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 *Куйбіди Данила Віталійовича*
(ПІБ)

академічної групи *121-18-1*
(шифр)

спеціальності *121 Інженерія програмного забезпечення*
(код і назва спеціальності)

освітньої програми *Інженерія програмного забезпечення*
(назва освітньої програми)

на тему: *Розробка 2D платформеру з використанням зберігання даних на сервері за допомогою Unity 2D.*

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 *121-18-1* (група) Куйбіди Данила Віталійовича (прізвище та ініціали)

тема кваліфікаційної роботи *Розробка 2D платформеру з використанням зберігання даних на сервері за допомогою Unity 2D.*

затверджена наказом ректора НТУ «ДП» від №

Розділ	Зміст виконання	Термін виконання
Спеціальний	***На основі матеріалів виробничої 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 _____ *Куйбіда Д.В.*
(підпис) (прізвище, ініціали)

Дата видачі завдання: *14.01.2022 р.*

Термін подання кваліфікаційної роботи до ЕК: *11.06.2022 р.*

РЕФЕРАТ

Пояснювальна записка: 89 с., 28 рис., 3 дод., 21 джерело.

Об'єкт розробки: 2D платформер з використанням зберігання даних на сервері за допомогою Unity 2D.

Мета кваліфікаційної роботи: розробка 2D платформеру з використанням зберігання даних на сервері за допомогою Unity 2D.

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, розробляється 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 інформаційної системи, що дозволяє користувачам знаходити необхідний товар в одному місці з багатьох магазинів, порівнювати його характеристики відносно інших товарів, відслідковувати динаміку зміни ціни на протязі певного періоду часу, та обирати найвигіднішу пропозицію, яку пропонують інтернет магазини.

Актуальність програмного продукту визначається великим ростом популярності електронної комерції, а особливо сфери продаж, та тим, що сьогодні багато людей намагаються заощадити час та кошти і дана система допомагатиме користувачам в цьому.

Список ключових слів: ІГРОВИЙ ДОДАТОК, ПРОГРАМА, WINDWS, КОРИСТУВАЧ, ІНФОРМАЦІЙНА СИСТЕМА.

ABSTRACT

Explanatory note: 89 p., 28 figs., 3 appx., 21 sources.
Object of development: 2D platformer with collecting data on the server for the help of Unity 2D.
Meta-qualification work: development of a 2D platformer with victorious data collection on the server for the help of Unity 2D.
At the beginning of the meeting, the analysis of the current state of the problem is being clarified, the setting of the task, the meta-qualification of work and the task of zastosuvannya are being clarified, the actuality of those is being clarified.
At the first division, further research is carried out on the subject area and the main decisions, the relevance of the task and recognition of the development is determined, the setting of the task is determined.
From another distribution, a platform for distribution is collected, software design and development is reviewed, a description of the algorithm and structure of the system functioning is given, input and output data are determined, characteristics of the warehouse of parameters are specified in technical specifications, and the robot of the program is described.
In the economic distribution, the laboriousness of the developed software product is determined, the work is carried out according to the schedule and the cost is an hour for each creation.
Практичне значення полягає у розробці інформаційної системи, що дозволяє користувачам знаходити необхідний товар в одному місці з багатьох магазинів, порівнювати його характеристики відносно інших товарів, відслідковувати динаміку зміни ціни на протязі певного періоду часу, та обирати найвигіднішу пропозицію, яку пропонують інтернет магазини.
The relevance of the software product is due to the great growth in the popularity of electronic commerce, and especially in the field of sales, because today a lot of people are trying to save a lot of money and a system is given to help koristuvachs in cioma.
List of keywords: GAME ADDITION, PROGRAM, WINDWS, KORISTUVACH, INFORMATION SYSTEM.

ЗМІСТ

СПИСОК УМОВНИХ ПОЗНАЧЕНЬ

JSON	JavaScript Object Notation, текстовий формат обміну даними
ІС	Інформаційна система;
ОС	Операційна система;
ПК	Персональний комп’ютер;
ПЗ	Програмне забезпечення;
ІТ	Інформаційні технології;
ПД	Програмний додаток.

ВСТУП

Ігри завжди були, є, і будуть частиною життя підлітків та навіть дорослих людей. В інтернеті знаходиться дуже багато площадок де можна придбати ту чи іншу гру. Там можно знайти від футболу до шутерів. Дуже багато 3D ігор. Також є і 2D ігри, але вони не мають таку популярність як 3D.

Після 2019 року, коли почалася пандемія, багато шкіл та університетів перейшли на дистанційне навчання. І звісно що у школярів та студентів, котрі почали навчатися дистанційно, з'явилося багато вільного часу, а це значить що вони почали більше грати у комп'ютерні ігри. Через це виріс попит на мобільні та комп'ютерні ігри.

Серед всіх жанрів я виділив той, що, як на мою думку, дуже специфічний жанр, тому що не кожному він до вподоби. Це жанр 2D платформер[13] у жанрі Pixel Art Games. Прикладами таких ігор можуть бути Donkey Kong, Pac-Man. Цей жанр походить з далеких часів коли ігри запускалися на таких машинах як Arcade Pro.

Основою таких ігор є сюжет. За це люди й люблять цей жанр. Прикладом гри з крутим та цікавим сюжетом може бути така гра як Blasphemous.

Переваги щодо цього жанру це низькі технічні характеристики компьютера, а саме - у вас немає потреби купувати дорогі комплектуючі для вашого персонального компьютера, задля того щоб пограти.

Якщо повернутися на тему актуальності, то ми побачимо що в Україні не так і багато було видано таких ігор. Через це я вибрав тему кваліфікаційної роботи "2D платформер з використанням зберігання даних на сервері за допомогою Unity 2D". Завдання цієї кваліфікаційної роботи та об'єкт його діяльності безпосередньо пов'язані з напрямом підготовки "Інженерія програмного забезпечення" та відповідає узагальненій тематиці кваліфікаційних робіт.

РОЗДІЛ 1

АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1. Загальні відомості з предметної галузі

Розробка відеоігор – це процес розробки відеоігор. Ці зусилля бере на себе розробник, починаючи від окремої людини і закінчуючи міжнародною командою, розкиданою по всьому світу. Розробники відеоігор, також відомі як розробники ігор, відповідають за проектування та розробку відеоігор для ПК, консолей і мобільних додатків. Їхня робота полягає в тому, щоб закодувати базовий двигун на основі ідей команди дизайнерів. Вони також можуть брати участь у дизайні персонажів, дизайні рівнів, анімації та модульному тестуванні. Розробка традиційних комерційних комп'ютерних і консольних ігор зазвичай фінансується видавцем, і для завершення може знадобитися кілька років. Інді-ігри зазвичай займають менше часу та грошей і можуть бути створені окремими особами та меншими розробниками. Незалежна ігрова індустрія була на підйомі, чому сприяло зростання доступного програмного забезпечення для розробки ігор, такого як платформа Unity або Unreal Engine, а також нові системи онлайн-розповсюдження, такі як Steam і Uplay, а також ринок мобільних ігор для Android і пристрої iOS.

Згідно з дослідженням, загальна аудиторія ігор, над якими працюють українці, становить понад 770 мільйонів користувачів. Найбільшою аудиторією були великі компанії, такі як Plarium, Ubisoft і Wargaming. Більшість розробників ігор працюють над іграми для мобільних платформ, найчастіше для iOS та Android.

Більше половини компаній створюють або планують створювати ігри для пристроїв VR/AR. Згідно з опитуванням, Unity залишається найпопулярнішою платформою для розробки ігор. Найпопулярнішими жанрами в розробці українських ігор є Action та Adventure.

Коли йдеться про міжнародні IT-компанії та компанії з відеоігор, Україна вважається одним із найкращих варіантів аутсорсингу. Цей регіон

відомий як джерело талановитих та кваліфікованих розробників. З кожним роком все більше компаній звертають увагу на цю східноєвропейську країну, адже вона багата на кваліфікованих IT-фахівців. Низькі ціни на забудову роблять цей напрямок ще більш привабливим. Багато представників європейських ігрових компаній щороку відвідують конференції та з'їзди в Україні, щоб знайти та найняти досвідчених партнерів. Сьогодні ця країна є одним з найпопулярніших і вигідних варіантів аутсорсингу.

Іноземні компанії мають великий інтерес до найму та аутсорсингу українських інженерів, дизайнерів та розробників. Тисячі експертів стають партнерами великих європейських компаній і співпрацюють для створення видатних продуктів, які досягають неймовірного успіху. У цій статті ми склали список компаній з відеоігор, які можуть похвалитися кваліфікованою робочою силою, яка приваблює компанії з усього світу. Ці компанії розвивають як український, так і міжнародний ринок ігор.

Іноземні компанії, які відкривають свої офіси в Україні, роблять це не дарма. Українські працівники відомі як креативні, віддані своїй справі, високоосвічені та кваліфіковані. Більше того, вони дуже високо цінують трудову етику, тому великі студії вважають співпрацю з професіоналами з Києва та інших міст вигідною у всіх сенсах. З України походять такі хіти ігрового світу, як World of Tanks, Warface, Assassin's Creed, що є найкращим доказом майстерності місцевих розробників.

1.2. Призначення розробки та галузь застосування

Протягом останнього десятиліття ігри, можливо, стали найбільшою формою інформаційних систем дозвілля (ІС). Проте сьогодні ігри все частіше використовуються для різноманітних інструментальних цілей. Хоча ігри привернули значну дослідницьку увагу протягом останнього десятиліття, науково-дослідницька література розсіяна, і досі немає чіткого та надійного

розуміння того, чому використовуються ігри та як вони поміщені в усталений утилітарно-гедонічний континуум. інформаційних систем. Щоб усунути цю прогалину, ми провели мета-аналіз кількісної літератури, який досліджував причини використання ігор. Крім того, ми порівняли результати в іграх, які призначені як для дозвілля, так і для інструментального використання. Незважаючи на те, що ігри, як правило, розглядаються як вершина гедонічно орієнтованих ІС, наші результати показують, що задоволення та корисність є однаково важливими детермінантами їх використання (хоча їхня остаточна роль різниться між типами ігор). Таким чином, можна стверджувати, що ігри є багатоцільовими , які, тим не менш, покладаються на гедонічні фактори, навіть у гонитві за інструментальними результатами. Це дослідження вносить свій внесок у теоретичне й емпіричне розуміння багатоцільових ІС та способів їх використання та покращує наше теоретичне та емпіричне розуміння.

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, випусковою кафедрою, та затверджується наказом ректора.

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

- освітня програма 121 «Інженерія програмного забезпечення»;
- навчальний план та графік навчального процесу;
- наказ ректора Національного технічного університету «Дніпровська політехніка» № 317 від 07.06.2021 р;
- завдання на кваліфікаційну роботу на тему “Розробка 2D платформеру з використанням зберігання даних на сервері за допомогою Unity 2D”.

1.4. Постановка завдання

Завдання даної кваліфікаційної роботи є розробка 2D платформеру з використанням зберігання даних не сервері за допомогою Unity 2D.

Загальні характеристики розробки проекту мають бути:

- Можливість користувача зареєструватися для того щоб почати гру
- Можливість користувача увійти до аккаунту для того щоб почати гру
- Збереження даних та відправка їх на сервер
- Набрати задану кількість балів для того щоб перемогти
- Написання програмного коду

Поставлена задача може бути досягнута при виконанні наступних вимог:

- вивчення предметної області завдання;
- проведення порівняльної характеристики можливостей схожих ігор;
- вибір платформи розробки;
- написання програмного коду;
- розробка довідника для пояснення користування додатком.

У даній грі буде використано веб-додаток Firebase, а саме його компонент для збереження даних користувача, таких як пошта, пароль до акаунту та різні ігрові дані.

Кінцевий продукт має бути 2D грою, а саме - 2D платформером із збереженням даних на сервері за допомогою Unity 2D. Для першого запуску якого нам обов’язково треба доступ до інтернету, щоб користувач міг зберегти свій ігровий прогрес. У випадку якщо у користувача немає інтернету, гра не буде почата до того моменту як не з’явиться зв’язок з інтернетом.

1.5. Вимоги до програми або програмного виробу

1.5.1. Вимоги до функціональних характеристик

Кінцевий продукт повинен дотримуватися наступних функціональних вимог:

- Інтуїтивно зрозумілий інтерфейс користувача;
- Дані повинні вводитися користувачем на клавіатурі;
- Дані зберігаються на сервері;

1.5.2. Вимоги до інформаційної безпеки

Основні вимоги до інформаційної безпеки:

- Конфіденційність інформації;
- Цілісність даних;
- Доступність інформації.

Під час роботи додатку є тільки один режим - авторизований користувач.

Додаток запаковано в один файл, крім цього у користувача немає необхідності реєструватися і створювати аккаунт всередині програми. З цієї причини ніяких вимог до інформаційної безпеки немає.

Користувач повинен мати наступні функції:

- Реєстрацію та логін
- Вхід до акаунту;
- Вихід з акаунту.

1.5.3. Вимоги до складу та параметрів технічних засобів

Для підтримки ПЗ на ОС Windows, слід дотримуватися таким технічним вимогам:

- ОС Windows 7 або вище;
- Доступно не менше 1 ГБ вільного місця;
- Доступно 2 та більше ГБ оперативної пам’яті.

1.5.4. Вимоги до інформаційної та програмної сумісності

При збірці проекту з’являється папка із готовим ПЗ, яку користувач може запустити на ПК із ОС Windows 7 та вище. Для того щоб запустити ПЗ не треба ніяких додатків або додаткових програм.

Версії ОС Windows які можуть запустити ПЗ:

- Windows 7;
- Windows 8.1;
- Windows 10;
- Windows 11.

РОЗДІЛ 2
ПРОЕКТУВАННЯ ТА РОЗРОБКА ПРОГРАМНОГО ПРОДУКТУ

2.1 Функціональне призначення програми

Результатом даної кваліфікаційної роботи має бути комп'ютерна гра для платформи Windows з мережевим кодом за для відправки та перевірки імейлу та пароля користувача. Через це даний додаток потребує з'єднання до інтернету.

Основний функціонал додатка полягає у доланні ворогів та проходження рівнів.

Крім цього, для кращої і якісної роботи програми, повинен бути присутнім наступний додатковий функціонал:

– Збереження даних на сервер.

2.2. Опис застосованих математичних методів

Під час проектування та розробки даної інформаційної системи використовувалися лише прості арифметичні дії. Математичні методи не використовувалися.

2.3. Опис використаної архітектури та шаблонів проектування

Під час проєктування ігрового додатку було прийнято рішення використовувати Unity.

Unity — це 2D та 3D ігровий движок, який існує з 2005 року. Розроблений Unity Technologies, він був створений для того, щоб надати більшій кількості розробників доступ до інструментів розробки ігор, що в ті часи було новим підприємством. Протягом свого довгого терміну служби двигун різко змінився та розширився, зуміючи йти в ногу з найновішими практиками та технологіями.

Навіть сьогодні ігровий движок зосереджений на тому, щоб забезпечити найнадійніший набір інструментів для індустрії розробки ігор, а також зробити його якомога легшим для розробників ігор будь-якого рівня

кваліфікації (так, включаючи початківців розробники). Вони також розширили свій охоплення в інших галузях, зосередившись на розробці 3D в режимі реального часу, що робить його одним з найпотужніших доступних двигунів.

Плюси використання Unity:

- Unity чудово підходить для кросплатформної розробки та багатоплатформних ігор.
- Як повідомляється, магазин активів також чудовий у порівнянні з іншими платформами. В основному, технічна підтримка Unity ефективна.
- Що стосується візуальної платформи, Unity відмінно підходить. Але ми не можемо оскаржити те саме у випадку невізуальної платформи. Це може відрізнятися з точки зору застосування та використання програми.

Мінуси використання Unity:

- Документація щодо кількох функцій досить застаріла, а в деяких випадках і зовсім відсутня.
- Навіть такі технології, як текстури візуалізації, профайлер і підтримка буфера трафаретів, які є поширеними в багатьох безкоштовних інструментах, все ще стоять за платним екраном у 1500 доларів.
- Двигун Terrain і Movie Texture взагалі не хороші, і вони потребують багато зусиль без причини.

Найважливіший ігровий об’єкт – GameObject

Практично все у вашій сцені є GameObject. Подумайте про System.Object у .NET Framework. Від нього походять майже всі види. Така ж концепція

стосується GameObject. Це базовий клас для всіх об’єктів у вашій сцені Unity. Усі об'єкти, показані на малюнку 2.1 , є похідними від GameObject.

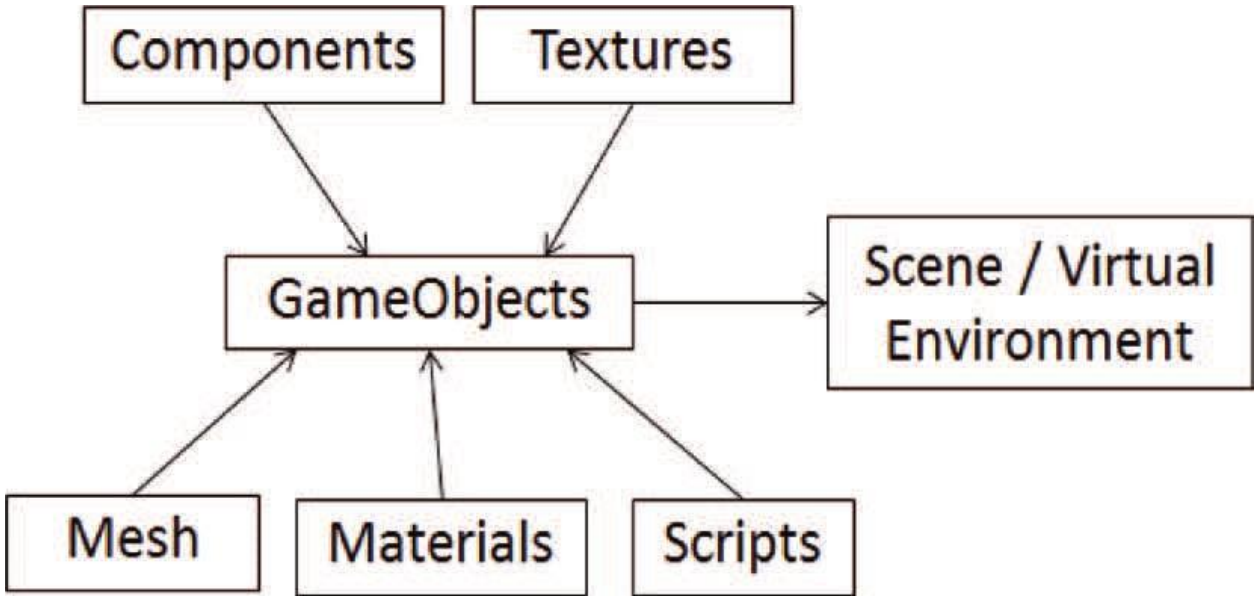

Рис 2.1. – GameObject

GameObject досить простий, оскільки він відноситься до вікна інспектора. На малюнку 6 можна побачити, що до сцени додано порожній GameObject; відзначте його властивості в панелі Inspector. GameObject за замовчуванням не мають візуальних властивостей. На даний момент це просто порожній об’єкт, малюнок 2.2.

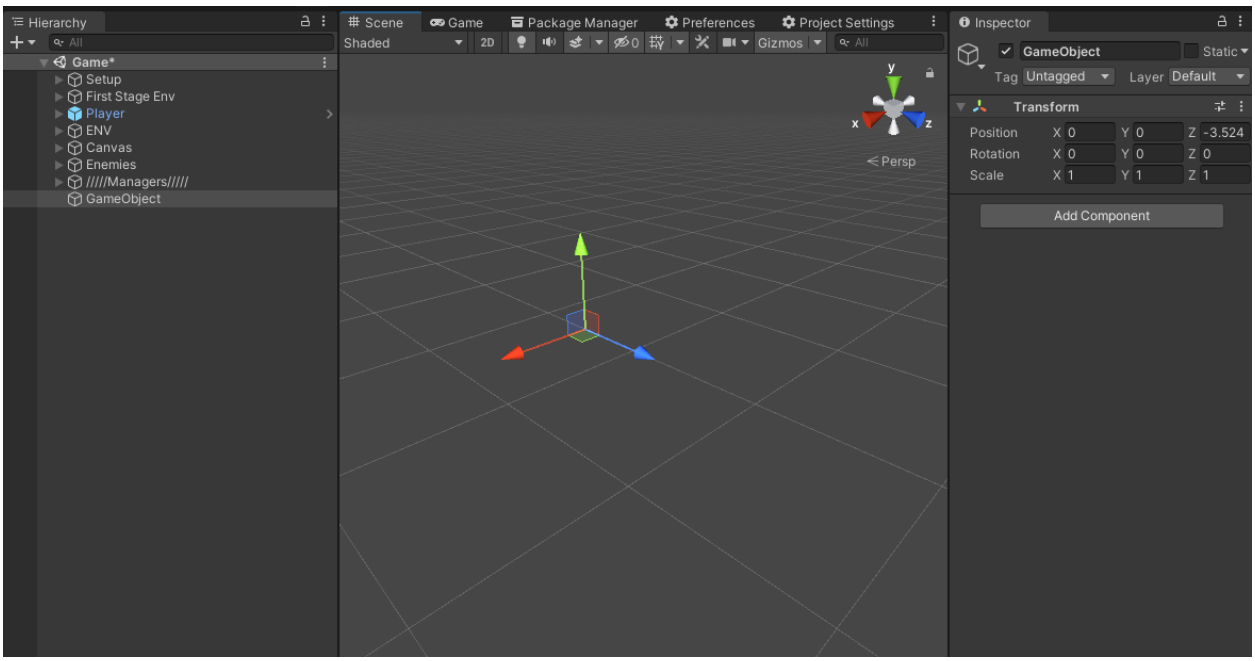

Рис 2.2. – GameObject на сцені

Написання коду

У базовому класі GameObject, який ми використовуємо, є багато функціональних можливостей. Основними методами є ті, які Unity буде викликати, якщо вони існують у вашому класі. Є кілька методів, які можна викликати. Хоча існує багато методів, зазвичай ви використовуєте лише декілька. Ось методи які дуже часто використовуться в коді для реалізації у класах, які стосуються послідовності подій для класів, похідних від MonoBehavior:

Awake: викликається, коли сценарій вперше завантажується або коли створюється екземпляр об’єкта, до якого він приєднаний. Він викликається лише один раз для кожного сценарію і лише після ініціалізації інших об’єктів.

Функція Update визначена в класі MonoBehaviour і запускатиметься автоматично в кожному кадрі гри, якщо MonoBehaviour активний. Коли ми створюємо новий скрипт в Unity, за замовчуванням ми отримуємо вже написаний код. У цьому коді визначено клас програмування, який називається рівним імені, яке ми дали скрипту, і який розширює або успадковує його поведінку MonoBehaviour, це простими словами означає,

що наш скрипт сам по собі є MonoBehaviour або окремим випадком MonoBehaviour [1].

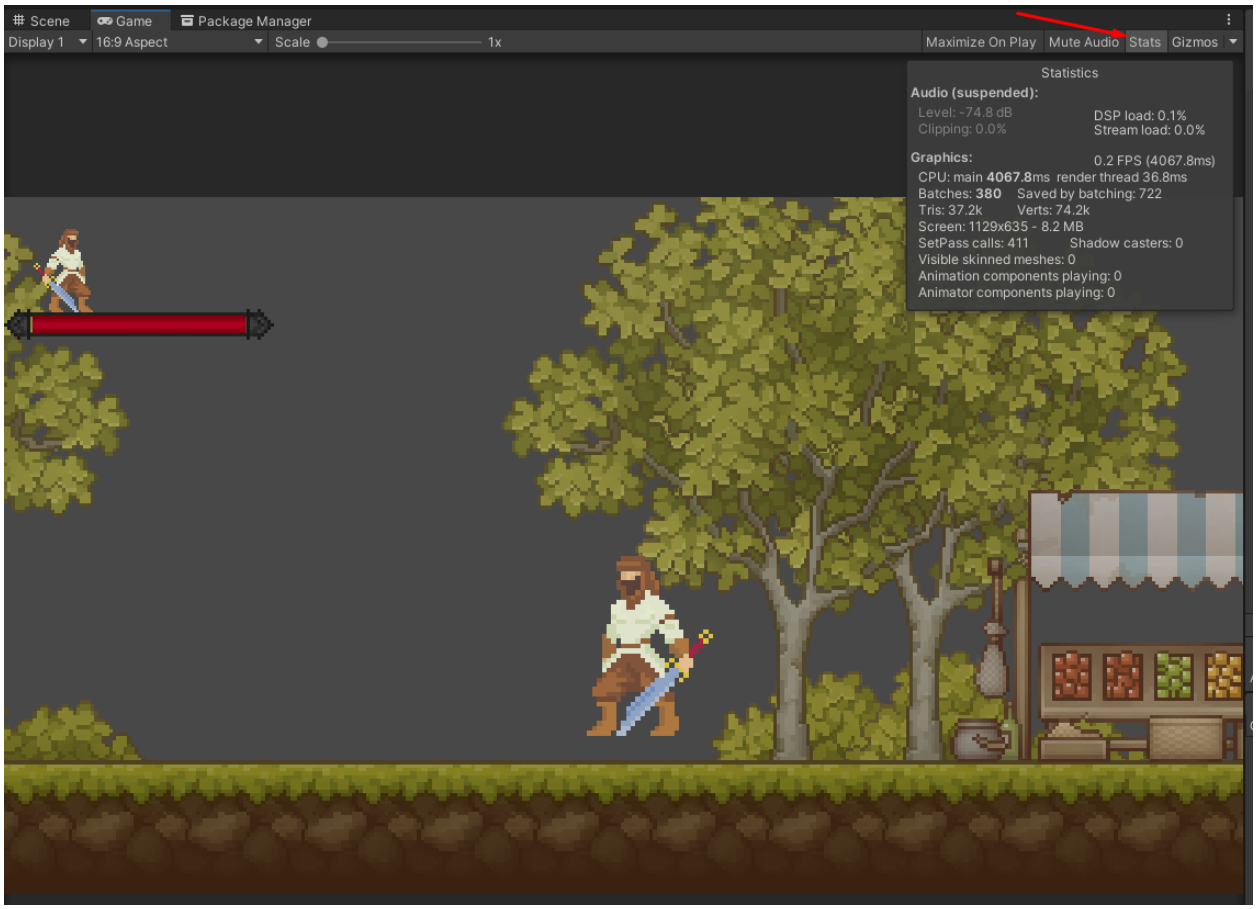

Рис 2.3. – панель Draw Calls [17]

Fixed Update виконується з фіксованим часовим кроком, який майже завжди буде відрізнятися від змінної швидкості оновлення. Це означає, що Fixed Update може бути викликано один раз під час кадру, двічі або не викликати взагалі, залежно від швидкості гри..

Частота виконання FixedUpdate задається у панелі ProjectSettings -> Time -> FixedUpdate на малюнку 2.4.

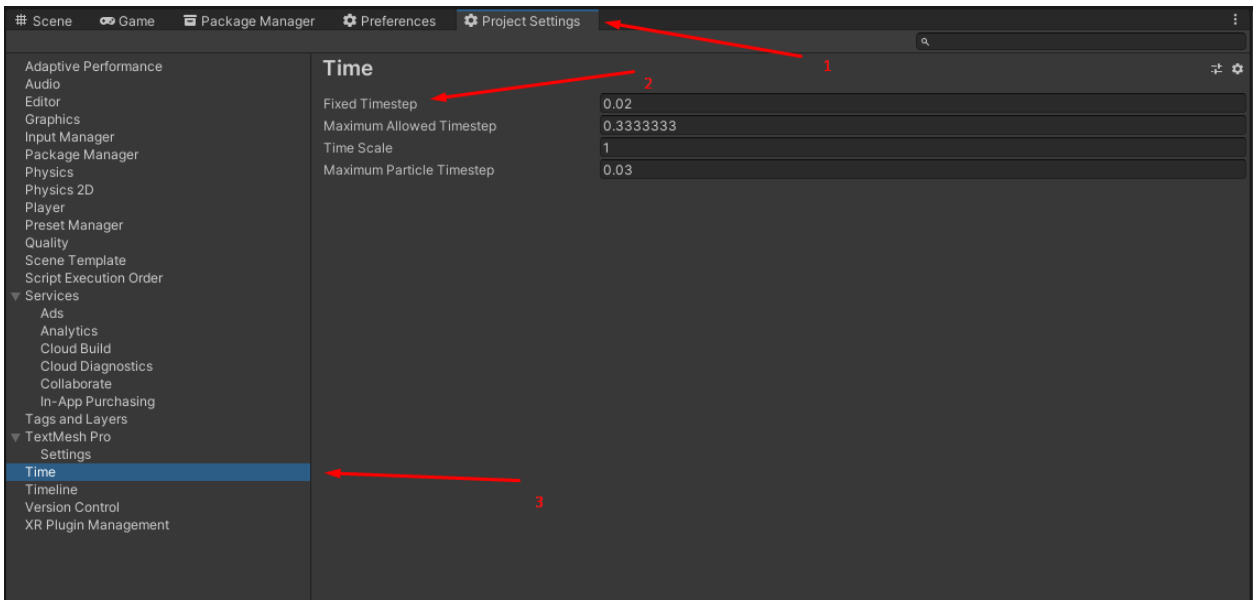

Рис 2.4. – де знаходиться частота виконання FixedUpdate

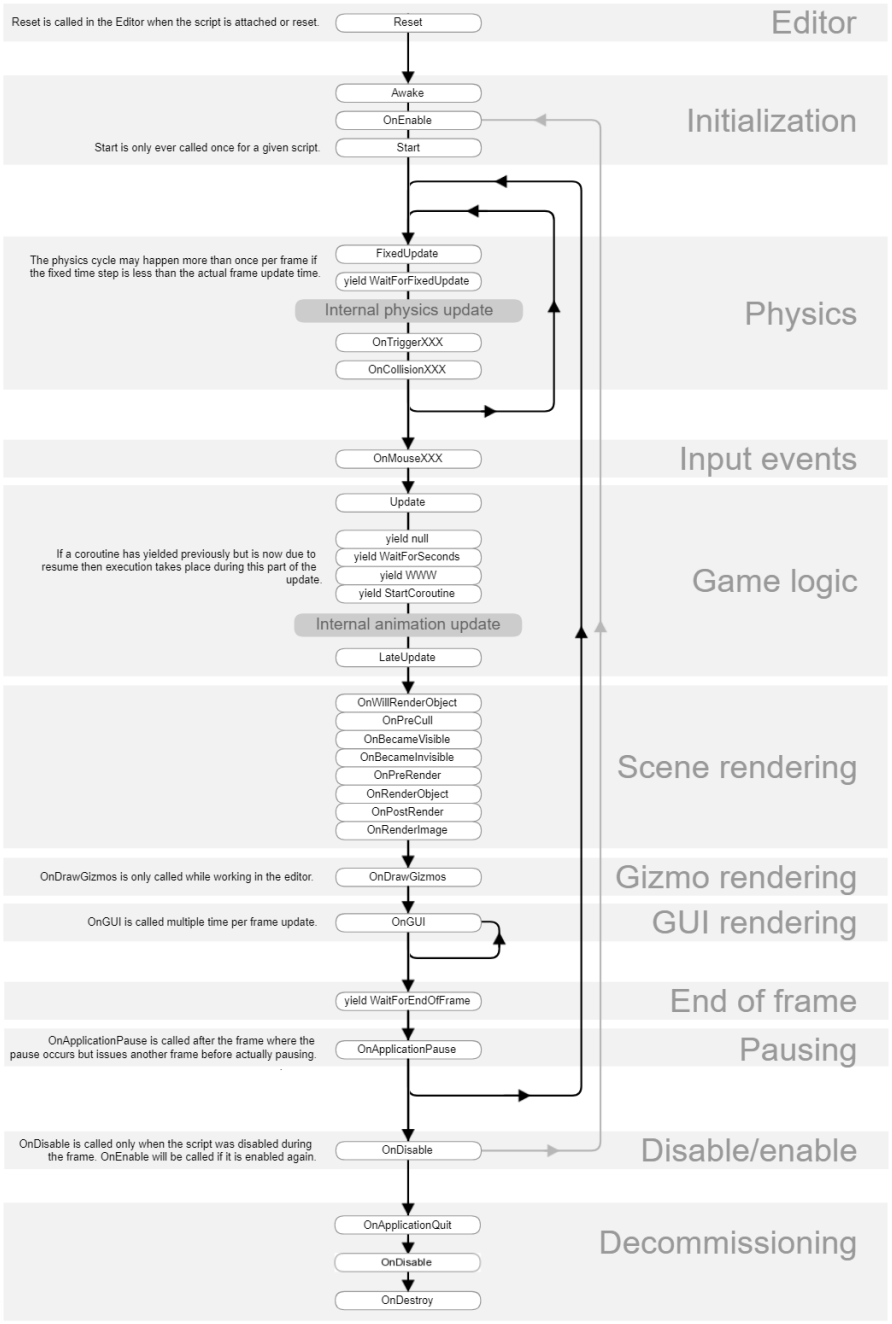

Рис 2.5. – цикл життя ігрового об’єкту в Unity[6]

2.4. Опис використаних технологій та мов програмування

У проекті використовувалися такі пакети як:

1. Unity 2D;
2. C#
3. JSON
4. 2D Animation;
5. 2D Common;
6. 2D Pixel Perfect;
7. 2D Sprite;
8. 2D Tilemap Editor;
9. Cinemachine;
10. Core RP Library;
11. JetBrains Rider Editor;
12. Mathematics;
13. Test Framework;
14. TextMeshPro;
15. Timeline;
16. Unity UI;
17. Universal RP.
18. Firebase
19. Firebase Realtime Database

Unity 2D – це двигун для створення 2д програм та ігор[1].

C# — це високорівнева об’єктно-орієнтована мова програмування, яка також створена як розширення C. Скрипти для Unity пишеться на двох мовах C# та JavaScript [2].

(JSON) JavaScript Object Notation

The JSON format is syntactically identical to the code for creating JavaScript objects. Because of this similarity, a JavaScript program can easily convert JSON data into native JavaScript objects [3].

2D Animation надає всі необхідні інструменти та компоненти часу виконання для скелетної анімації за допомогою спрайтів[4].

2D Common — це пакет, який містить спільні функції, які використовуються більшістю інших 2D-пакетів[5].

Пакет 2D Pixel Perfect містить компонент Pixel Perfect Camera, який гарантує, що ваше піксельне зображення залишається чітким і чітким при різних роздільних здатність і стабільним у русі.

2D Sprites – це об'єкти 2D-графіки. Якщо ви звикли працювати в 3D, спрайти - це, по суті, просто стандартні текстури, але є спеціальні прийоми комбінування текстур спрайтів та управління ними для підвищення ефективності та зручності під час розробки[7].

2D Tilemap Editor — це пакет, який містить функції редактора для редагування Tilemap[8].

Cinemachine - Розумні інструменти для камери для захоплених творців.

Core RP Library - SRP Core спрощує створення або налаштування сценарію Render Pipeline (SRP). SRP Core містить код для повторного використання, включаючи шаблонний код для роботи з графічними API для певної платформи, допоміжні функції для загальних операцій візуалізації та бібліотеки шейдерів. Код у ядрі SRP використовується конвеєром візуалізації високої чіткості (HDRP) та Universal Render Pipeline (URP). Якщо ви створюєте власний SRP з нуля або налаштовуєте попередньо створений SRP, використання SRP Core заощадить ваш час.

JetBrains Rider Editor - Пакет JetBrains Rider Editor забезпечує інтеграцію для використання JetBrains Rider IDE як редактора коду для Unity. Він додає підтримку для створення файлів .csproj для завершення коду та автоматичного виявлення інсталяцій[16].

Mathematics - математична бібліотека SIMD C# від Unity надає векторні типи та математичні функції із синтаксисом шейдера.

Shader Graph - Пакет Shader Graph додає інструмент візуального редагування шейдерів до Unity. Ви можете використовувати цей інструмент

для створення шейдерів у візуальний спосіб замість написання коду. Конкретні конвеєри візуалізації можуть реалізувати конкретні функції графіка. Наразі і конвеєр візуалізації високої чіткості, і універсальний конвеєр візуалізації підтримують Shader Graph.

Test Framework - Тестовий фреймворк для запуска тестів режиму редагування та режиму відтворення в Unity.

TextMeshPro - TextMeshPro — найкраще текстове рішення для Unity. Це ідеальна заміна тексту інтерфейсу користувача Unity і застарілої Text Mesh.

Потужний і простий у використанні TextMeshPro (також відомий як TMP) використовує розширені методи візуалізації тексту разом із набором користувацьких шейдерів; забезпечуючи суттєві покращення якості візуального ефекту, надаючи користувачам неймовірну гнучкість, коли справа доходить до стиля та текстурування тексту.

TextMeshPro забезпечує покращений контроль над форматуванням і макетом тексту з такими функціями, як інтервал між символами, словами, міжрядками та абзацами, кернінг, вирівняний текст, посилання, доступні понад 30 тегів розширеного тексту, підтримка кількох шрифтів і спрайтів, користувацькі стилі тощо.

Чудова продуктивність. Оскільки геометрія, створена TextMeshPro, використовує два трикутника на символ, як і текстові компоненти Unity, ця покращена візуальна якість і гнучкість не забезпечують додаткових витрат на продуктивність.

Timeline - Використовуйте хронологію Unity для створення кінематографічного вмісту, ігрових секцій, аудіосекцій та складних ефектів частинок.

Unity UI - Unity UI — це набір інструментів для розробки інтерфейсів користувача для ігор і програм. Це система інтерфейсу користувача на основі GameObject, яка використовує компоненти та Game View для організації, розташування та стилю інтерфейсу користувача. Ви не можете

використовувати Unity UI для створення або зміни інтерфейсів користувача в редакторі Unity.

Universal RP – Універсальний конвеєр візуалізації (URP) — це попередньо вбудований конвеєр рендеринга для сценаріїв, виготовлений Unity. URP забезпечує зручні для художників робочі процеси, які дозволяють швидко та легко створювати оптимізовану графіку на різних платформах, від мобільних до високоякісних консолей і ПК [15].

Firebase - Firebase - це повний набір продуктів, який пропонує багато функцій, таких як спрощена автентифікація, хмарний хостинг, машинне навчання, аналіз та моніторинг збоїв. Однак сервіс, з якого все це почалося, - Firebase Realtime Database, база даних NoSQL, на яку клієнти можуть підписатися через WebSockets.

База даних Firebase Realtime – це база даних, розміщена у хмарі. Дані зберігаються у форматі JSON та синхронізуються для кожного підключеного клієнта. Коли ви створюєте кросплатформні програми за допомогою наших платформ Apple, Android та JavaScript SDK, всі ваші клієнти спільно використовують один екземпляр бази даних і автоматично отримують оновлення з новітніми даними.

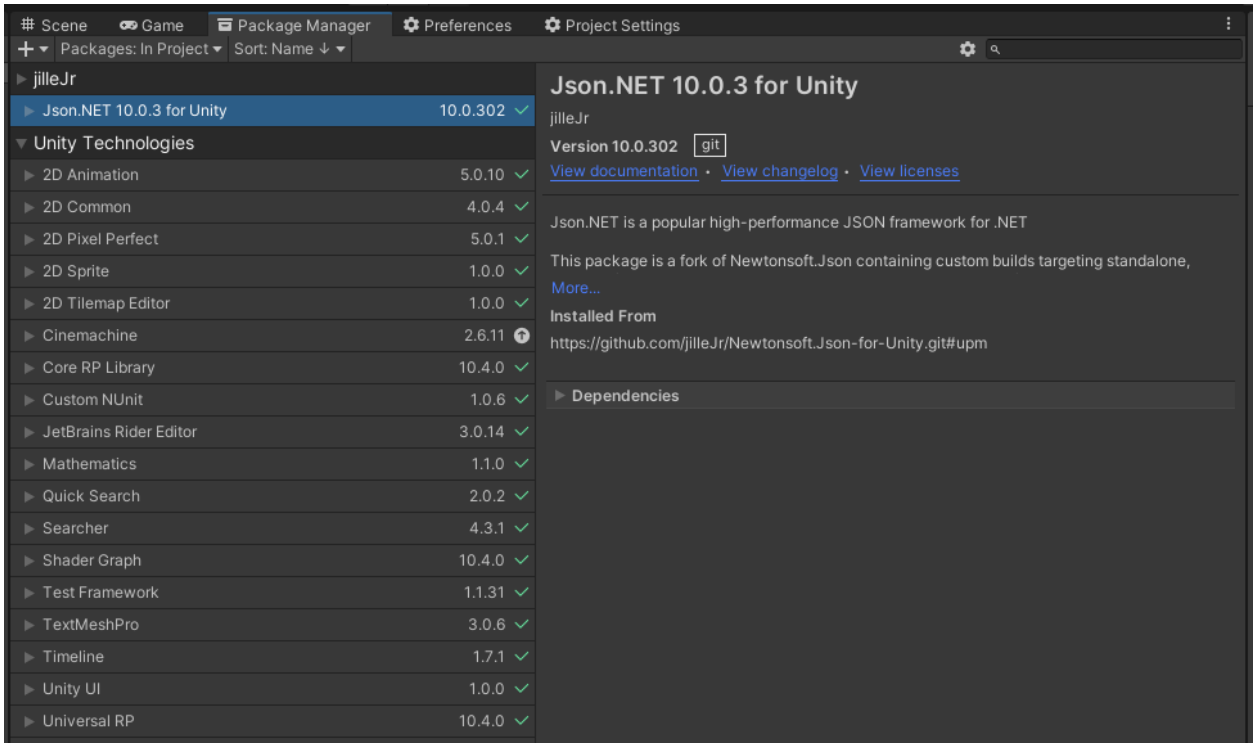

Рис 2.6. – пакети що використовувалися

2.5. Опис структури програми та алгоритми її функціонування

У даному проекті ми маємо 5 сцен:

- Preloader
- NoInternetConnection
- Authorization
- Menu
- Game

Першою у нас завантажується сцена “Preloader”. Якщо у користувача немає інтернету то переходить на на сцену “ NoInternetConnection”, інакше буде перевірка на те чи авторизований користувач чи ні малюнок 2.7. Якщо ж у користувача є інтернет, то програма перевіряє чи авторизований користувач. Якщо так, то переходить одразу до сцени “Menu”. Якщо користувача не авторизований – переходить на сцену “Authorization”, дивить малюнок 2.7

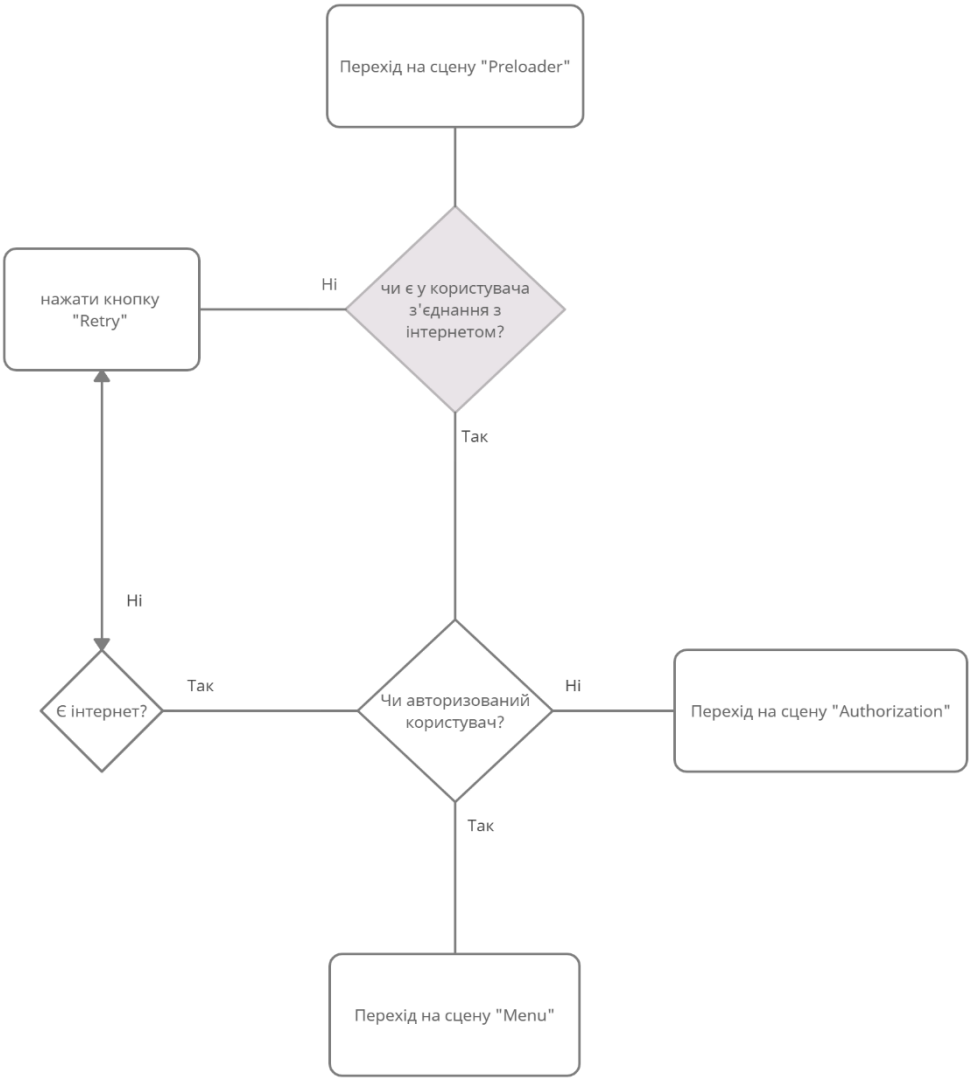

Рис 2.7. – діаграма сцени “Preloader”

На сцені “NoInternetConnection” знаходиться кнопка, для того щоб перевірити наявність інтернету. Якщо інтернет з’явився, то користувач переходить до наступної сцени. Дивись малюнок 2.8.

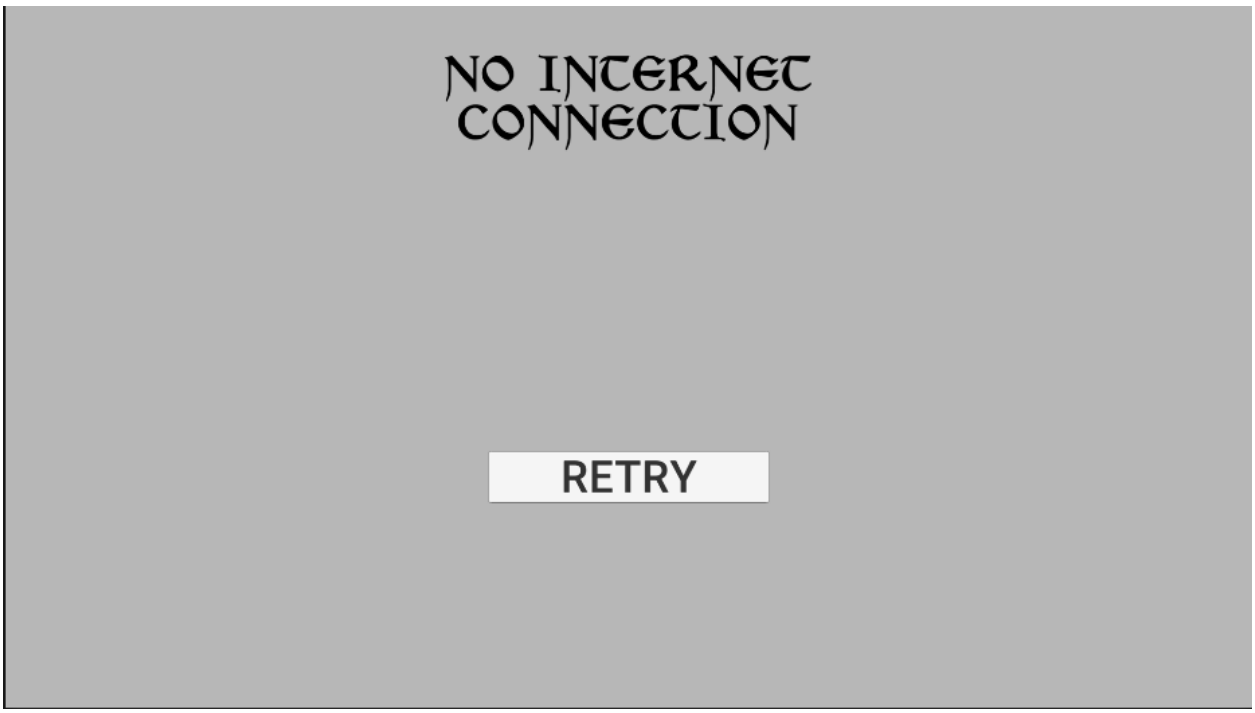

Рис 2.8. – сцена “NoInternetConnection”

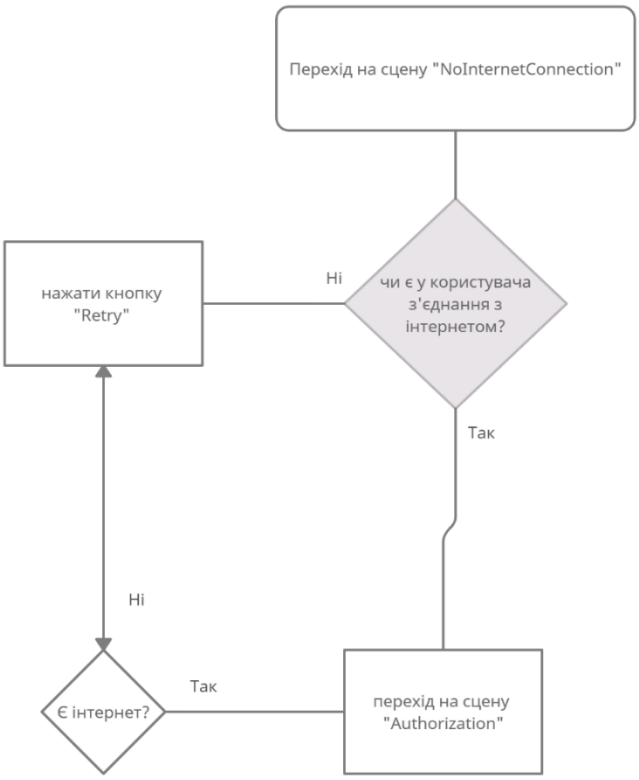

Рис 2.9. – діаграма сцени “NoInternetConnection”

На сцені “Authorization” користувач має зареєструватися, або, якщо у нього вже є обліковий запис зайти до наступної сцени. В іншому випадку користувач має створити свій обліковий запис, а вже після цього переходити на сцену, дивись малюнок 2.10 та малюнок 2.11.

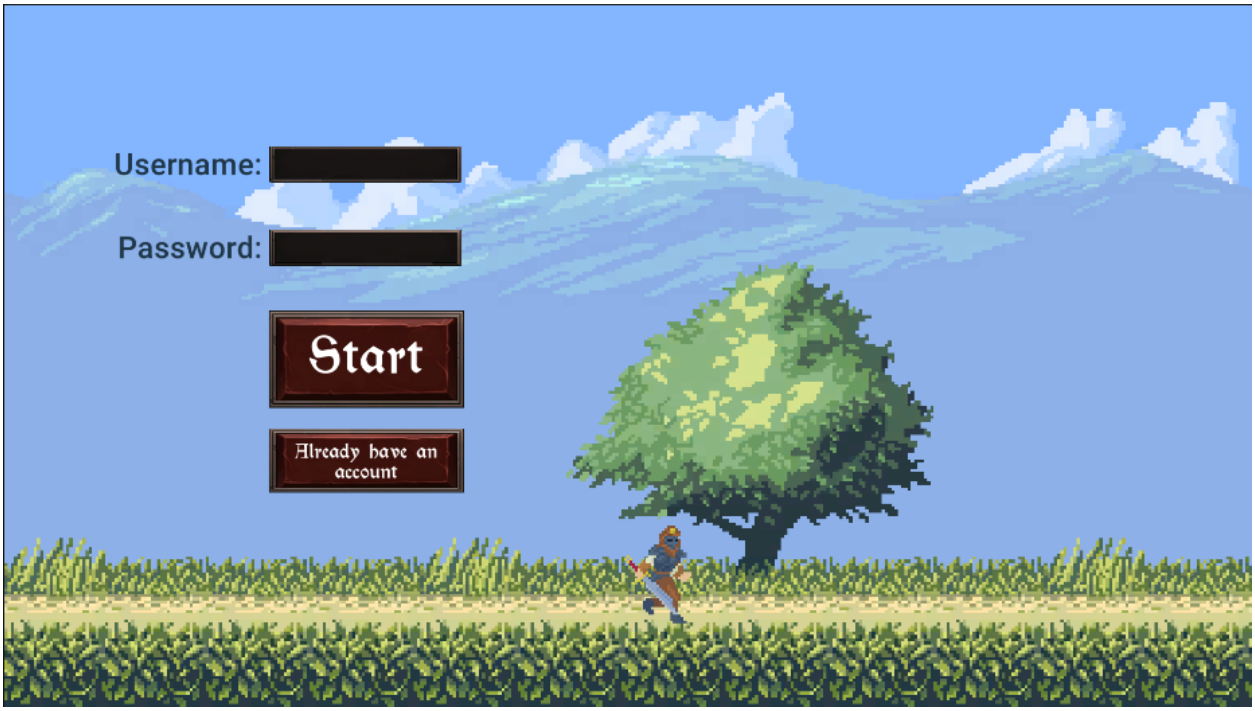

Рис. 2.10. – сцена “Authorization”

Рис 2.11. – діаграма сцени “Authorization”

На наступній сцені, сцені “Menu”, користувач може зробити Logout (вийти з акаунту), для виходу з акаунту, вимкнути звуки, вийти із додатку або ж натиснути кнопку для початку гри, а саме – перейти на останню сцену – “Game”. Тут вже буде відбуватися основні події.

Рис. 2.12. – сцена “Menu”

У сцені “Game” користувач має проходити рівні, долаючи ворогів. Якщо персонаж даної гри помирає, то у нього запитуюсь чи хоче він пройти рівень знову чи перейти на сцену “Menu”. У кейсі якщо користувач пройшов рівень та його головний персонаж не помер, то йому на його екрані з'являється “result screen” та на ньому пропонується перегратри даний рівень або перейти на сцену “Menu”, малюнок 2.13. Якщо ж користувач виграє, то дані відсилаються на ссервер та зберігаться. Також на екрані йому пропонується перейти до меню, або ж до наступного рівня, малюнок 2.14

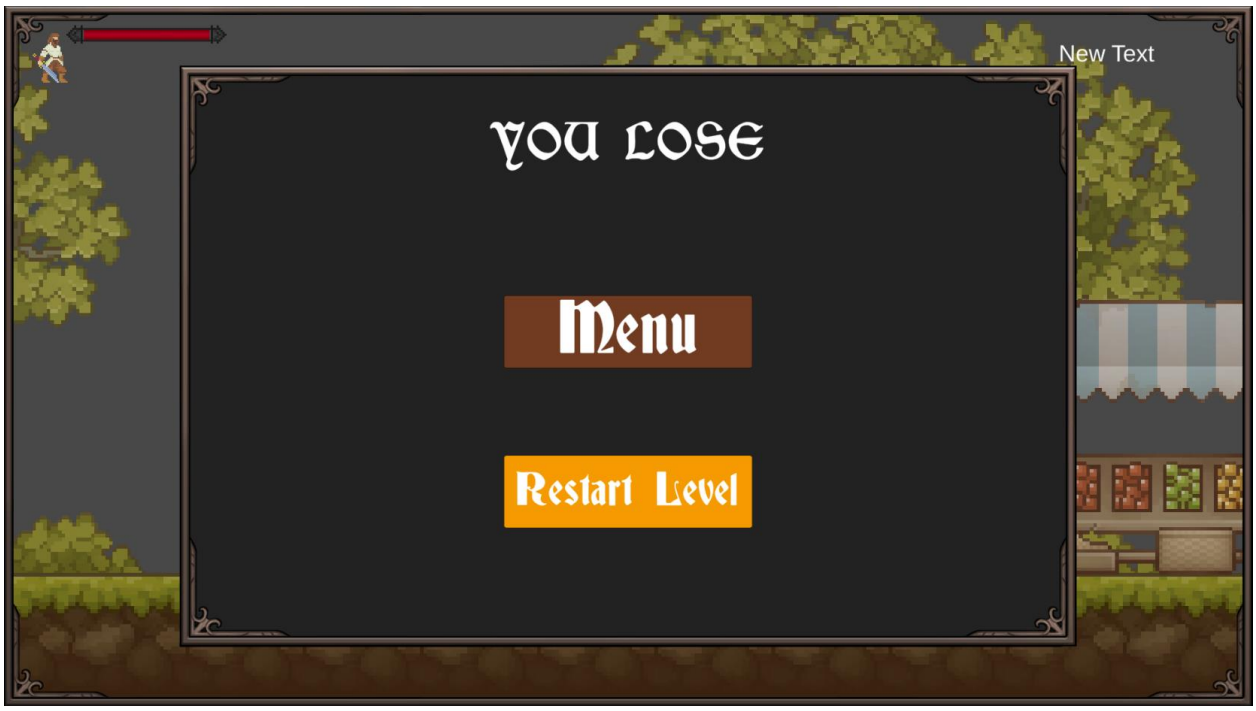

Рис 2.13. резалт скрін поразки.

Рис 2.14 – резалт скрін победи.

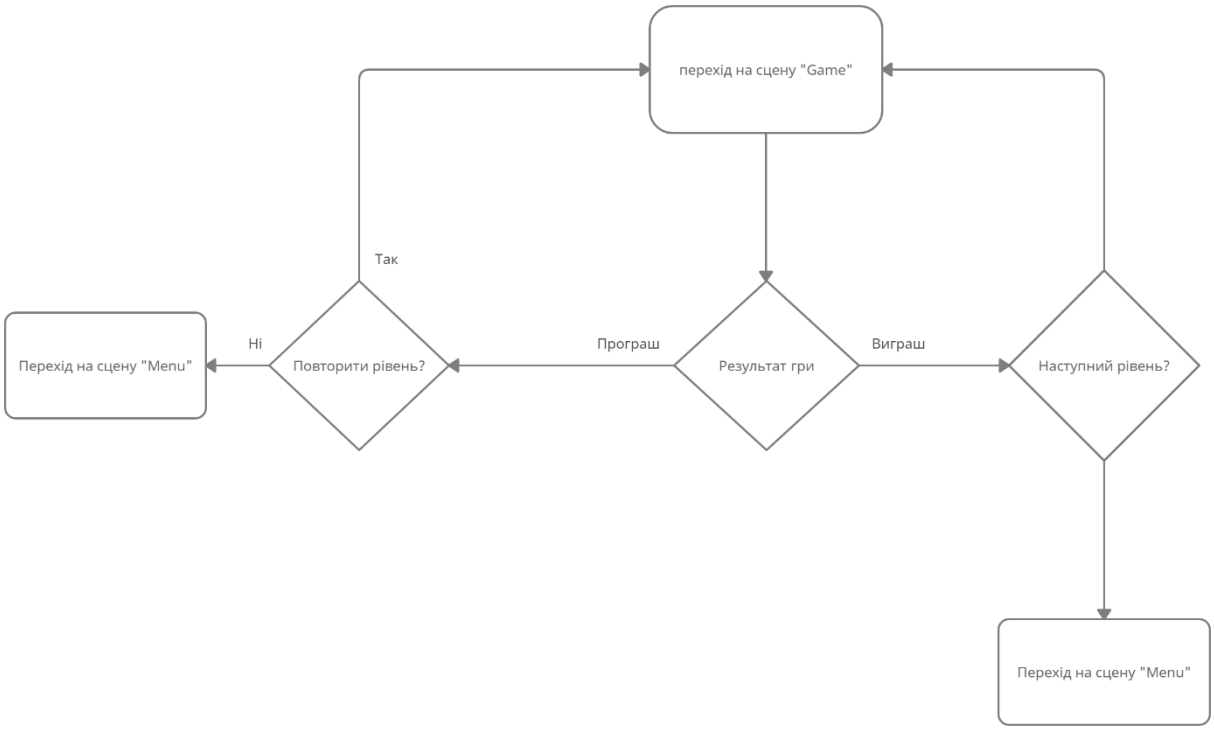

Рис 2.15. діаграма сцени “Game”

Серверна частина реалізована за допомогою Firebase[10] та його компонента Firebase Realtime Database [11]. Через цю API ми зможемо додавати та змінювати дані користувачів. Дані можно подивитися на сайті додатку у Firebase Console[12], малюнок 2.16 та 2.17.

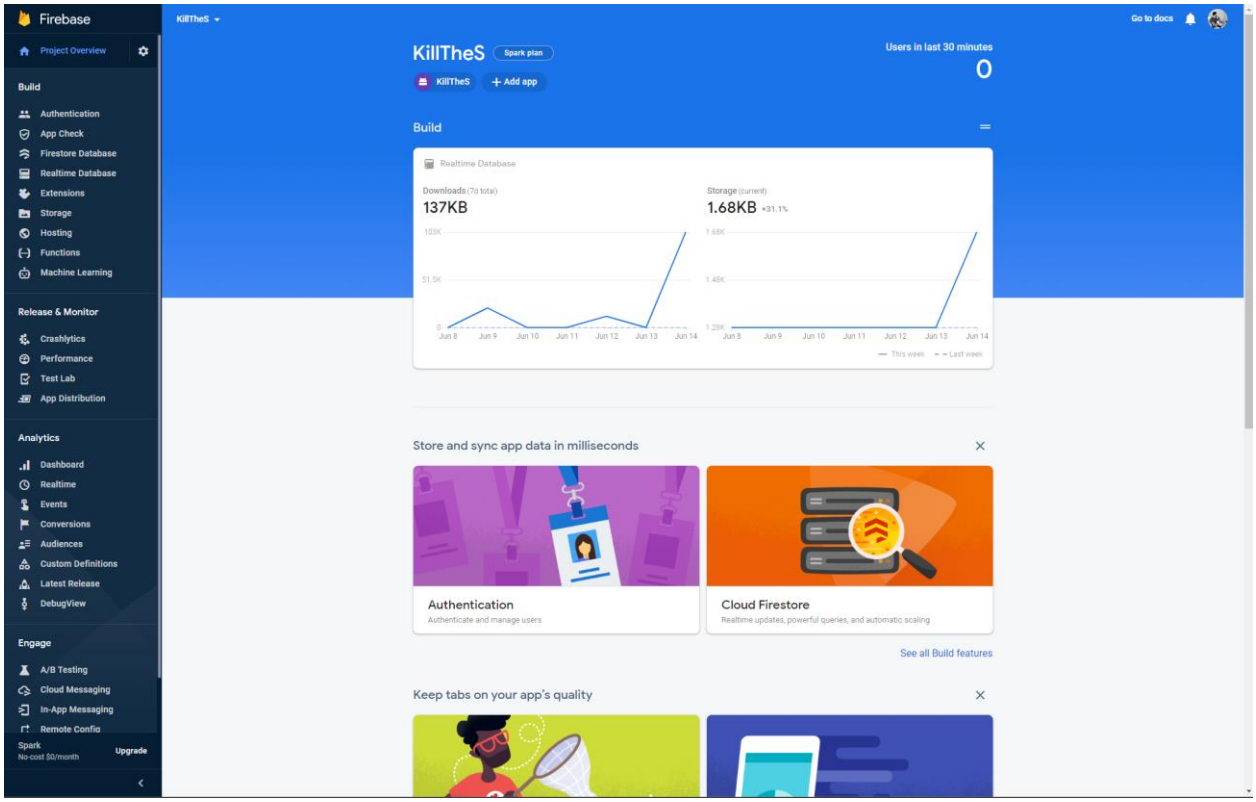

Рис 2.16. – FirebaseConsole

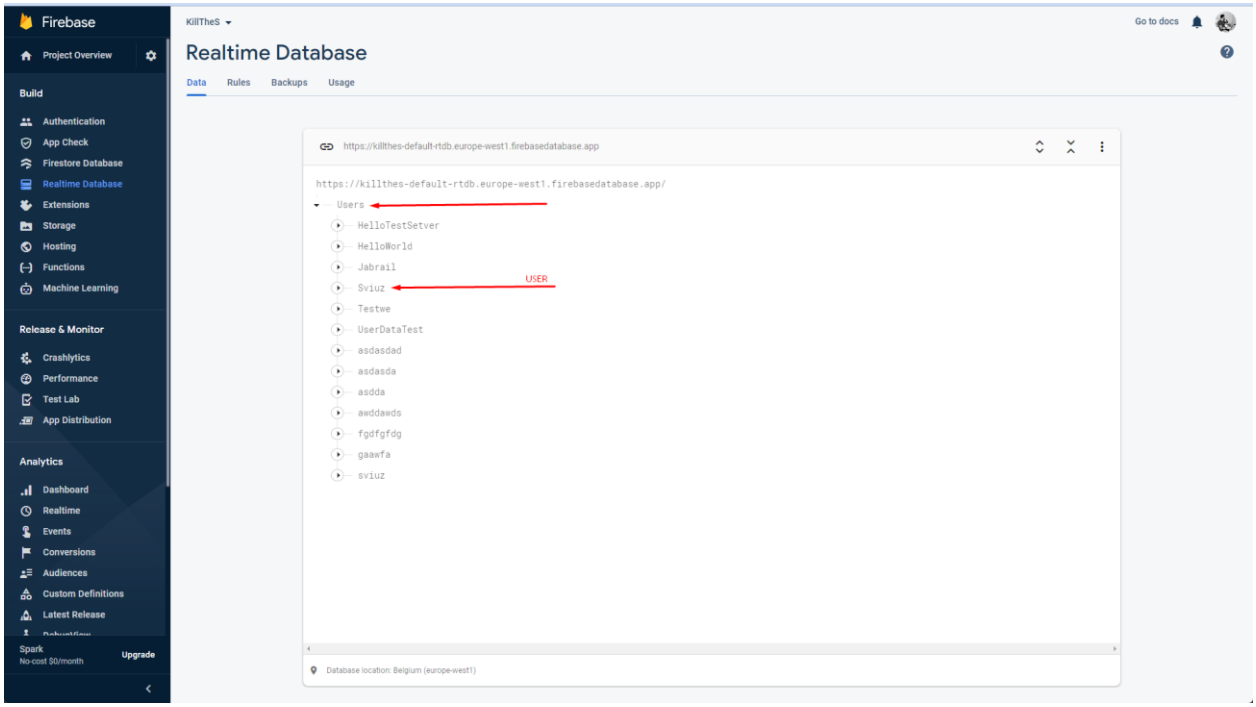

Рис 2.17. – Firebase Realtime Database

2.6. Обґрунтування та організація вхідних та вихідних даних програми

Аналізуючи те що гра буде для персональний комп’ютерів, то я мав підібрати зрозумілий інтерфейс, який повністю забезпечує зручне користування гри у користувача. Тому для керування гри на комп’ютері доцільно буде використовувати мишку та клавіатуру.

Вхідні дані: управління у додатку виконується за допомогою клавіатури та комп’ютерної миші.

Вихідні дані які генеруються при роботі додатку при обробці даних які отримує користувач при .

Вихідні дані в застосунку це реакція інтерфейсу на натискання кнопок та клавіш та клавіш миші.

В ігровій сцені це переміщення позиції гравця або створення бомб

2.7. Опис розробленого програмного продукту

Файлі, котрі ми получаємо після збірки програмного продукту, містять в собі виконавчий файл з розширенням .exe та багато інший файлів та папок. Вибраний формат файлу може запускатися лише на системах Windows. Видалення файлів з папки з проектом може буди причиною нестабільної роботи програми.

2.7.1. Використані технічні засоби

При розробці програми була використана персональна ЕОМ з наступними характеристиками:

- Процесор i7 10700H;
- Відеопроцесор Nvidia RTX 3060 6 ГБ GDDR6;
- Оперативна пам'ять 32 ГБ DDR5-6000.

Тестування проводилося на таких пристроях:

- Asus Tuf Dash F15.

2.7.2. Використані програмні засоби

Гру було написано в Unity 2020.3.5f1. Програмний код був написаний в середовищі написанні коду Rider Jetbrains. Для застосування контроля версій було використано таку програму як Git Bash з візуальною оболонкою у виді GitHub Desktop. Для коригування картинок була використана така програма як Adobe Photoshop. Також для внутрішнього тестування використовувався внутрішній емулятор з Unity.

2.7.3. Виклик та завантаження програми

Для запуску програмної гри встановлення на персональний комп'ютер не потребується. Від користувача потребується лише запустити файл з розширенням .exe.

2.7.4. Опис інтерфейсу користувача

Інтерфейс додатку є стандартним та зручним для розуміння користувачу будь-якого рівня. Після запуску програми, користувач має змогу бачити меню аутентифікації(малюнок 2.18.), або, якщо він авторизований, то головне меню(малюнок 2.20.). Увесь функціонал додатку є інтуїтивно зрозумілим і не передбачає спеціалізованих знань у даній сфері. Окрім того, для кращого розуміння роботи гри, користувач може прочитати коротку інструкцію з користування.

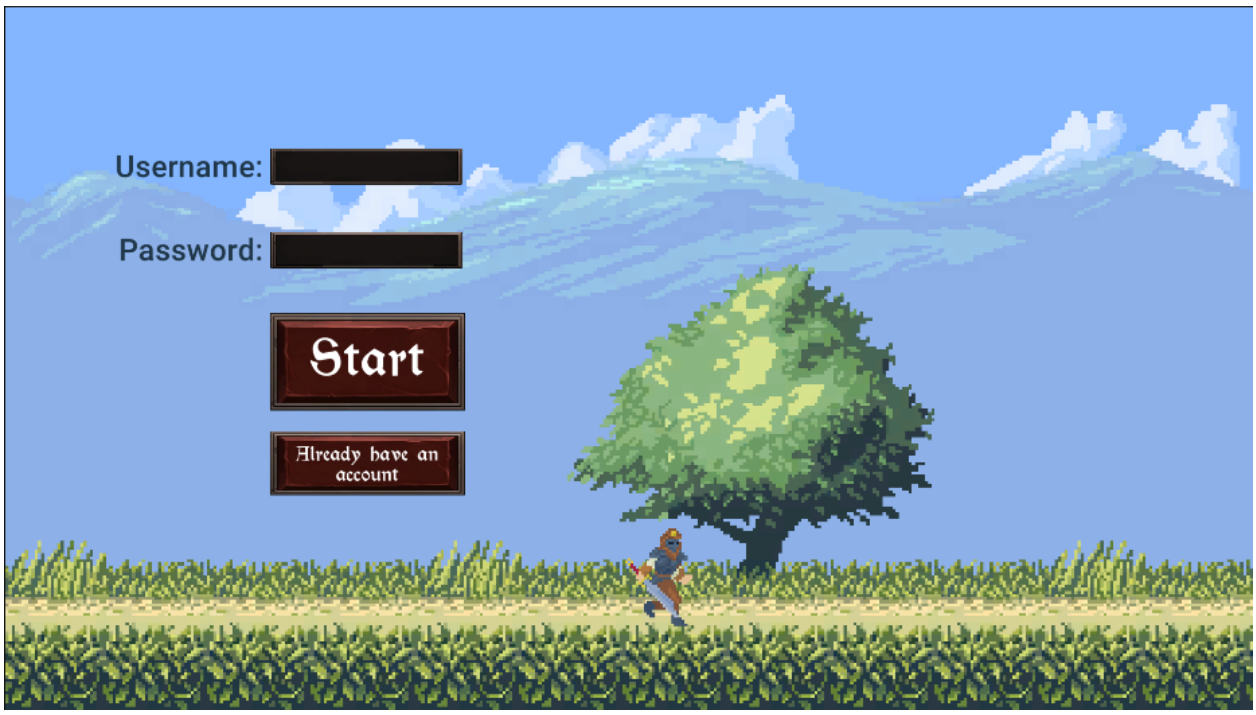

Рис. 2.18. – Меню авторизації

При неправильному вводі даних, виникає помилка, та програма пише що сталася помилка(малюнок 2.19.).

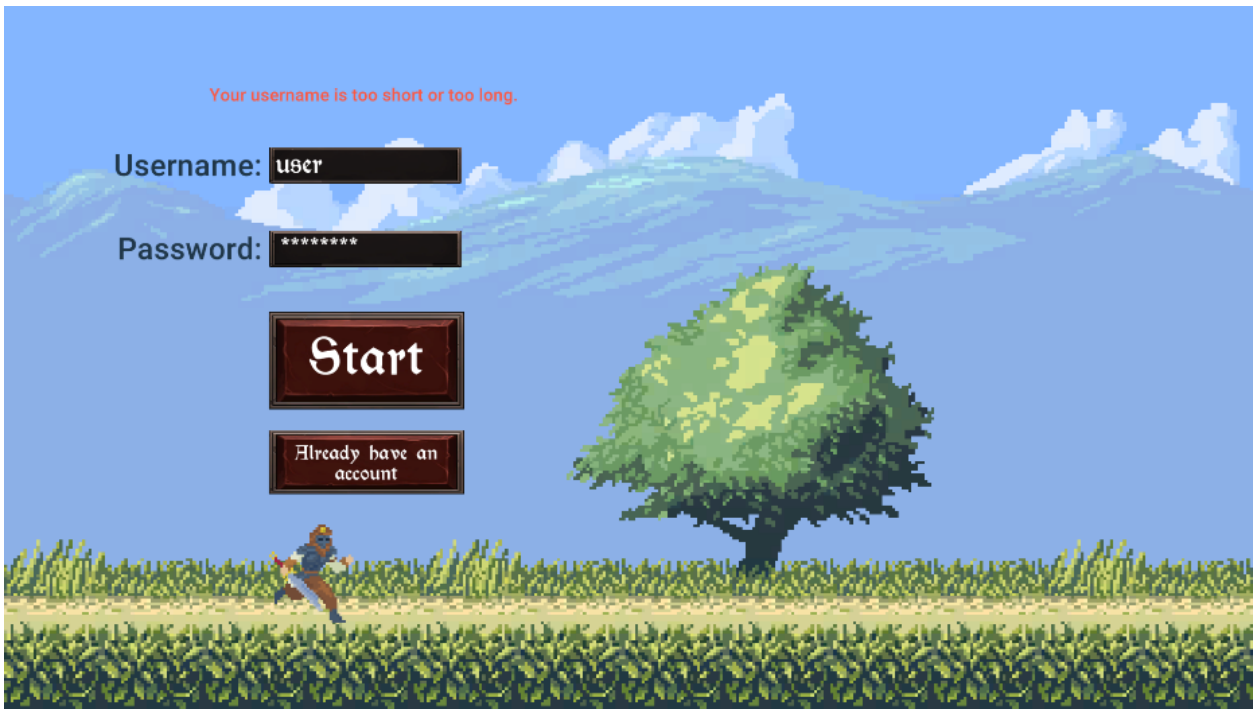

Рис. 2.19. – помилка при введені некоректних даних

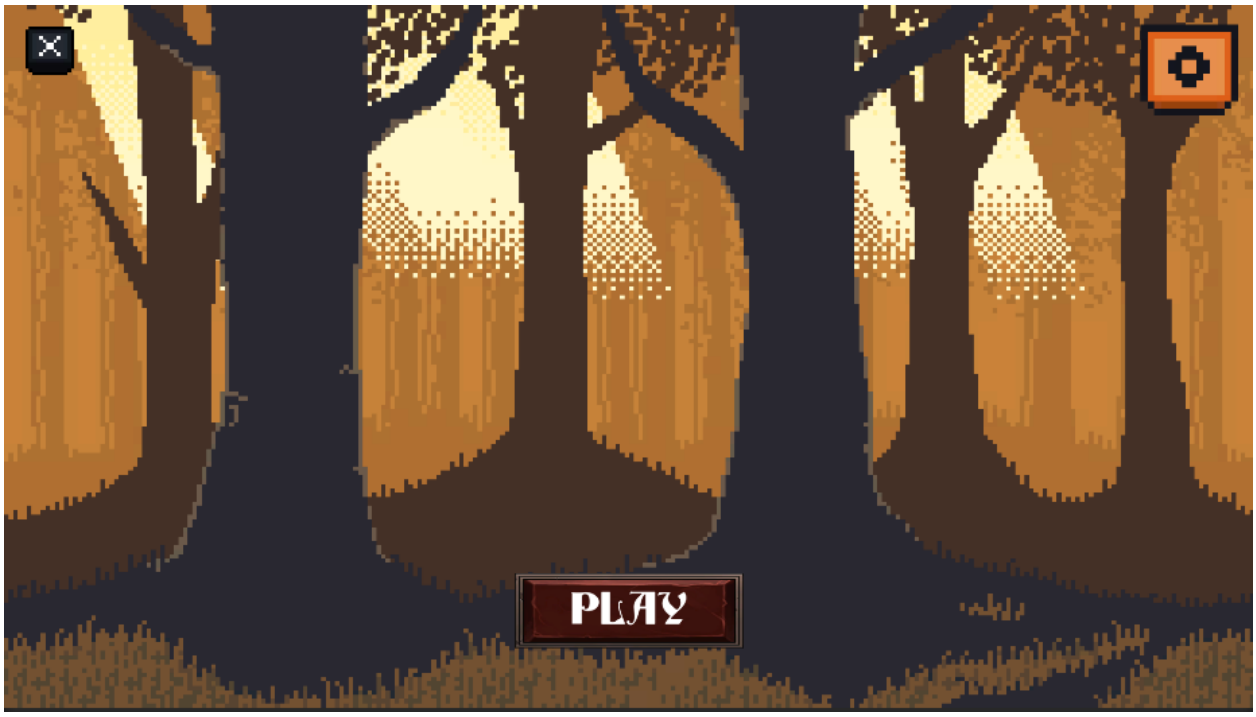

Рис. 2.20. Головне меню

При натисканні кнопки Налаштувань, відкриваються 2 нові кнопки: вимкнути звук та вийти з акаунту(малюнок 2.21).

Рис. 2.21. – налаштування та дві кнопки

При натасканні кнопки Play починається основна сцена Game(малюнок 2.22).

Рис. 2.22. – Основна сцена Game

РОЗДІЛ 3
ЕКОНОМІЧНІЙ РОЗДІЛ

3.1. Розрахунок трудомісткості та вартості розробки програмного продукту

Початкові дані:

1. передбачуване число операторів програми – 1311;
2. коефіцієнт складності програми – 1,5;
3. коефіцієнт корекції програми в ході її розробки – 0,07;
4. годинна заробітна плата програміста – 120 грн/год;
5. коефіцієнт збільшення витрат праці внаслідок недостатнього опису задачі – 1,2;
6. коефіцієнт кваліфікації програміста, обумовлений від стажу роботи з даної спеціальності – 0,9;
7. вартість машино-години ЕОМ – 15 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 (1500);

C – коефіцієнт складності програми (1,5);

p – коефіцієнт кореляції програми в ході її розробки (0,07).

$$Q = 1500 \cdot 1{,}5 \cdot (1 + 0{,}07) = 2265{,}75;$$

Витрати праці на вивчення опису задачі 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 (0.9);

$$t_u = \frac{2265{,}75 \cdot 1{,}2}{85 \cdot 0.9} = 35{,}53, \text{ людино-годин.}$$

Витрати праці на розробку алгоритму рішення задачі:

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

$$t_a = \frac{2265,75}{20 \cdot 1,1} = 125,87;$$ людино-годин.

Витрати на складання програми по готовій блок-схемі:

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

$$t_n = \frac{2265,75}{25 \cdot 0,9} = 100,7;$$ людино-годин.

Витрати праці на налагодження програми на ЕОМ:

– за умови автономного налагодження одного завдання:

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

$$t_{\text{отл}} = \frac{2265,75}{5 \cdot 0,9} = 503,5,$$ людино-годин,

– за умови комплексного налагодження завдання:

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

$$t_{\text{отл}}^{\text{к}} = 1,2 \cdot 503,5 = 604,2,$$ людино-годин,

Витрати праці на підготовку документації:

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

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

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

$$t_{\partial} = \frac{2265{,}75}{20 \cdot 0.9} = 125{,}87, \text{ людино-годин,}$$

де $t_{до}$ – трудомісткість редагування, печатки й оформлення документації;

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

$$t_{до} = 0{,}75 \cdot 125{,}87 = 94{,}4, \text{ людино-годин.}$$

$$t_{\partial} = 125{,}87 + 94{,}4 = 220{,}27, \text{ людино-годин.}$$

Отримаємо трудомісткість розробки програмного забезпечення:

t = 50 + 35,53+ 125,87 + 100,7 + 503,5 + 220,27 = 1035,6, людино-годин.

У результаті ми розрахували, що в загальній складності необхідно 1035,6 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 $З_{ЗП}$ 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 120 грн/год, то отримаємо:

$$З_{ЗП} = 1035{,}6 \cdot 120 = 124272, \text{ грн.}$$

Вартість машинного часу $З_{МВ,}$ 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.

$$З_{МВ} = 503{,}5 \cdot 15 = 7552{,}5 \text{ грн.}$$

Звідси витрати на створення програмного продукту:

$$К_{ПО} = 124272 + 7552{,}5 = 131824{,}5 \text{ грн.}$$

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:

$$T = \frac{1035{,}6}{1 \cdot 176} = 5{,}9 \text{ міс.}$$

Висновки. Мобільна гра має вартість 131824,50 грн. Ймовірний очікуваний час розробки – 5,9 місяці при стандартному 40-годинному робочому тижні і 176-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 1035,6 людино-годин.

ВИСНОВКИ

Метою кваліфікаційної роботи бакалавра є розробкою 2д платформеру з використанням зберігання даних на сервері за допомогою Unity 2D. В основному даний додаток був створений не для комерційного застосування, а для користувачів для персонально особистих цілей. Користувач може використовувати додаток тільки якщо він авторизований та має стабільне підключення до мережі інтернет.

Під час виконання кваліфікаційної роботи був розроблений практичний додаток, використовуючи нові технології та підходи до створення програмного забезпечення, а саме – комп'ютерної гри.

Додаток був створений для платформи Windows, яка э майже у кожного користувача на його персональному комп'ютері. Гра була реалізована на основі Unity 2D. Також використовувалися компоненти цієї системи для прокращення візуальних ефектів, візуального інтерфейсу та фізики даної гри. Кодування було реалізовано на високорівневій мові програмування C#.

Сервером у даному додатку виступає Firebase Realtime Database.

В «Економічному розділі» було визначено трудомісткість розробки програмного забезпечення (1035,6 люд-год) на одного молодшого розробника ігор. Також були підраховані витрати на створення програмного забезпечення (131824,50 грн.). Підрахований період розробки додатку становить (5,9 міс.).

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Unity Documentation скриптування https://docs.unity3d.com/Manual/ScriptingSection.html
2. C# Microsoft Documentation https://docs.microsoft.com/en-us/dotnet/csharp/
3. JSON https://docs.microsoft.com/en-us/dotnet/standard/serialization/system-text-json-how-to?pivots=dotnet-6-0
4. Unity 2D Animation Documentation https://docs.unity3d.com/Packages/com.unity.2d.animation@8.0/manual/index.html
5. Unity 2D Common https://docs.unity3d.com/2019.3/Documentation/Manual/com.unity.2d.common.html
6. Unity та C#. Геймдев від ідеї до реализації. 2-е изд., Бонд Д.
7. Unity 2D Sprites https://docs.unity3d.com/Manual/Sprites.html
8. Unity 2d Tilemap Editor https://docs.unity3d.com/Manual/com.unity.2d.tilemap.html
9. Unity Cinemachine https://unity.com/ru/unity/features/editor/art-and-design/cinemachine
10. Unity Firebase SDK https://gametorrahod.com/firebase-behaviour-in-unity/
11. Firebase Realtime Database https://firebase.google.com/products/realtime-database#:~:text=The%20Firebase%20Realtime%20Database%20is,app%20data%20at%20global%20scale.
12. Firebase Console https://console.firebase.google.com/u/0/project/_/database/data
13. Платформер Википедия https://ru.wikipedia.org/wiki/%D0%9F%D0%BB%D0%B0%D1%82%D1%84%D0%BE%D1%80%D0%BC%D0%B5%D1%80

14. Unity URP https://docs.unity3d.com/Packages/com.unity.render-pipelines.universal@11.0/manual/
15. Unity Draw call Bathcing https://docs.unity3d.com/2019.4/Documentation/Manual/DrawCallBatching.html
16. JetBrains Rider IDE https://www.jetbrains.com/ru-ru/rider/

ДОДАТОК А

КОД ПРОГРАМИ

Лістинг PlayerDataChecker.cs

```
using System.Collections.Generic;
using System.Linq;
using Firebase;
using static Other.PlayerConstants;

namespace Core.Player {
  public static class PlayerDataChecker {

    public static AuthenticationErrorType CheckUserProvidedData(string username, string password) {
      List<string> listOfUsernames = FirebaseDatabaseManager.Instance.GetListOfUsernames();
      List<string> listOfPasswords = FirebaseDatabaseManager.Instance.GetListOfPasswords();

      if (listOfUsernames.Select((t, index) => (t, listOfPasswords[index])).Any(obj => username == obj.Item1 && password == obj.Item2)) {
        return AuthenticationErrorType.AllGood;
      }

      if (listOfUsernames.Contains(username)) {
        return AuthenticationErrorType.UsernameAlreadyExist;
      } if (username.Length<6 || username.Length>16 || string.IsNullOrEmpty(username)) {
        return AuthenticationErrorType.NotValidUsername;
      } if (password.Length<6 || string.IsNullOrEmpty(password)) {
        return AuthenticationErrorType.NotValidPassword;
      }

      return AuthenticationErrorType.AllGood;
    }
  }
}
```

Лістинг програми PlayerMovement.cs

```
using System;
using Behaviour.Based;
using Behaviour.HealthItem;
using Behaviour.Objects;
using JetBrains.Annotations;
using Level;
using Ui.Level;
using UnityEngine;
using static Other.Constants.Constants;

namespace Behaviour {
  public class PlayerMovement : SubjectParameters {
    public static Action<bool> OnSetMovement;
    [SerializeField]
    private float _jumpForce = 7.5f;
    [SerializeField]
    private PlayerState _groundSensor;

    private Animator _animator;
    private Rigidbody2D _body2d;
    private BoxCollider2D _boxCollider;
    private bool _grounded;
    private bool _combatIdle;
    private bool _isDead;
    private bool _move = true;
    private bool _attack;

    private Vector2 _defaultBoxColliderX = new Vector2(0.8f, 1.5f);
    private Vector2 _defendedBoxColliderX = new Vector2(2f, 1.5f);

    private void Awake() {
      _animator = GetComponent<Animator>();
      _body2d = GetComponent<Rigidbody2D>();
      _boxCollider = GetComponent<BoxCollider2D>();

      OnSetMovement += SetMovement;
    }

    private void OnDestroy() {
      OnSetMovement -= SetMovement;
    }

    private void Update() {
      if (!_move) return;
      Grounded();
      float inputX = Input.GetAxis("Horizontal");
      Move(inputX);
      MainBehaviour(inputX);
    }

    private void MainBehaviour(float value) {
```

```
  if (Input.GetMouseButtonDown(0)) {
    Attack();
  } else if (Input.GetKeyDown(KeyCode.Mouse1)) {
    ChangeCombatPose();
  } else if (Input.GetKeyDown("space") && _grounded) {
    Jump();
  } else if (Mathf.Abs(value) > Mathf.Epsilon) {
    Move();
  } else if (_combatIdle) {
    CombatIdle();
  } else {
    Idle();
  }
}

private void SetMovement(bool status) {
  _move = status;
}

private void Move(float inputX) {
  if (inputX > 0) {
    transform.localScale = new Vector3(-1.0f, 1.0f, 1.0f);
  } else if (inputX < 0) {
    transform.localScale = Vector3.one;
  }

  _body2d.velocity = new Vector2(inputX * Speed, _body2d.velocity.y);

  _animator.SetFloat(nameof(MoveTriggers.AirSpeed), _body2d.velocity.y);
}

private void Move() {
  _animator.SetInteger(nameof(MoveTriggers.AnimState), 2);
}

private void Attack() {
  if (_attack) return;

  _attack = true;
  _animator.SetTrigger(nameof(MoveTriggers.Attack));
}

[UsedImplicitly]//used in animations
public void ReturnState() {
  _attack = false;
}

[UsedImplicitly]
private void Hit() {
  Collider2D[] cols = Physics2D.OverlapCircleAll(AttackPosition.position, AttackRange);
  if (cols.Length == 0) return;
```

```
  for (int i = 0; i < cols.Length; i++) {
    if (cols[i].TryGetComponent(out EnemyHealth _health)) {
      _health.TakeDamage(10);
    }
  }

  for (int i = 0; i < cols.Length; i++) {
    if (cols[i].TryGetComponent(out Chest _chest)) {
      _chest.BrokeChest();
    }
  }
}

private void Idle() {
  _animator.SetInteger(nameof(MoveTriggers.AnimState), 0);
}

public void Hurt() {
  _animator.SetTrigger(nameof(MoveTriggers.Hurt));
}

public void Death() {
  _animator.SetTrigger(!_isDead
    ? nameof(MoveTriggers.Death)
    : nameof(MoveTriggers.Recover));
  _isDead = !_isDead;
  LevelUI.Instance.Lose();
}

private void Jump() {
  _animator.SetTrigger(nameof(MoveTriggers.Jump));
  _grounded = false;
  _animator.SetBool(nameof(MoveTriggers.Grounded), _grounded);
  _body2d.velocity = new Vector2(_body2d.velocity.x, _jumpForce);
  _groundSensor.Disable(0.2f);
}

private void CombatIdle() {
  _animator.SetInteger(nameof(MoveTriggers.AnimState), 1);
}

private void ChangeCombatPose() {
  _combatIdle = !_combatIdle;
  _boxCollider.size = _combatIdle ? _defendedBoxColliderX : _defaultBoxColliderX;
}

private void Grounded() {
  if (!_grounded && _groundSensor.State()) {
    _grounded = true;
    _animator.SetBool(nameof(MoveTriggers.Grounded), _grounded);
  }
```

```
      if (!_grounded || _groundSensor.State()) return;

      _grounded = false;
      _animator.SetBool(nameof(MoveTriggers.Grounded), _grounded);
    }

    private void OnTriggerEnter2D(Collider2D col) {
      if (col.TryGetComponent(out Objects.Items.HealthItem item)) {
        Debug.Log(item.name);
        item.Pick(transform.position);
      }

      if (col.TryGetComponent(out WinObject winObject)) {
        winObject.Win();
      }
    }
  }
}
```

Лістинг AuthenticationMenu.cs

```
using System;
using Core.Player;
using Firebase;
using Helpers;
using Other;
using UnityEngine;
using UnityEngine.SceneManagement;

namespace Ui.Auth {
  public class AuthenticationMenu : MonoBehaviour {
    [SerializeField] private AuthenticationMenuContainer _container;

    private bool login = true;

    private void Awake() {
      if (_container == null) {
        _container = GetComponent<AuthenticationMenuContainer>();
      }
    }

    private void Start() {
      _container.SaveButton.onClick.AddListener(SaveButtonClick);
    }

    private void SaveButtonClick() {
      PlayerConstants.AuthenticationErrorType taskResult =
        PlayerDataChecker.CheckUserProvidedData(
        _container.UsernameInputField,
        _container.PasswordInputField);

      switch (taskResult) {
```

```
        case PlayerConstants.AuthenticationErrorType.UsernameAlreadyExist:
          _container.ErrorTMPText.text = "Username is already exists.";
          break;
        case PlayerConstants.AuthenticationErrorType.NotValidUsername:
          _container.ErrorTMPText.text = "Your username is too short or too long.";
          break;
        case PlayerConstants.AuthenticationErrorType.NotValidPassword:
          _container.ErrorTMPText.text = "Your password is weak";
          break;
        case PlayerConstants.AuthenticationErrorType.AllGood:
          SuccessfulDataInput();
          break;
        default:
          _container.ErrorTMPText.text = "Internal Error";
          break;
      }
    }

    private void SuccessfulDataInput() {
      string id = DataGenerator.GenerateId(10);
      FirebaseDatabaseManager.Instance.SaveData(
        id,
        _container.UsernameInputField,
        _container.PasswordInputField
      );
      PlayerDataSaver.LogIn();
      PlayerDataSaver.SaveData(
        id,
        _container.UsernameInputField,
        _container.PasswordInputField);
      SceneManager.LoadScene("Menu");
    }
  }
}
```

Enemy.cs

```
using Other.Constants;
using UnityEngine;

namespace Behaviour.Enemy {
  public class EnemyMovement : MonoBehaviour {
    [SerializeField] private Transform _leftEdge;
    [SerializeField] private Transform _rightEdge;
    [SerializeField] private Transform _enemy;
    [SerializeField] private float _speed;
    [SerializeField] private float _idleDuration;
    [SerializeField] private Animator _anim;
    [SerializeField] private BoxCollider2D _boxCollider;
    [SerializeField] private Rigidbody2D _rigidbody;
    private int _facingDirection = -1;
```

```
private float _idleTimer;
private Vector3 _initScale;
private bool _moveLeft = true;

private void Awake() {
  _initScale = _enemy.localScale;
}

private void Update() {
  if (_moveLeft) {
    if (_enemy.position.x >= _leftEdge.position.x) {
      MoveInDirection(-1);
    } else {
      DirectionChange();
    }
  } else {
    if (_enemy.position.x <= _rightEdge.position.x) {
      MoveInDirection(1);
    } else {
      DirectionChange();
    }
  }
}

private void OnDisable() {
  _anim.SetBool(ObjectConstants.Move, false);
}

private void DirectionChange() {
  _anim.SetBool(ObjectConstants.Move, false);
  _idleTimer += Time.deltaTime;

  if (_idleTimer > _idleDuration) {
    _moveLeft = !_moveLeft;
  }
}

public void Dead() {
  _rigidbody.bodyType = RigidbodyType2D.Static;
  _boxCollider.enabled = false;
}

private void MoveInDirection(int _direction) {
  _idleTimer = 0;
  _anim.SetTrigger(ObjectConstants.Move);
  Vector3 position = _enemy.position;
  float x = position.x + Time.deltaTime * _direction * _speed;
  float xDir = Mathf.Abs(_initScale.x) * _direction * _facingDirection;

  position = new Vector3(x, position.y, position.z);

  _enemy.localScale = new Vector3(xDir, _initScale.y, _initScale.z);
```

```
      _enemy.position = position;
    }
  }
}
```

HealthItem.cs

```
using Behaviour.Based.Interfaces;
using Behaviour.HealthItem;
using DG.Tweening;
using UnityEngine;

namespace Behaviour.Objects.Items {
  public class HealthItem : Item, IHealable {
    public void Heal(float value) {
      Health.OnHeal?.Invoke(value);
    }

    public override void Pick(Vector3 position) {
      base.Pick(position);

      var health = (float)Random.Range(3, 10);
      Heal(health);
      Destroy();
    }

    protected override void Destroy() {
      var sq = DOTween.Sequence();
      sq.Append(transform.DOScale(1.3f, 0.4f).SetEase(Ease.InBounce));
      sq.Append(transform.DOScale(0f, 0.2f).SetEase(Ease.Linear));
      sq.OnComplete(() => base.Destroy());
    }
  }
}
```

Health.cs

```
v using System;
using Other.Constants;
using UnityEngine;
using UnityEngine.UI;

namespace Behaviour.HealthItem {
  public class Health : MonoBehaviour{
    public static Action<float> OnHeal;
    [SerializeField] private float startingHealth;
    [SerializeField] protected Slider _slider;
    [SerializeField] private UnityEngine.Behaviour[] components;

    private float currentHealth { get; set; }
    private Animator anim;
    private bool dead;
```

```
    public virtual void Awake() {
      InitParams();
      _slider.maxValue = startingHealth;
      _slider.value = startingHealth;
    }

    private void InitParams() {
      currentHealth = startingHealth;
      anim = GetComponent<Animator>();
      OnHeal += AddHealth;
    }

    private void OnDestroy() {
      OnHeal -= AddHealth;
    }

    public void TakeDamage(float _damage) {
      currentHealth = Mathf.Clamp(currentHealth - _damage, 0, startingHealth);
      _slider.value = currentHealth;
      if (currentHealth > 0) {
        anim.SetTrigger(ObjectConstants.Hurt);
      } else {
        if (dead) return;

        Dead();
      }
    }

    public virtual void Dead() {
      _slider.value = 0;
      anim.SetTrigger(ObjectConstants.Death);

      foreach (UnityEngine.Behaviour component in components)
        component.enabled = false;

      dead = true;
    }

    private void AddHealth(float _value) {
      if (!gameObject.CompareTag("Player")) return;

      currentHealth = Mathf.Clamp(currentHealth + _value, 0, startingHealth);
      _slider.value = currentHealth;
    }
  }
}
```

Singleton.cs

```
using UnityEngine;

namespace Other {
  public class Singleton<T> : MonoBehaviour where T : MonoBehaviour {
```

```
    private static bool _mShuttingDown;
    private static readonly object MLock = new object();
    private static T _mInstance;

    private static void GetInstance() {
      if (_mInstance == null) {
        _mInstance = (T) FindObjectOfType(typeof(T));
      }
    }

    private static void StartManager() {
      lock (MLock) {
        GetInstance();
      }
    }

    private void Start() {
      StartManager();
    }

    public static T Instance {
      get {
        if (_mShuttingDown) {
          return null;
        }

        lock (MLock) {
          GetInstance();
          return _mInstance;
        }
      }
    }

    private void OnApplicationQuit() {
      _mShuttingDown = true;
    }
  }
}
```

ResultScreen.cs

```
using System;
using UnityEngine;
using UnityEngine.SceneManagement;
using UnityEngine.UI;

namespace ResultScreens {
   public class ResultScreen : MonoBehaviour, IWinResultScreen, ILoseResultScreen
   {
      private enum ResultType{
         Win,
         Lose
      }

      [SerializeField] private GameObject _prefab;
      [SerializeField] private ResultType type;
      [SerializeField] private Button _homeButton;
      [SerializeField] private Button _additionalButton;

      private void Start() {
         _prefab.SetActive(false);
      }

      public void Win() {
         _prefab.SetActive(true);
         SubscribeButtons();
      }

      public void Lose() {
         _prefab.SetActive(true);
         SubscribeButtons();
      }

      private void SubscribeButtons() {
         _homeButton.onClick.AddListener(HomeButtonClick);
         switch (type) {
            case ResultType.Win:
               _additionalButton.onClick.AddListener(NextLevelClick);
               break;
            case ResultType.Lose:
               _additionalButton.onClick.AddListener(RestartLevelClick);
               break;
            default:
               throw new ArgumentOutOfRangeException();
         }
      }

      private static void RestartLevelClick() {
         int scene = SceneManager.GetActiveScene().buildIndex;
         SceneManager.LoadScene(scene);
      }
```

```
        private void NextLevelClick() {
            var index = PlayerPrefs.GetInt("LevelIndex", 0);
            SceneManager.LoadScene(index);
        }

        private static void HomeButtonClick() {
            SceneManager.LoadScene("Menu");
        }
    }
}
```

Authentication.cs

```
using System;
using Core.Player;
using Firebase;
using Helpers;
using Other;
using UnityEngine;
using UnityEngine.SceneManagement;

namespace Ui.Auth {
  public class AuthenticationMenu : MonoBehaviour {
    [SerializeField] private AuthenticationMenuContainer _container;

    private bool login = true;

    private void Awake() {
      if (_container == null) {
        _container = GetComponent<AuthenticationMenuContainer>();
      }
    }

    private void Start() {
      _container.SaveButton.onClick.AddListener(SaveButtonClick);
    }

    private void SaveButtonClick() {
      PlayerConstants.AuthenticationErrorType taskResult =
        PlayerDataChecker.CheckUserProvidedData(
        _container.UsernameInputField,
        _container.PasswordInputField);

      switch (taskResult) {
        case PlayerConstants.AuthenticationErrorType.UsernameAlreadyExist:
          _container.ErrorTMPText.text = "Username is already exists.";
          break;
        case PlayerConstants.AuthenticationErrorType.NotValidUsername:
          _container.ErrorTMPText.text = "Your username is too short or too long.";
          break;
        case PlayerConstants.AuthenticationErrorType.NotValidPassword:
```

```
        _container.ErrorTMPText.text = "Your password is weak";
        break;
      case PlayerConstants.AuthenticationErrorType.AllGood:
        SuccessfulDataInput();
        break;
      default:
        _container.ErrorTMPText.text = "Internal Error";
        break;
    }
  }

  private void SuccessfulDataInput() {
    string id = DataGenerator.GenerateId(10);
    FirebaseDatabaseManager.Instance.SaveData(
      id,
      _container.UsernameInputField,
      _container.PasswordInputField
    );
    PlayerDataSaver.LogIn();
    PlayerDataSaver.SaveData(
      id,
      _container.UsernameInputField,
      _container.PasswordInputField);
    SceneManager.LoadScene("Menu");
  }
 }
}
```

AuthenticationMenuContainer.cs

```
using TMPro;
using UnityEngine;
using UnityEngine.UI;

namespace Ui.Auth {
 public class AuthenticationMenuContainer : MonoBehaviour {
  [SerializeField] private TMP_Text _usernameInputField;
  [SerializeField] private TMP_InputField _passwordInputField;
  [SerializeField] private Button _saveButton;
  [SerializeField] private TMP_Text _errorTMPText;
  [SerializeField] private TMP_Text _bottomButtonText;
  [SerializeField] private Button _bottomButton;

  public TMP_Text BottomButtonText {
    get {
      return _bottomButtonText;
    }
  }

  public Button BottomButton {
    get {
```

```
      return _bottomButton;
    }
  }

  public TMP_Text ErrorTMPText {
    get {
      return _errorTMPText;
    }
  }

  public string PasswordInputField {
    get {
      return _passwordInputField.text;
    }
  }

  public string UsernameInputField {
    get {
      return _usernameInputField.text;
    }
  }

  public Button SaveButton {
    get {
      return _saveButton;
    }
  }
 }
}
```

Constants.cs

```
namespace Other.Constants {
 public static class Constants {
  public const string UPPER_CHARS = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
  public const string LOWER_CHARS = "abcdefjhijklmnopqrstuvwxyz";
  public const string NUMBERS = "0123456789";
  public static string SavedFullInventory = nameof(SavedFullInventory);
  public static string SavedQuickInventory = nameof(SavedQuickInventory);
  public const string TRUE = "true";
  public const string FALSE = "false";

  public enum State {
    walking,
    hit,
    dead
  }

  public enum ItemType {
    Food,
    Poison,
    Money,
```

```
    Quest,
    Other
  }

  public enum EnemyType {
    Skeleton,
    Wizard
  }

  public enum InventoryType {
    QuickInventory,
    FullInventory,
    Inventory
  }

  public enum MoveTriggers {
    Hurt,
    Attack,
    Jump,
    Grounded,
    AnimState,
    AirSpeed,
    Death,
    Recover
  }

  public struct Tags {
    public static string Player = nameof(Player);
    public static string Enemy = nameof(Enemy);
    public static string Quest = nameof(Quest);
  }

  public enum TagsEnum {
    Player,
    Enemy,
    Drag
  }

  public enum EnemyState {
    Angry,
    Attack,
    Idle
  }
 }
}
```

FirebaseManager.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
```

```
using Firebase.Database;
using Newtonsoft.Json;
using UnityEngine;
using UnityEngine.SceneManagement;

namespace Firebase {
  public class FirebaseDatabaseManager : MonoBehaviour {
    public static FirebaseDatabaseManager Instance;
    private FirebaseDatabase _database;
    private DatabaseReference _databaseReference;
    private readonly List<string> _listOfUsernames = new List<string>();
    private readonly List<string> _listOfPassword = new List<string>();

    private void Awake() {
      if (Instance == null) {
        Instance = this;
      } else {
        Destroy(gameObject);
      }
    }

    private void Start() {
      GetInternalUsernames();
    }

    public void SaveData(string id, string username, string password) {
      _database = FirebaseDatabase.DefaultInstance;
      _databaseReference = _database.RootReference;

      var user = new User(id, username, password);
      string json = JsonConvert.SerializeObject(user);
      _databaseReference.Child("Users")
        .Child(user.username)
        .SetRawJsonValueAsync(json);
    }

    public List<string> GetListOfUsernames() {
      return _listOfUsernames;
    }

    public List<string> GetListOfPasswords() {
      return _listOfPassword;
    }

    private void GetInternalUsernames() {
      _database ??= FirebaseDatabase.DefaultInstance;
      _database.GetReference("Users/").GetValueAsync().ContinueWith(ContinuationFunction());
    }

    private Action<Task<DataSnapshot>> ContinuationFunction() {
      return task => {
            if (!task.IsCompleted) return;
```

```
            DataSnapshot snapshot = task.Result;
            snapshot.Children.ToList()
              .ForEach(
                child => _listOfUsernames.Add(child.Key));
            snapshot.Children.ToList()
              .ForEach(
                child =>
                  child.Children.ToList().ForEach(_value => {
                                        if (_value.Key == "password") {
                                          _listOfPassword.Add(_value.Value.ToString());
                                        }
                                      }));
          };
    }
  }
}
```

FirebaseAppManager.cs

```
using System.Threading.Tasks;
using UnityEngine;

namespace Firebase {
  public class FirebaseAppManager : MonoBehaviour {
    public static FirebaseAppManager Instance;
    private FirebaseApp app;

    private void Awake() {
      if (Instance == null) {
        Instance = this;
      } else {
        Destroy(gameObject);
      }
    }

    private void Start() {
      ValidateVersions();
    }

    private void ValidateVersions() {
      FirebaseApp.CheckAndFixDependenciesAsync().ContinueWith(CheckFirebaseApp);
    }

    private void CheckFirebaseApp(Task<DependencyStatus> task) {
      DependencyStatus dependencyStatus = task.Result;
      if (dependencyStatus == DependencyStatus.Available) {
        app = FirebaseApp.DefaultInstance;
      }
    }
  }
}
```

User.cs

```
using Helpers;

namespace Firebase {
  public class User {
    public string id;
    public string username;
    public string password;
    public int lastLevelIndex;

    public User(string _id, string _username, string _password) {
      id = _id;
      username = _username;
      password = _password;
      lastLevelIndex = 0;
    }
  }
}
```

AutomaticUiAnchoringEditor.cs

```
using UnityEditor;
using UnityEngine;

namespace ExternalLibs.AutomaticUIAnchoring.Editor {
  public class AutomaticUiAnchoringEditor : UnityEditor.Editor {
    private static void Anchor(RectTransform rectTransform) {
      RectTransform parentRectTransform = null;
      if (rectTransform.transform.parent) {
        parentRectTransform = rectTransform.transform.parent.GetComponent<RectTransform>();
      }

      if (!parentRectTransform) {
        return;
      }

      Undo.RecordObject(rectTransform, "Anchor UI Object");
      Rect parentRect = parentRectTransform.rect;
      rectTransform.anchorMin = new Vector2(rectTransform.anchorMin.x +
rectTransform.offsetMin.x / parentRect.width,
        rectTransform.anchorMin.y + rectTransform.offsetMin.y / parentRect.height);
      rectTransform.anchorMax = new Vector2(rectTransform.anchorMax.x +
rectTransform.offsetMax.x / parentRect.width,
        rectTransform.anchorMax.y + rectTransform.offsetMax.y / parentRect.height);
      rectTransform.offsetMin = Vector2.zero;
      rectTransform.offsetMax = Vector2.zero;
      rectTransform.pivot = new Vector2(0.5f, 0.5f);
      rectTransform.pivot = new Vector2(0.5f, 0.5f);
    }

    [MenuItem("Tools/Automatic UI Anchoring/Anchor Selected UI Objects _F1")]
    private static void AnchorSelectedUIObjects() {
```

```
    for (var i = 0; i < Selection.gameObjects.Length; i++) {
      var rectTransform = Selection.gameObjects[i].GetComponent<RectTransform>();
      if (rectTransform) {
        Anchor(rectTransform);
      }
    }
  }
 }
}
```

MeleeEnemy.cs

```
using Behaviour.HealthItem;
using JetBrains.Annotations;
using Other.Constants;
using UnityEngine;

namespace Behaviour.Enemy {
 public class MeleeEnemy : MonoBehaviour {
  [SerializeField] private float attackCooldown;
  [SerializeField] private float range;
  [SerializeField] private int damage;
  [SerializeField] private float colliderDistance;
  [SerializeField] private BoxCollider2D boxCollider;
  [SerializeField] private LayerMask playerLayer;
  private float cooldownTimer = Mathf.Infinity;

  private Animator anim;
  private Health playerHealth;
  private EnemyMovement enemyPatrol;

  private void Awake() {
   anim = GetComponent<Animator>();
   enemyPatrol = GetComponentInParent<EnemyMovement>();
  }

  private void Update() {
   cooldownTimer += Time.deltaTime;

   if (PlayerInSight()) {
    if (cooldownTimer >= attackCooldown) {
     cooldownTimer = 0;
     anim.SetTrigger(ObjectConstants.Attack);
    }
   }

   if (enemyPatrol != null)
    enemyPatrol.enabled = !PlayerInSight();
  }

  private bool PlayerInSight() {
   var bounds = boxCollider.bounds;
```

```
    var originVector = bounds.center + transform.right * range * transform.localScale.x *
colliderDistance;
    var sizeVector = new Vector3(bounds.size.x * range, bounds.size.y,
      bounds.size.z);
    RaycastHit2D hit = Physics2D.BoxCast(originVector, sizeVector, 0, Vector2.left, 0,
playerLayer);

    if (hit.collider != null) {
      playerHealth = hit.transform.GetComponent<PlayerHealth>();
    }

    return hit.collider != null;
  }

  private void OnDrawGizmos() {
    Gizmos.color = Color.black;
    var bounds = boxCollider.bounds;
    var center = bounds.center + transform.right * range * transform.localScale.x *
colliderDistance;
    var size = new Vector3(bounds.size.x * range, bounds.size.y, bounds.size.z);
    Gizmos.DrawWireCube(center, size);
  }

  [UsedImplicitly]
  private void DamagePlayer() {
    if (PlayerInSight())
      playerHealth.TakeDamage(damage);
  }
 }
}
```

ДОДАТОК Б

ВІДГУК КЕРІВНИКА ЕКОНОМІЧНОГО РОЗДІЛУ

ДОДАТОК В

Перелік файлів на магнітному носії

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

ВІДГУК

на кваліфікаційну роботу бакалавра
на тему:
"2D платформер з використанням зберігання даних на сервері за допомогою Unity 2D "
студента групи 121-18-1 Куйбіди Данила Віталійовича

Розроблена в кваліфікаційній роботі програма призначена для особистого користування та особистих цілей.

Як інструмент для проектування і реалізації було використане середовище розробки ігрових додатків Unity. Кодування додатку було реалізоване у середовищі JetBrains Rider за допомогою мови програмування C#.

Практична значимість створення даної програми полягає у наданні користувачеві пограти та дізнатися про світ піксельних платформерів.

Працездатність представленої інформаційної системи підтверджена налагоджувальними випробуваннями та тестуванням програми.

В економічному розділі визначено трудомісткість розробленої інформаційної системи, 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 освітньою програмою 121 Інженерія програмного забезпечення.

Оформлення пояснювальної записки до роботи виконано відповідно до стандартів на програмну документацію.

Кваліфікаційну роботу виконано самостійно та заслуговує оцінки 80 балів «добре», а студент Куйбіди Данила Віталійовича заслуговує присвоєння йому кваліфікації бакалавра з інженерії програмного забезпечення.

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

РЕЦЕНЗІЯ

на кваліфікаційну роботу бакалавра
на тему:
"2D платформер з використанням зберігання даних на сервері за допомогою Unity 2D "
студента групи 121-18-1 Куйбіди Данила Віталійовича

Кваліфікаційну роботу на тему "2D платформер з використанням зберігання даних на сервері за допомогою Unity 2D "виконано в повному обсязі, відповідно до технічного завдання.

Мета кваліфікаційної роботи: метою дипломного проекту є розробка 2D платформеру з використанням зберігання даних на сервері за допомогою Unity 2D системи у середовищі Jetbrains Rider .

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

Вважаю завдання і зміст кваліфікаційної роботи відповідним для перевірки ступеня підготовленості Куйбіда Данила Віталійовича за напрямом 121 Інженерія програмного забезпечення.

Список літератури, наведений в роботі, налічує 16 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 Куйбіда Данила Віталійовича заслуговує присвоєння йому кваліфікації бакалавра з інженерії програмного забезпечення.

Рецензент кваліфікаційної роботи