseDown()
    {
        if (CanSelect())
        {
            Debug.LogError("Selected");
            Select();
        }
    }

    private bool CanSelect()
    {
        if (owner != PlayerManager.Instance.currentPlayer)
        {
            return false;
        }
        return true;
    }

    private void Select()
    {
        PlayerManager.Instance.isMovingUnit = true;
        _isSelected = true;
        _isRecentlySelected = true;
        _selectionParticles.gameObject.SetActive(true);
        SetUpScreen();
        Invoke(nameof(ResetSelection), 0.5f);
        SetWalkableCells(true);
    }

    private void ResetSelection()
    {
        _isRecentlySelected = false;
    }
```

```
    private void Deselect()
    {
        Debug.LogError("Deselected");
        _isSelected = false;
        _selectionParticles.gameObject.SetActive(false);
        PlayerManager.Instance.isMovingUnit = false;
        _unitScreen.gameObject.SetActive(false);
        SetWalkableCells(false);
    }

    private void SetUpScreen()
    {
        _unitScreen.SetName(gameObject.name);
        _unitScreen.SetStrength(_strength);
        _unitScreen.SetHealth(_currentHealth);
        _unitScreen.SetMovement(_movementLeft, _movement);
        _unitScreen.gameObject.SetActive(true);
    }

    private bool CellIsWalkable(HexCell cell)
    {
        if (cell.cellStatus.Equals(CellStatus.ClaimedPlayer1) ||
cell.cellStatus.Equals(CellStatus.ClaimedPlayer2))
        {
            return true;
        }
        return false;
    }

    private void Conquer()
    {
        if ((_currentCell.cellStatus.Equals(CellStatus.ClaimedPlayer2) &&
owner.Equals(Player.Player1)) ||
            (_currentCell.cellStatus.Equals(CellStatus.ClaimedPlayer1) &&
owner.Equals(Player.Player2)))
        {
            _currentCell.ClaimCell(owner);
            CellsManager.Instance.ChangeOwner(owner, _currentCell);
        }
    }

    public void TakeDamage(int amount, UnitScript attacker)
    {
        _currentHealth -= amount;
        if (attacker)
        {
            attacker.TakeDamage(_strength, null);
        }
        SetHealth();
    }
```

```
    private void SetHealth()
    {
        _currentHealth = Mathf.Clamp(_currentHealth, 0, _health);
        _healthbar.fillAmount = (float)_currentHealth / _health;
        if (_currentHealth <= 0)
        {
            Die();
        }
        if (_isSelected)
        {
            _unitScreen.SetHealth(_currentHealth);
        }
    }

    private void Die()
    {
        if (_isSelected)
        {
            Deselect();
        }
        ParticlesManager.Instance.PlayParticle(Particles.Death, transform.position);
        gameObject.SetActive(false);
    }

    public void SetUpUnit(HexCell cell)
    {
        _currentCell = cell;
        cell.unitStanding = this;
        owner = PlayerManager.Instance.currentPlayer;
        transform.position = AdjustedPosition();
    }
}
```

Лістинг ResourceCounter.cs:

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

public class ResourceCounter : MonoBehaviour, IPointerEnterHandler,IPointerExitHandler
{
    [SerializeField]
    private ResourceType _productionType;
    [SerializeField]
    private Text _countText;
    [SerializeField]
    private Text _incomeText;

    private void Start()
    {
```

```
        SetCounterText();
        EventManager.UpdateResources += SetCounterText;
    }

    private void SetCounterText()
    {
        _countText.text =
CellsManager.Instance.GetProductionAmount(_productionType,PlayerManager.Instance.currentPla
yer).ToString();
    }

    private void OnDestroy()
    {
        EventManager.UpdateResources -= SetCounterText;
    }

    public void OnPointerEnter(PointerEventData eventData)
    {
        _incomeText.gameObject.SetActive(true);
        ShowIncome();
    }

    private void ShowIncome()
    {
        int income = CellsManager.Instance.GetProductionIncome(_productionType,
PlayerManager.Instance.currentPlayer);
        if (income > 0)
        {
            _incomeText.color = Color.green;
        }
        else
        {
            _incomeText.color = Color.red;
        }
        _incomeText.text = income.ToString();
    }

    public void OnPointerExit(PointerEventData eventData)
    {
        _incomeText.gameObject.SetActive(false);
    }

}
```

Лістинг ResourceCounter.cs:

```
using UnityEngine;
using UnityEngine.UI;

public class MainMenuManager : MonoBehaviour
{
    [SerializeField]
```

```
    private Button _playButton;
    [SerializeField]
    private Button _settingsButton;
    [SerializeField]
    private Button _exitButton;

    private void Start()
    {
        _playButton.onClick.AddListener(Play);
        _settingsButton.onClick.AddListener(OpenSettings);
        _exitButton.onClick.AddListener(ExitGame);
        AudioManager.Instance.PlayMusic(Music.MenuMusic);
    }

    private void Play()
    {
        Debug.LogError("Play");
        if (!WindowManager.Instance.WindowIsOpen())
        {
            AudioManager.Instance.PlaySound(Sounds.Button);
            WindowManager.Instance.OpenWindow(WindowType.StartWindow);
        }
    }

    private void OpenSettings()
    {
        if (!WindowManager.Instance.WindowIsOpen())
        {
            AudioManager.Instance.PlaySound(Sounds.Button);
            WindowManager.Instance.OpenWindow(WindowType.SettingsWindow);
        }
    }

    private void ExitGame()
    {
        AudioManager.Instance.PlaySound(Sounds.Button);
        Application.Quit();
    }
}
```

Решта файлів може бути переглянута на носії, що йде з кваліфікаційною роботою. Там можна ознайомитися з повним лістингом коду та запустити готовий додаток.

ДОДАТОК Б

ВІДГУК
керівника економічного розділу
на кваліфікаційну роботу бакалавра
на тему:
"Розробка мультиплатформеного ігрового додатку за допомогою Unity"
студента групи 122-18-2 Тріска Антона Євгеновича

Керівник економічного розділу
доц. каф. ПЕП та ПУ, к.е.н **Л.В. Касьяненко**

ДОДАТОК В

ПЕРЕЛІК ФАЙЛІВ НА ДИСКУ

Ім’я файлу	Опис
Пояснювальні документи	
Кваліфікаційна робота Тріско.docx	Пояснювальна записка до кваліфікаційної роботи. Документ Word.
Кваліфікаційна робота Тріско.pdf	Пояснювальна записка до кваліфікаційної роботи в форматі PDF
Програма	
Тріско.rar	Архів. Містить коди програми і скомпільовану програму
Презентація	
Тріско.ppt	Презентація кваліфікаційної роботи

ВІДГУК

на кваліфікаційну роботу бакалавра
на тему:
"Розробка мультиплатформеного ігрового додатку за допомогою Unity"
студента групи 122-18-2 Тріска Антона Євгеновича

В результаті виконання даної кваліфікаційної роботи був розроблений мультиплатформений ігровий додаток на основі ігрового двигуна Unity. Актуальність розробленого програмного продукту обумовлена зростанням ігрової індустрії, особливо враховуючи збільшення вільного часу у людей на фоні пандемії COVID-19. Для реалізації програмного забезпечення була використана мова програмування C#. В основі додатку лежить ігровий двигун Unity.

Працездатність представленої програми підтверджена налагоджувальними випробуваннями та тестуванням програми.

В економічному розділі визначено трудомісткість розробленої кваліфікаційної роботи проведений підрахунок вартості роботи по створенню програми та розраховано час на його створення.

Тема кваліфікаційної роботи безпосередньо пов'язана з об'єктом діяльності бакалавра за напрямом підготовки 122 «Комп’ютерні науки».

Оформлення пояснювальної записки до кваліфікаційної роботи проекту виконано відповідно до стандартів на програмну документацію. Недоліком розглянутої роботи є невелика кількість пояснювальних коментарів і недостатня кількість деталей щодо алгоритму функціонування програмного забезпечення.

Кваліфікаційна робота виконана самостійно та заслуговує оцінки 84 бали, а студент Тріско Антон Євгенович присвоєння йому кваліфікації бакалавра за спеціальністю «бакалавр з комп’ютерних наук».

Керівник кваліфікаційної роботи
доцент каф. ПЗКС, к.т.н. **С.Д. Приходченко**

РЕЦЕНЗІЯ
на кваліфікаційну роботу бакалавра
на тему:
"Розробка мультиплатформеного ігрового додатку за допомогою Unity"
студента групи 122-18-2 Тріска Антона Євгеновича

Кваліфікаційну роботу на тему «Розробка мультиплатформеного ігрового додатку за допомогою Unity» виконаний в повному обсязі, відповідно до технічного завдання.

Мета кваліфікаційної роботи: Метою кваліфікаційної роботи є аналіз стану ігрової індустрії та розробка мультиплатформеного ігрового додатку за допомогою ігрового двигуна Unity.

У пояснювальній записці розглянуто необхідність створення і сфера застосування розробленої системи, виконано постановку завдання, опис вхідних і вихідних даних, розроблено інформаційне забезпечення системи, наведені загальні відомості про додаток, визначені джерела, використані при розробці.

Вважаю завдання і зміст кваліфікаційної роботи відповідним для перевірки ступеня підготовленості Тріска А.Є. за напрямом 122 Комп'ютерні науки.

Список літератури, наведений в роботі, налічує 20 джерел, що свідчить про вміння автора працювати з літературою та іншими джерелами інформації. Якість оформлення кваліфікаційної роботи можна визнати добре, з незначними недоліками.

Кваліфікаційну роботу виконано самостійно та заслуговує оцінки «добре», а студент Тріско Антон Євгенович заслуговує присвоєння йому кваліфікації «бакалавр з комп'ютерних наук».

Рецензент кваліфікаційної роботи