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 *Савостяненка Володимира Івановича*
(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 кросплатформеної мереживої гри на Unity з використанням Photon*

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)

(підпис) (прізвище, ініціали)

« » 2022 року

ЗАВДАННЯ
на кваліфікаційну роботу

бакалавра
(назва освітньо-кваліфікаційного рівня)

студента *121-18-1* *Савостяненка В.І.*
(група) (прізвище та ініціали)

тема кваліфікаційної роботи *Розробка кросплатформеної мережевої гри на Unity з використанням Photon*

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

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 _____ *Савостяненко В.І.*
(підпис) (прізвище, ініціали)

Дата видачі завдання: *14.01.2022 р.*
Термін подання кваліфікаційної роботи до ЕК: *13.06.2022 р.*

РЕФЕРАТ

Об’єкт розробки: тривимірна гра на Android та Windows з мультиплеєром та використанням Photon.

Мета кваліфікаційної роботи: створення тривимірної гри на Android та Windows, яка буде використовувати технології Photon для мереживої гри в реальному часі.

У вступі розглядається аналіз та сучасний стан проблеми, конкретизується мета кваліфікаційної роботи та галузь її застосування, наведено обґрунтування актуальності теми та уточнюється постановка завдання.

У першому розділі проаналізовано предметну галузь, визначено актуальність завдання та призначення розробки, сформульовано постановку завдання, зазначено вимоги до програмної реалізації, технологій та програмних засобів.

У другому розділі проаналізовані наявні рішення, обрано платформи для розробки, виконано проектування і розробка програми, описана робота програми, алгоритм і структура її функціонування, а також виклик та завантаження програми, визначено вхідні і вихідні дані, охарактеризовано склад параметрів технічних засобів.

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

Практичне значення полягає у створені кросплатформеного застосунку на Unity з використанням Photon.

Актуальність даного програмного продукту визначається великим попитом на подібні розробки, люди завжди грають в ігри та не уявляють своє життя без них.

Список ключових слів: МОБІЛЬНИЙ, ДОДАТОК, WINDOWS, ANDOROID, ТЕЛЕФОН, МЕНЮ. UNITY, PHOTON, КРОСПЛАТФОРМА.

ABSTRACT

Object of development: a three-dimensional game on Android and Windows with multiplayer and the use of Photon.

The purpose of the qualification work: to create a three-dimensional game on Android and Windows, which will use Photon technology for real-time online gaming.

The introduction considers the analysis and current state of the problem, specifies the purpose of the qualification work and the field of its application provides a justification for the relevance of the topic and clarifies the problem.

In the first section the subject area is analyzed, the urgency of the task and the purpose of development are determined, the statement of the task is formulated, the requirements to the software implementation, technologies and software are specified.

The second section analyzes the available solutions, selected platforms for development, designed and developed the program, describes the program, algorithm and structure of its operation, as well as calling and loading the program, determines the input and output data, describes the parameters of hardware.

The economic section determines the complexity of the developed information system, calculates the cost of work to create a program and calculates the time for its creation.

The practical value is to create a cross-platform application on Unity using Photon.

The relevance of this software product is determined by the high demand for such developments, people always play games and cannot imagine their lives without them.

List of keywords: MOBILE, APP, WINDOWS, ANDOROID, PHONE, MENU. UNITY, PHOTON, CROSSPLATFORM

ЗМІСТ

СПИСОК УМОВНИХ ПОЗНАЧЕНЬ

3D – тривимірний простір (англ. three-dimensional);

2D – двовимірний простір;

APK – Android Package;

Pun – Photon Unity Networking;

SDK – Software Development Kit;

ПК – персональний компютер.

ВСТУП

Темою даної кваліфікаційної роботи є розробка застосунку, який буде працювати на Windows та Android з використанням технології мереживого коду Photon.

Метою даної кваліфікаційної роботи є вивчення інструментальних засобів Unity, вивчення і визначення різних методів побудови застосунків на базі платформи Android і Windows. Дослідження кращого способу розробки програми, а також вивчення різних алгоритмів за визначенням об’єктів.

Unity – це процесійний багатоплатформний інструмент для розробки ігор та застосунків як 3D, так і 2D. Photon Unity Networking (PUN) – це пакет Unity для швидкого створення мереживих ігор. Швидке сполучення в мережі приводить ваших гравців до кімнат, де об’єкти можуть бути синхронізовані мережею. RPC, властивості або «низькорівневі» події фотонів – це лише деякі з функцій. Швидкий та надійний зв’язок здійснюється через віддалені сервери Photon, тому клієнтам не потрібно підключатися один до одного [17].

Інтерес до ігор завжди є. Люди вже дуже давно створюють ігрові застосунки. В Вікіпедії перша згадка про відео ігри була у 1947 році, коли вже був написаний перший ігровий застосунок. Люди, граючи в ігрові додатки, намагаються відпочити. Але всі люблять різні жанри, через це ігрових застосунків надзвичайно багато.

Комерційний зріст застосунку, що розробляється в ході кваліфікаційної роботи – надвеликий, тому що він кросплатформений, тобто його можна буде запустити на Windows та Android. На сьогодні ринок застосунків та ігор є одним з найприбутковіших.

Найбільшою перевагою у комерційному плані гри є те, що вона мережива. Ще зовсім недавно великою популярністю користувалася гра «Bomberman 2» на приставці Dendy.

Зважаючи на те, що гра «Bomberman 2» застаріла, вважається за доцільне надихнутись цим застосунком та дати нове життя грі, яка користувалася широким попитом серед багатьох користувачів.

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ ТА ПОСТАНОВКА ЗАВДАННЯ

1.1. Загальні відомості з предметної галузі

Всі ми любимо ігри. Всі ми граємо у них. Надихнувшись уже існуючою грою «Boomberman 2», відбувається розробка застосунку для кваліфікаційної роботи. «Boomberman 2» – це відеогра, яка була створена у 1991 році. Ця гра є продовженням першої частини гри «Boomberman». Жанр цього застосунку – це аркадний лабіринт. Гравець керує персонажем, який знаходиться в лабіринті, зробленим з блоків, які можна зламати та блоків, які є незламними. Кожен гравець може залишати під собою бомбу, яка вибухає через якийсь проміжок часу і руйнує стіни, що знаходяться біля неї. Також у грі є спеціальні бонуси, які дозволяють збільшити кількість одночасно створених бомб. Ціль гри – перемога над іншим гравцем.

Застосунок кваліфікаційної роботи буде створюватись за принципами ООП. Об’єктно-орієнтоване програмування (ООП) – одна з парадигм програмування, яка розглядає програму як множину «об’єктів», що взаємодіють між собою. В ній використано декілька технологій від попередніх парадигм, зокрема успадкування, модульність, поліморфізм та інкапсуляцію. Незважаючи на те, що ця парадигма з’явилась в 1960-тих роках, вона не мала широкого застосування до 1990-тих. Сьогодні багато мов програмування (зокрема, Java, C#, C++, Python, PHP, Ruby та Objective-C, ActionScript 3) підтримують ООП [35].

Об’єктно-орієнтоване програмування сягає своїм корінням до створення мови програмування Симула в 1960-тих роках. Разом із тим, як ускладнювалось апаратне та програмне забезпечення, було дуже важко зберегти якість програм. Об’єктно-орієнтоване програмування частково розв’язує цю проблему шляхом наголошення на модульності програми.

На відміну від традиційних поглядів, коли програму розглядали як набір підпрограм, або як перелік інструкцій комп’ютеру, ООП програми можна вважати сукупністю об’єктів. Відповідно до парадигми об’єктно-орієнтованого

програмування, кожний об’єкт здатний отримувати повідомлення, обробляти дані, та надсилати повідомлення іншим об’єктам. Кожен об’єкт — своєрідний незалежний автомат з окремим призначенням та відповідальністю.

1.2. Призначення розробки та галузь застосування

Розробка ігрового застосунку на Unity для Android і Windows Photon з мереживим кодом. Так ігри це одна з найбільших галузей створення застосунків. Ґрунтуючись на досвіді, отриманому в результаті роботи на велику компанію, що створює додатки на Unity, приходимо до висновків, що саме ігровий рушій Unity має безліч переваг: простий інтерфейс, використаня для написання скріптів C# та, звичайно, кросплатформенність. Є багато допоміжних плагінів та інших застосунків, які полегшують роботу в Unity. Наприклад, такий плагін як Zendject за допомогою якого легко виристовувати DI контейнери. Але нічого не заважає створити свої плагіни та розширення, які будуть полегшувати роботу.

1.3. Підстава 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 з наказом ректора.

Таким чином, підставами для розробки (виконанням кваліфікаційної роботи) є:

– освітня програма спеціальності 121 «Інженерія програмного забезпечення»;

– навчальний план та графік навчального процесу;

– наказ ректора Національного технічного університету «Дніпровська політехніка» № 268-с від 07.06.2022 р;

– завдання на кваліфікаційну роботу на тему «Створення гри кросплатформеної мереживої гри на Unity з використанням Photon ».

1.4. Постановка завдання

Завданням даної роботи є написання кросплатформеного мереживого додатку. Основними характеристиками розробки повинні бути:

– функціонал меню;

– функціонал кімнат;

– функціонал бонусів;

– можливість керування персонажем як на Andoroid, так і на Windows;

– адаптивний та зрозумілий інтерфейс.

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.

Кінцевим результатом має бути як мобільний додаток, так і Widnows застосунок, який має можливість підключення до інших гравців.

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;

– можливість підключатись до інших гравців;

– можливість керувати персонажем та взаємодіяти з оточенням та іншими гравцями в мережі;

1.5.2. Вимоги до інформаційної безпеки

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.

1.5.3. Вимоги до складу та параметрів технічних засобів

Для запуску гри потрібні такі характеристики на телефоні:

– операційна система Android 7.1.1 або вище;

– доступно не менше 100 МБ вільного місця;

– доступно не менше 100 МБ оперативної пам’яті.

Для запуску гри потрібні такі характеристики на Windows:

– операційна система Windows 7 або вище;

– доступно не менше 100 МБ вільного місця;

– доступно не менше 200 МБ оперативної пам’яті.

1.5.4. Вимоги до інформаційної та програмної сумісності

Застосунок буде сумісний з усіма Andoroid та Windows системами, які дотримуються пункту 1.5.3. Вимоги до складу та параметрів технічних засобів.

Як основна мова програмування використовувалася C#.

Додаток було написано в Unity2021.3.21f1.

РОЗДІЛ 2
ПРОЄКТУВАННЯ ТА РОЗРОБКА ПРОГРАМНОГО ПРОДУКТУ

2.1. Функціональне призначення програми

Результатом даної кваліфікаційної роботи має бути додаток Android і Windows з мереживим кодом та можливістю грати в реальному часі з другими гравцями.

Основний функціонал додатка полягає в швидкості та в добре сконструйованому застосунку, який в подальшому буде легко розширити.

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

– можливість грати будь-де та з будь-якого пристрою Windows та Android які підходять згідно технічних характеристик;

– бот, який буде вмикатися, якщо ніхто не підключиться до гри за певний проміжок часу.

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

Проєктування застосунку відбувається на базі власного досвіду роботи з Unity. Unity - це ігровий рушій на базі C++, де користувач пише код на C#.
Проєкти Unity не подібні до проєктів Visual Studio. Ви вказуєте Unity на структуру папок і вона відкриває папку як проєкт. Проєкти містять папки Assets,

Library, ProjectSettings і Temp, але єдиний, який відображається в інтерфейсі, це папка Assets, яку ви можете побачити на малюнку Рис.2.1.

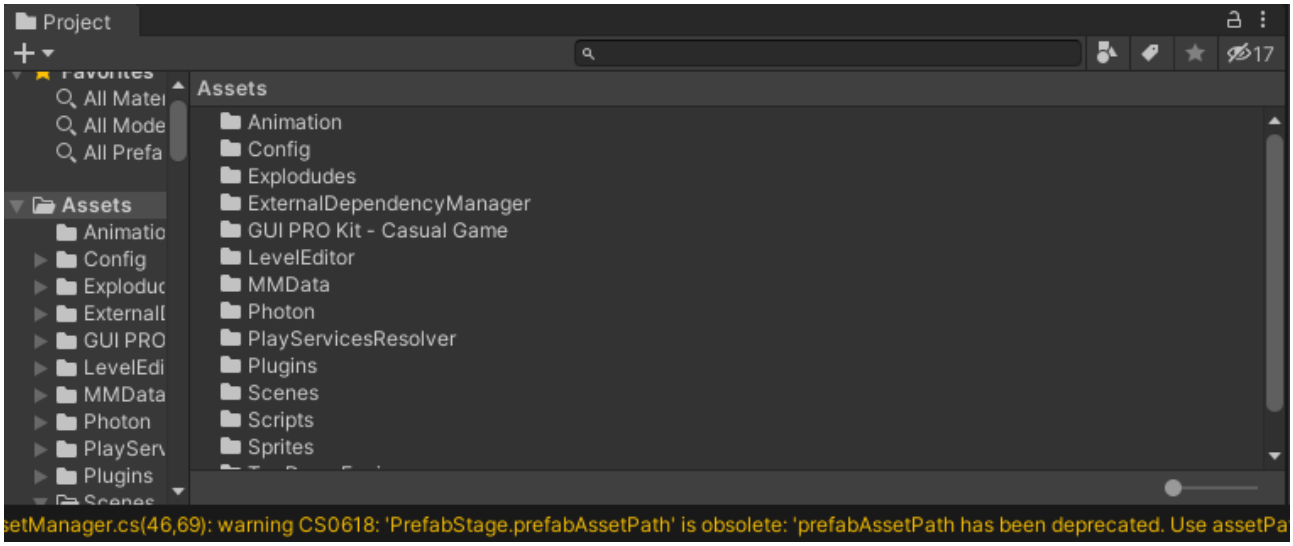

Рис.2.1. Структура проєкту в Unity

Папка Assets містить усі ваші ресурси - мистецтво, код, аудіо; кожен файл, який ви привносите до свого проєкту, йде сюди [7]. Це завжди папка верхнього рівня редактора Unity. Папка Assets – це локальний кеш для імпортованих ресурсів; він містить усі метадані для ресурсів [3]. У папці ProjectSettings зберігаються параметри. Папка Temp використовується для тимчасових файлів з Mono та Unity в процесі збирання. Важливо наголосити на важливості внесення змін тільки через інтерфейс Unity, а не безпосередньо через файлову систему [8]. Це включає навіть просте копіювання і вставку. Unity відстежує метадані для ваших об’єктів через редактор, тому використовуйте редактор для внесення змін (за винятком деяких випадків).

Майже всі у вашій сцені є GameObject. Подумайте про System.Object в .NET Framework [11]. Майже всі види походять від нього. Така ж концепція відноситься і до GameObject. Це базовий клас для всіх об’єктів у сцені Unity Рис 2.2.

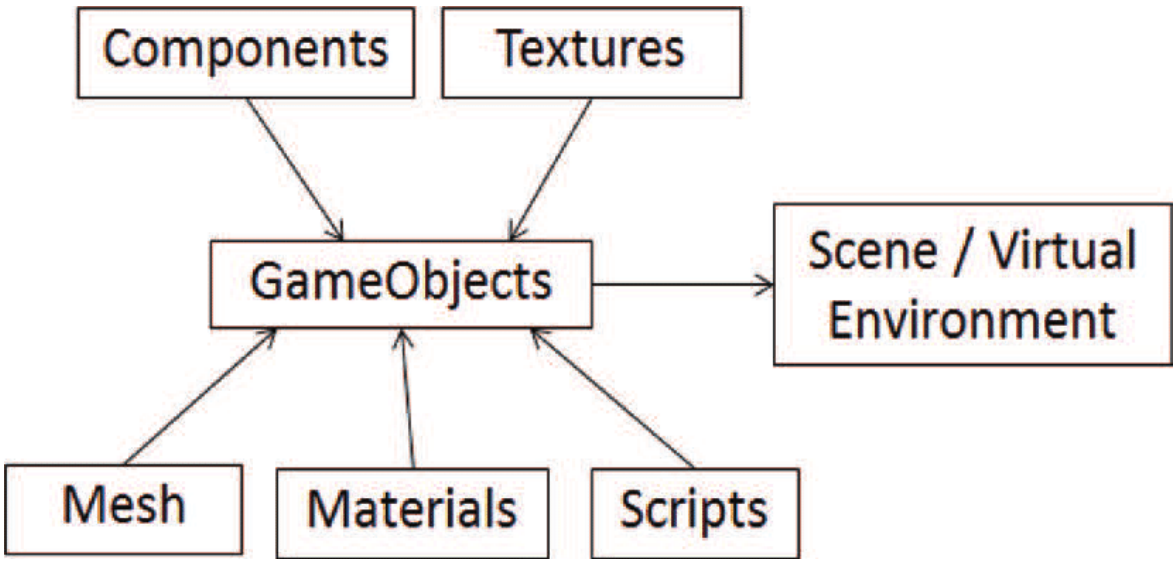

Рис.2.2. Візуалізація GameObject

Найкраще в Unity – це кросплатформеність, що полегшує роботу в декілька разів. Створити білд майже під будь-яку платформу на сьогоднішній день Рис.2.3.

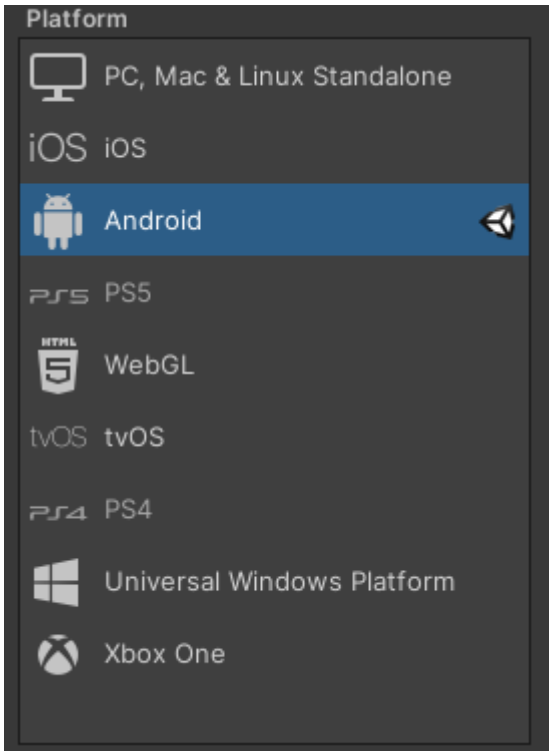

Рис.2.3. Платформи, на яких можна створити білд в Unity

Також для підвантаження ресурсів буде використано Resources. Клас Resources дозволяє знаходити об’єкти, включаючи ресурси, та отримувати доступ до них [14]. Редактор Resources.FindObjectsOfTypeAll можна використовувати для пошуку ресурсів та об’єктів Scene. Доступ до всіх ресурсів, що знаходяться в папці «Ресурси» в будь-якому місці папки «Ресурси», можна

отримати за допомогою функцій Resources.Load. Може існувати кілька папок Resources, і під час завантаження об’єктів кожна з них буде перевірена.

У Unity ви зазвичай не використовуєте імена шляхів для доступу до ресурсів, натомість ви надаєте посилання на ресурс, оголошуючи змінну поля, а потім призначаєте її в інспекторі [6]. У разі використання цієї техніки, Unity може автоматично розрахувати, які активи використовуються при побудові плеєра. Це радикально мінімізує розмір гравців до активів, які ви фактично використовуєте в побудованій грі. При розміщенні ресурсів у папках «Resources» це зробити неможливо, тому всі ресурси в папках « Resources» будуть включені до збирання [9].

Іншим недоліком використання імен шляхів є те, що це призводить до меншої кількості повторно використовуваного коду, оскільки сценарії матимуть конкретні жорстко закодовані вимоги до місця розміщення ресурсів. З іншого боку, використання посилань, що надаються в інспекторі, є самодокументованим і відразу очевидним для користувача вашого скрипта.

Однак, бувають ситуації, коли зручніше отримати актив на його ім’я, а не посилатися на нього в інспекторі. По суті, щоразу, коли незручно привласнювати посилання на об’єкт в інспекторі. Наприклад, потрібно створити ігровий об’єкт процедурно зі сценарію і призначити текстуру процедурно створеної сітці. Деякі завантажені ресурси, особливо текстури, можуть використовувати пам’ять, навіть якщо в сцені немає примірника. Щоб звільнити цю пам’ять, коли ресурс не потрібний, можна використовувати Resources.UnloadUnusedAssets [15].

Component system – це основний патерн проєктування проєктів в Unity [28]. Тому було вирішено дотримуватися цього патерну, суть якого полягає в наступному:

– один «предмет» - це один префаб;

– логіка для одного «предмета» - це один клас MonoBehaviour;

– додаток - це сцена із взаємопов’язаними префабами.

Поділ великих класів MonoBehavior.

Якби це була справжня гра, то ми спостерігали б поступове зростання обсягу класів MonoBehavior. Давайте розглянемо, як поділити їх на основі принципу одиночної відповідальності, який передбачає, що один клас повинен відповідати за один елемент. Якщо застосувати його правильно, то у вас має бути можливість дати коротку відповідь на запитання на кшталт «За що відповідає конкретний клас?» або «За що він не відповідає?». Завдяки цьому стає зрозумілим, що робить окремий клас. Це принцип, який можна застосувати до кодової бази будь-якого масштабу.

Тут можна розбити складну систему на невеликі елементи. Є безліч різних типів відповідальності для класів – ігрова логіка, обробка введення, моделювання фізики, представлення інформації та багато іншого.

Ось способи такого розбиття:

– загальна ігрова логіка, обробка введення, моделювання фізики та представлення даних можуть бути призначені MonoBehavior, об’єктам ScriptableObject або базовим класам C#;

– якщо потрібно зробити параметри доступними в Inspector, можна використовувати MonoBehavior або ScriptableObject;

– обробники подій ігрового рушія або управління існуванням об’єкта GameObject обов’язково повинні перебувати в MonoBehavior [10].

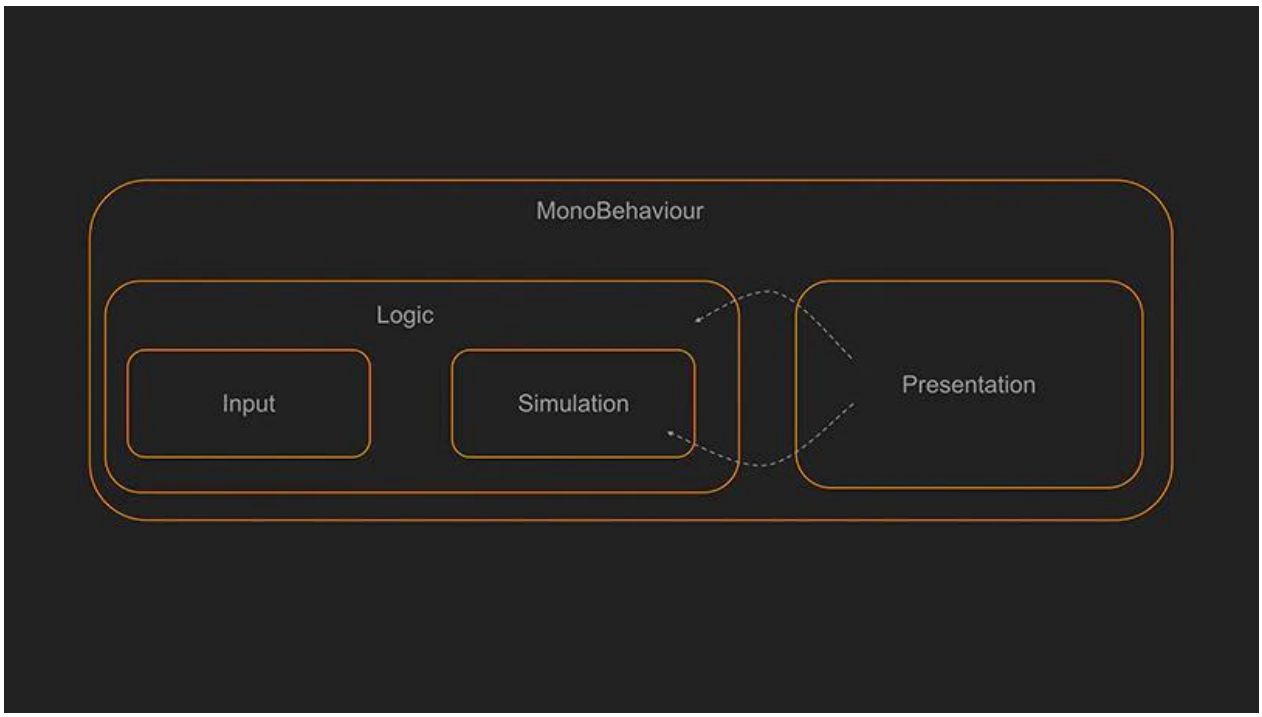

Рис.2.4. Візуалізація MonoBehaviour класів

Класам MonoBehaviour потрібно мати доступ до всієї інформації, тому що вони є обгорткою для всієї логіки, але симуляції в грі не обов'язково повинні знати, як працює логіка. Вони мають просто виконувати своє завдання. Іноді логіка передає сигнали симуляції і вона обчислює відповідну реакцію [1].

Обробці введення найвигідніше відвести власне, ізольоване місце. Тут відбуватиметься генерація подій введення та їх передача логічним алгоритмам. Все, що відбувається після, буде симуляцією.

Цей підхід добре працює з введенням та симуляцією. Але це загрожує проблемами з усім, що стосується представлення даних, наприклад, логіки, що відповідає за спецефекти, оновлення рахунку і так далі.

Статичний метод хороший тим, що якщо він не торкається глобальних змінних, то нам легко визначити зону відповідальності цього методу, просто глянувши на аргументи, що використовуються при виклику методу. Нам не потрібно аналізувати його реалізацію [2].

Цей підхід перетинається із областю функціонального програмування. Базова одиниця тут така: ви надсилаєте дані функції, а функція повертає результат або змінює один із зовнішніх параметрів. Спробуйте цей підхід і,

можливо, ви дійдете висновку, що він дає менше багів у порівнянні з класичним об’єктно-орієнтованим програмуванням. Також статичні методи дуже сильно допомагають з розширеннями або прорахунками, бо їх можна винести окремо від основної логіки.

2.4. Опис використаних технологій та мов програмування

Мова програмування C#. У процесі розробки використовували такі технології, як:

– JetBrains Rider Editor;

– TextMeshPro;

– Unity UI;

– Unity physics;

– Photon PUN;

– Quick Search;

– 2D Sprite;

– Assets Bundle;

– Device Simulator;

– ImageConversion;

– Memory Profiler;

– Nice Vibration;

– Animator.

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. Також дозволяє дебажити код.

TextMeshPro – це ідеальне текстове рішення для Unity. Це ідеальна заміна тексту інтерфейсу Unity і застарілої текстової сітки [20]. Потужний і простий у використанні, TextMeshPro (також відомий як TMP) використовує розширені

методи рендерингу тексту разом з набором шейдерів користувача [21]. Забезпечуючи значні покращення візуальної якості, надаючи користувачам неймовірну гнучкість, коли справа доходить до стилю тексту та текстурування. Має чудову продуктивність. Оскільки геометрія, створена TextMeshPro, використовує два трикутники на символ так само, як і текстові компоненти Unity, це покращена візуальна якість і гнучкість, що не потребують додаткових витрат на продуктивність [29].

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 [22].

Unity physics – реалізує тривимірну фізику в Unity [22].

Photon PUN (Photon Unity Networking) є клоном оригінального мереживого API Unity, що працює на надійній інфраструктурі Photon [16]. Крім повного підбору гравців, основними PUN є:

- серіалізація станів ігрових об’єктів (з вбудованою підтримкою перетворень тощо);
- видалені івенти процедур (RPC).

PUN дає розробнику прямий та повний контроль над тим, що відправляють та отримують у поєднанні з його гнучкою багатоадресною моделлю ретрансляції приміщень [11].

Quick Search – для швидкого пошуку об’єктів сцени, пункти меню, пакети, API, налаштування тощо [26].

2D Sprite – вікно редактора спрайтів Unity для створення та редагування властивостей об’єктів Sprite, таких як опорна точка, межі та фізична форма.

Модуль AssetBundle реалізує клас AssetBundle і пов’язані API для завантаження даних з AssetBundles.

Device Simulator є альтернативою традиційному вікну гри редактора Unity. Імітуючи поведінку класу Screen і SystemInfo, Device Simulator прагне дати точне уявлення про те, як програма буде виглядати на пристрої [27].

Модуль ImageConversion реалізує клас ImageConversion, який надає допоміжні методи для перетворення зображень у формати PNG, JPEG або EXR.

Memory Profiler пропонує уніфіковане рішення, яке дозволяє створювати профіль як невеликих проектів на мобільних пристроях, так і великих проектів AAA на високоякісних машинах. Він надає корисну інформацію про виділення пам’яті в рушії, щоб дозволити розробникам керувати та зменшувати використання пам’яті [31].

Nice Vibrations – це просте, але потужне рішення для додавання вібрацій та тактильних зворотних зв'язків у ваші ігри [4]. Побудований на основі власних API iOS та Android, він пропонує універсальний інтерфейс для одночасної взаємодії з геймпадами, iOS та Android, а також способи отримання правильної вібрації на кожній платформі [5].

Animator надає всі необхідні інструменти та компоненти часу для виконання скелетної анімації за допомогою спрайтів [24].

Також хотілось розповісти про отпимізацію в Unity.

Для відображення об'єкта на екрані двигун відправляє команду (draw call) графічному API (наприклад, OpenGL або Direct3D). Графічний API робить значну роботу для кожного DC, що дуже впливає на продуктивність CPU [33].

Unity використовує дві методики для вирішення цієї проблеми.

Dynamic batching: для досить малих сіток це перетворює їх вершини в ЦП, групує багато подібних вершин разом і малює їх усі за один раз.

Static batching: об’єднує статичні (не рухомі) об’єкти гри у великі сітки та швидше відтворює їх.

Батчаться тільки об'єкти, що мають один і той самий матеріал. Відповідно, для ефективного батчингу вам необхідно робити матеріали загальними для безлічі об'єктів, якщо це можливо.

Якщо у вас є два однакові матеріали, що відрізняються лише текстурами, можна об'єднати ці текстури в одну велику – процес, який часто називають створенням текстурного атласу. Так ви зможете використати один матеріал замість двох.

На рисунках Рис.2.5., Рис.2.6., Рис.2.7., Рис.2.8. бачимо статистику на сцена до та після отпимізаціїї сцен.

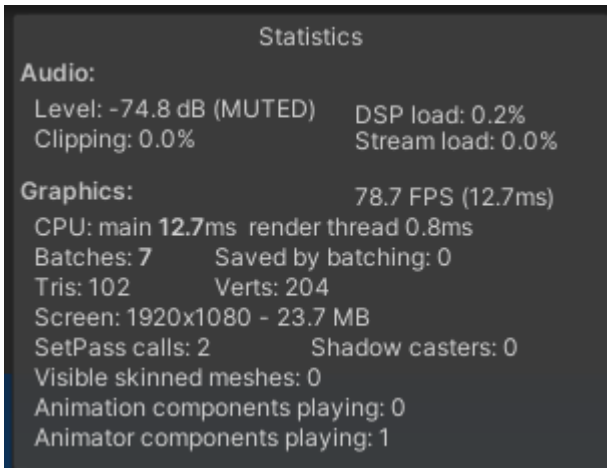

Рис.2.5. Показники в головному меню до оптимізації

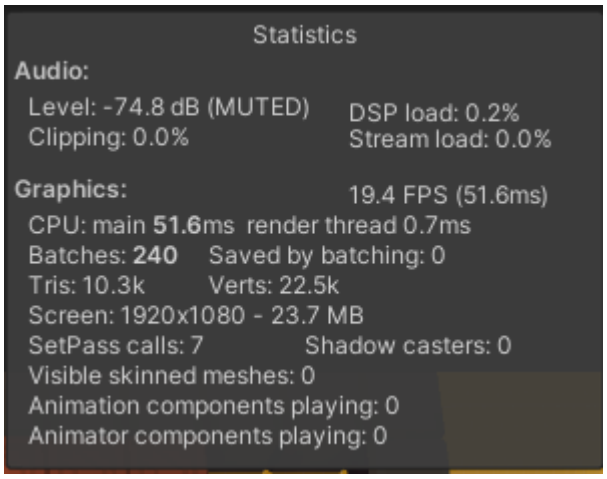

Рис.2.6. В показники ігровії сцені до оптимізації

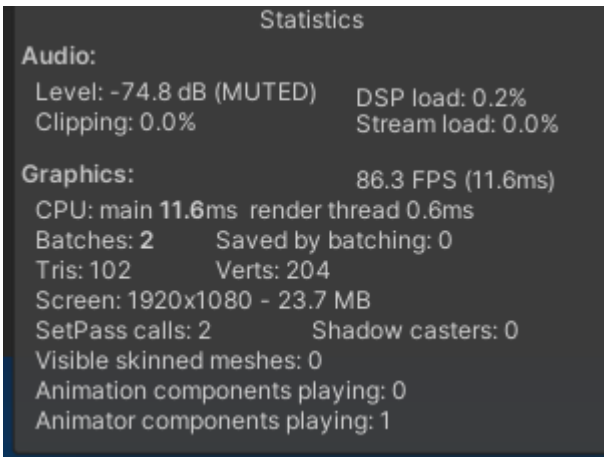

Рис.2.7. Показники в головному меню після оптимізації

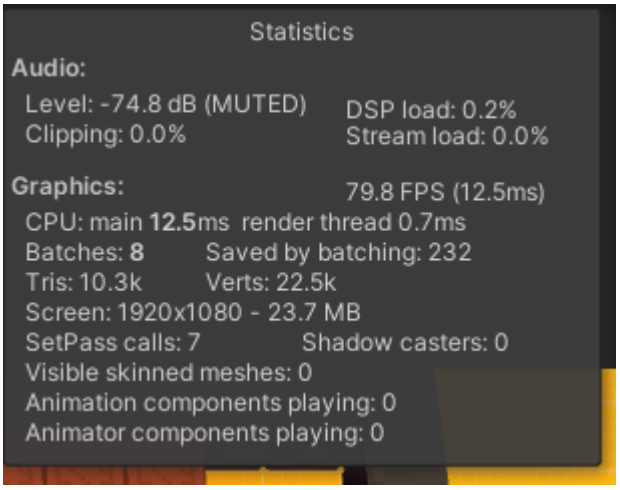

Рис.2.8. Показники в ігровії сцені після оптимізації

2.5. Опис структури програми та алгоритми її функціонування

Гра поділена на 2 основні сцени. Сцени містять об’єкти гри. Їх можна використовувати для створення головного меню, окремих рівнів та всього іншого. У кожній сцені ви будете розміщувати свої оточення, перешкоди та прикраси, по суті, проектуючи та будуючи свою гру частинами. У даному застосунку все розділено дуже просто: перша сцена – це меню та прелоадер. Друга – це все ігрова сцена.

То ж, в першій сцені використовується логіка ініціалізації на Photon. Поки у нас триває ініціалізація, ми бачимо екран завантаження Рис.2.9 з анімацією завантаження.

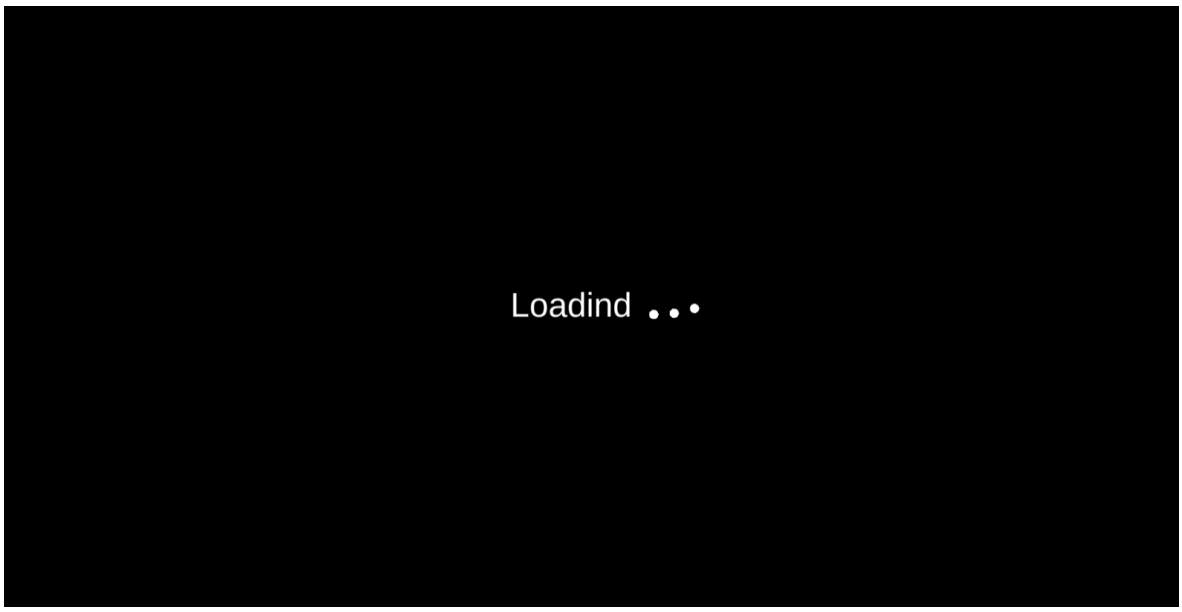

Рис.2.9. Екран завантаження

Далі після ініціалізації та підключення до Photon, ми потрапляємо до головного меню Рис.2.10.

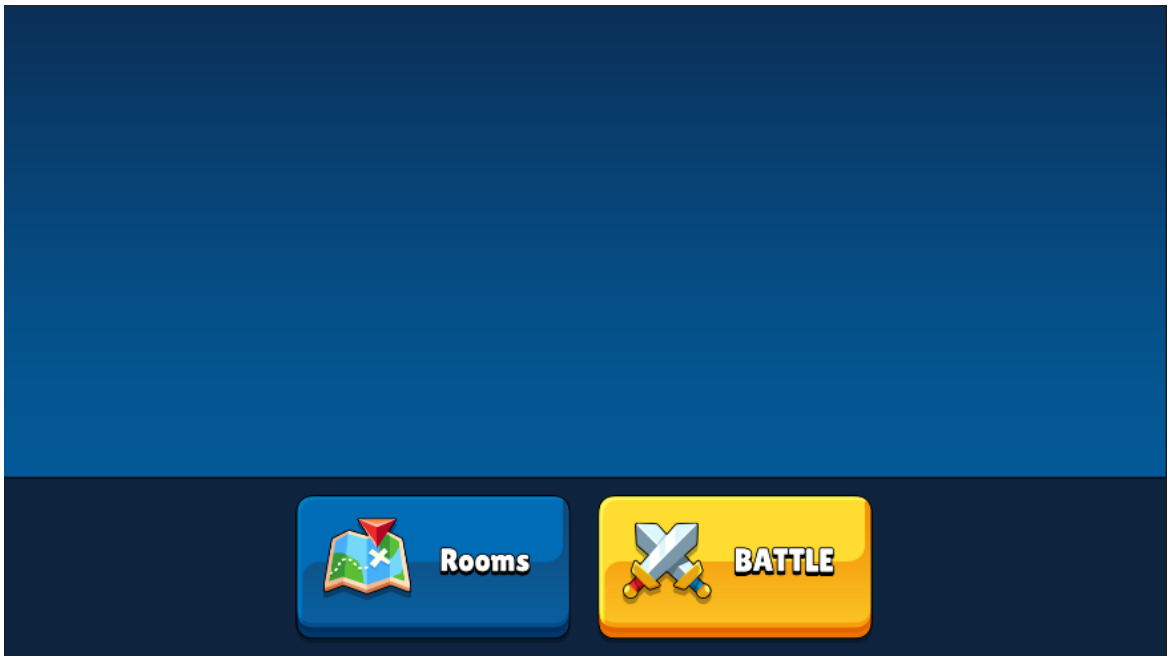

Рис.2.10. Головне меню

У головному меню ми бачимо дві кнопки: одна з них – це швидкий пошук гри, а друга кнопка відкриває панельку з кімнатами, до яких можна приєднатися. В списку вибору кімнат у нас є пошук кімнати та зручне приєднання до кімнати лише натисканням по ній. Кнопка швидкого пошуку гри має такий фунціонал як

приєднатись до будь-якої уже створеної кімнати, якщо ж кімната відсутня, то ми створюємо нову кімнату. Варіанти розвитку події в головному меню Рис2.11.

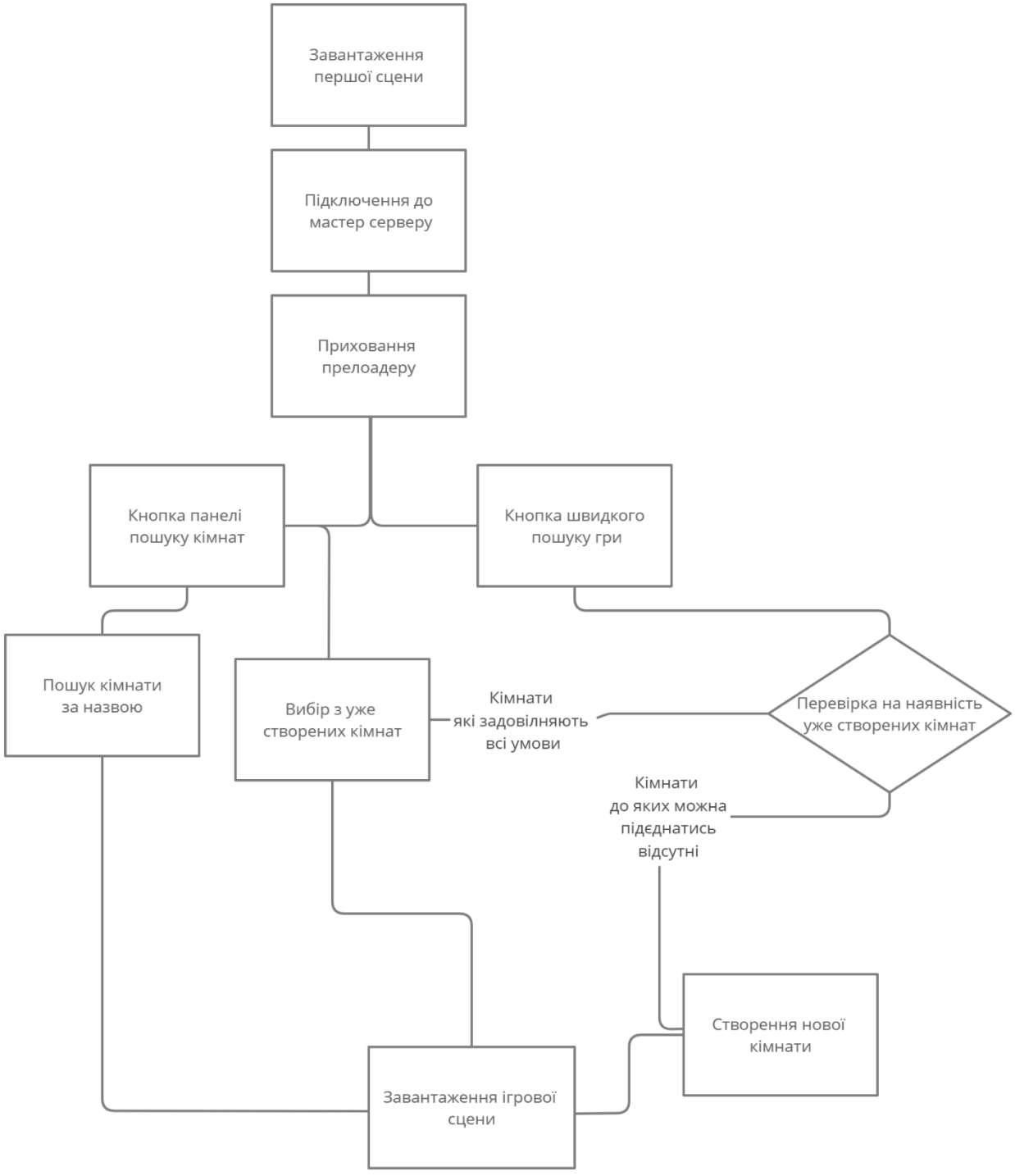

Рис.2.11. Варіанти розвитку події в головному меню

Під час приєднання або створення нової кімнати, ми бачимо екран завантаження Рис.2.9, доки у нас вантажиться ігрова сцена. Ігрова ж сцена має так само як і головне меню після екран завантаження. На цьому екрані ми

очікуємо підключення іншого гравця. Якщо за певний проміжок часу ми не маємо ще одного гравця, то ми створюємо бота, який буде грати з першим гравцем. На ігровій сцені ми маємо ігрове поле. Ігрове поле складається з сітки. Гравець може рухатись лише по клітинках цієї сітки Рис.2.12.

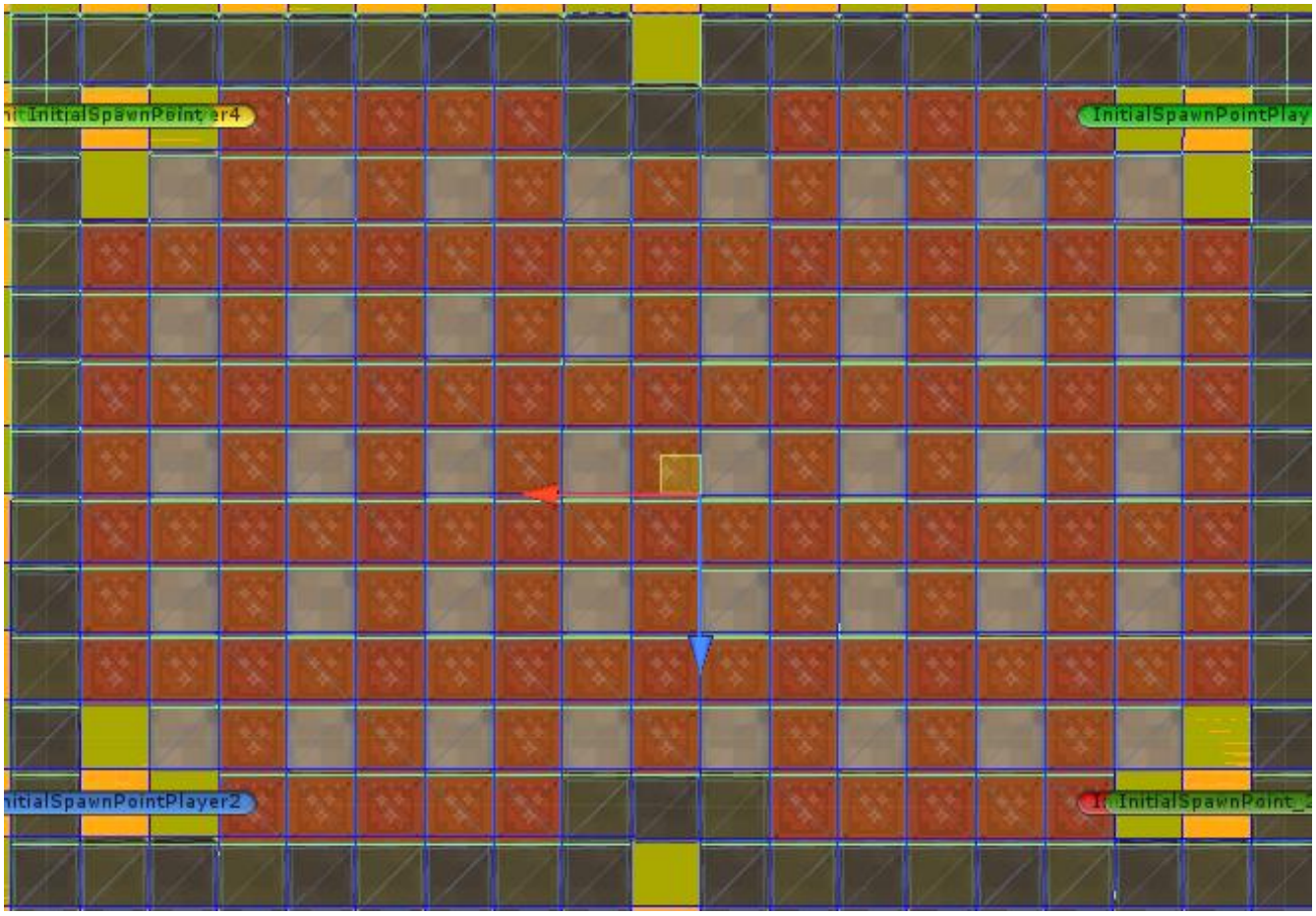

Рис.2.12. Сітка ігрового поля

Керування на смартфоні відбувається через джойстик та кнопку створення бомби Рис.2.13.

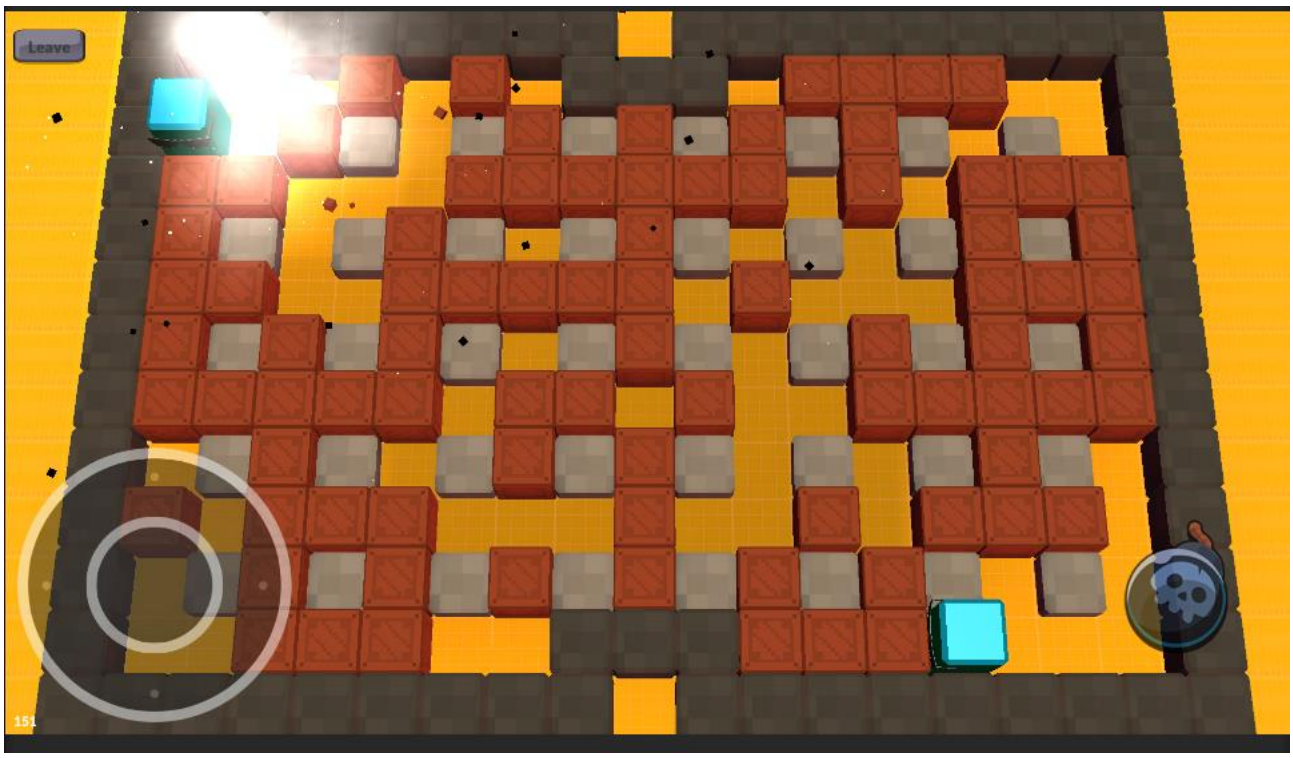

Рис.2.13. Скріншот з ігрової сцени

Ящики, які можна зруйнувати та стіни самого ігрового поля, які неможливо зруйнувати, знаходяться на ігровому полі. Кожен гравець напочатку гри має по 2 бомби, яка може вибухнути. Вибух має певний діапазон руйнування. Від центральної точки – це 4 клітинки вліво, чотири клітинки вправо, по чотири клітинки вгору та вниз від бомби та клітинка, в якій знаходиться бомба Рис.2.14.

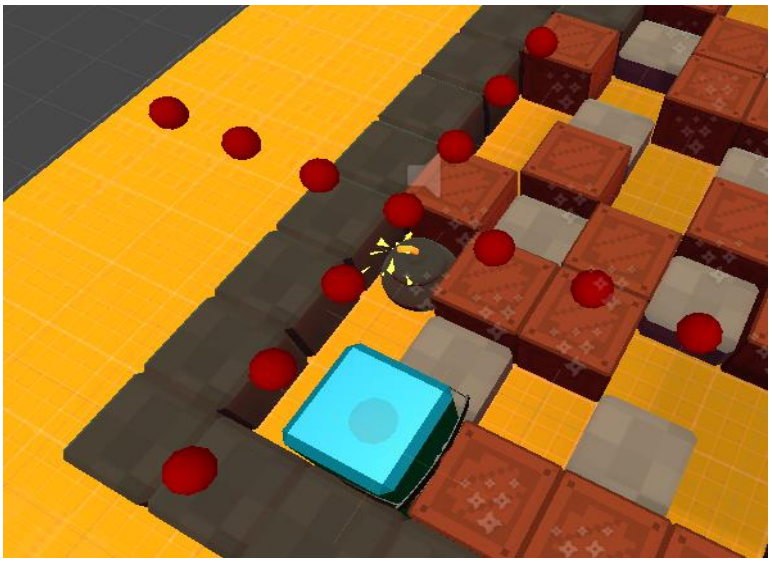

Рис.2.14. Вибухова хвиля дії бомби

Якщо вибух потрапляє в клітинку, де знаходиться або блок, що не руйнується, або який може бути зруйнований, то вибухова хвиля зупиниться в цьому блоці.

Для прорахунку вибухової хвилі використовуються колайдери. З деяких блоків може випасти бустер, який буде збільшувати кількість одночасно створених бомб гравцем Рис.2.15.

Рис.2.15. Бустер, який збільшує кількість бомб, які можна створити

Далі трохи більше інформації важливо надати про Photon Рис.2.16, на цій блок схемі представлена майже вся система Photon. Стисло – Photon це дуже потужна, надійна та гнучка система. Опанувати цю систему дуже просто завдяки зрозумілій документації і прикладах використання [11].

Всі клієнти підключаються до Photon Cloud, так як сервера знаходяться в різних куточках світу [18]. Це дає можливість розподілити навантаження на сервери та підтримувати низьку затримку. Знаходження найкращого сервера здійснюється динамічно. Далі в самій Unity через компонент PhotonView ми полегшуємо роботі ігрових об‘єктів [13].

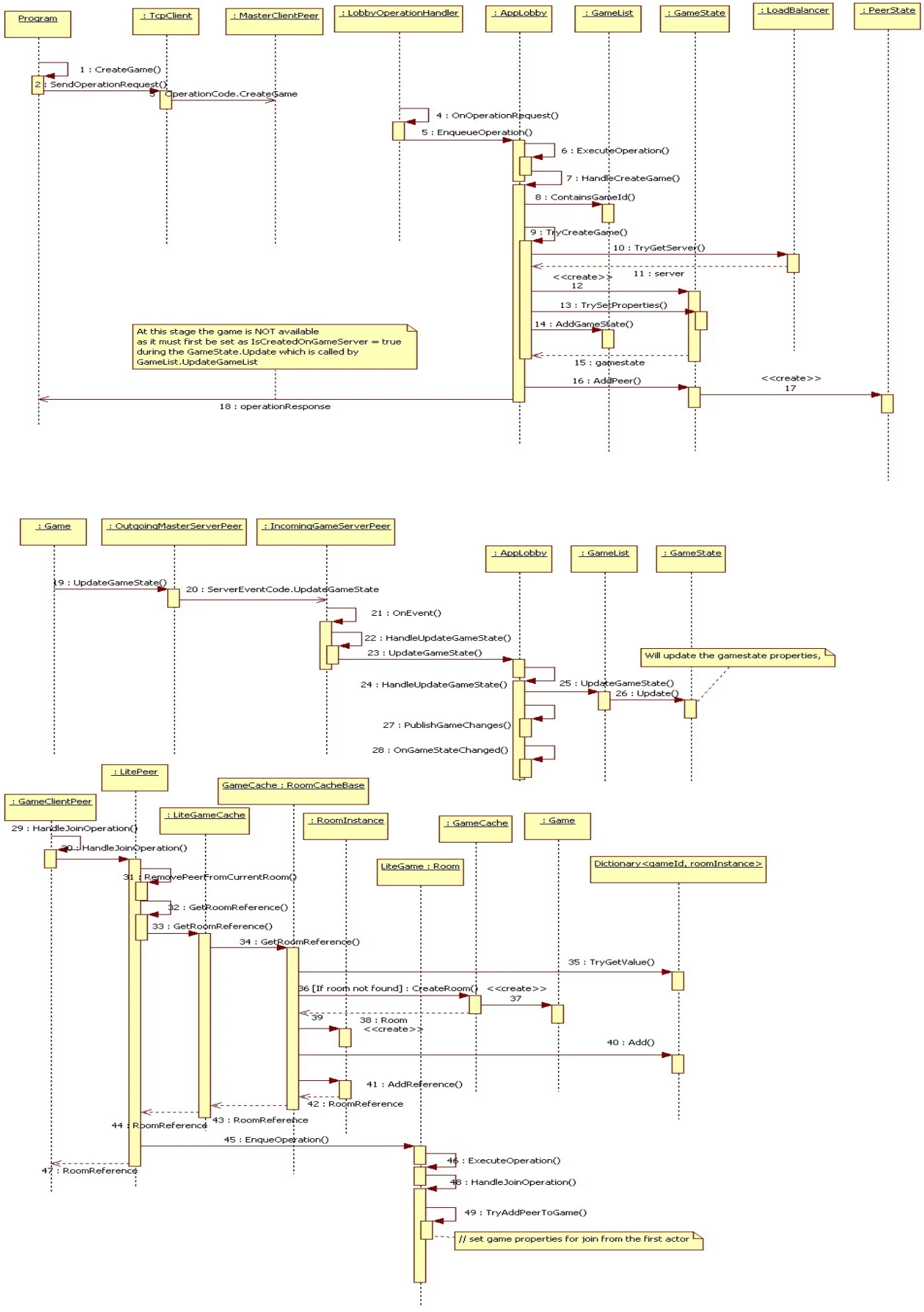

Рис.2.16. Блок схема роботи Photon

Для синхронізації позицій гравця використовуємо PhotonTransformView [12]. Це базовий клас, який долучається до проєкту при підключенні Photon.

Також PhotonNetwork дає нам змогу використовувати зворотні виклики, які були використано для синхронізації ігрового поля. Ми перетворюємо список з ящиками, які можна зруйнувати в BitArray, далі конвертуємо byte[] та відправляємо від мастер клієнта до користувача, який підключився до кімнати.

Наразі, необхідно надати більше інформації стосовно бота та його роботи. Бот створюється під час очікування підключення другого гравця. Якщо другий гравець не підключається, то через деякий проміжок часу створюється бот, який прораховує клітинки та має можливість знаходити найкращі позиції рухатись до них та створювати бомби.

Приклади роботи бота можна побачити на наступних рисунках.

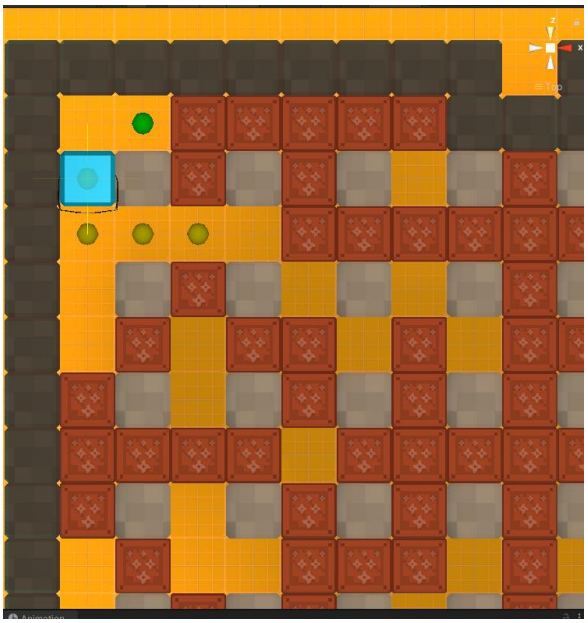

Рис.2.17. Знаходження ботом найкращої клітинки для створення бомби

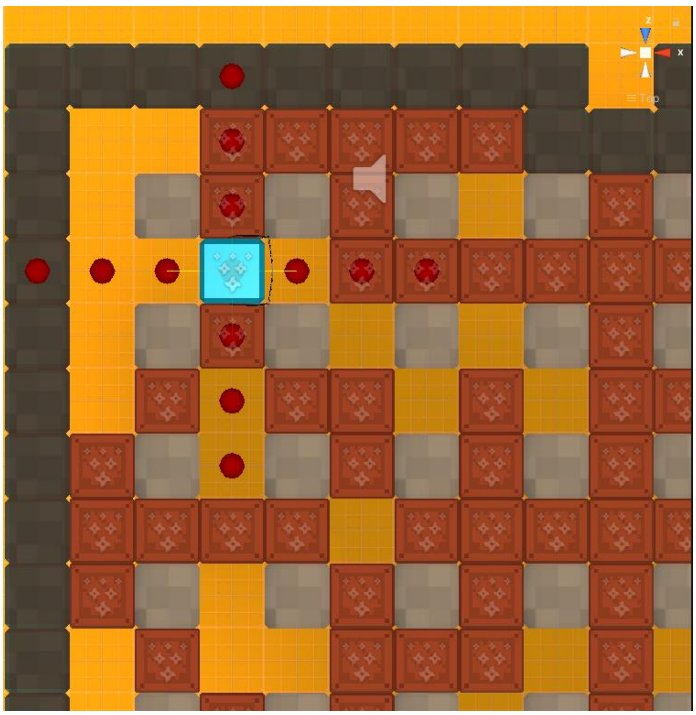

Рис.2.18. Створення ботом бомби та прорахунок вибуху бомби

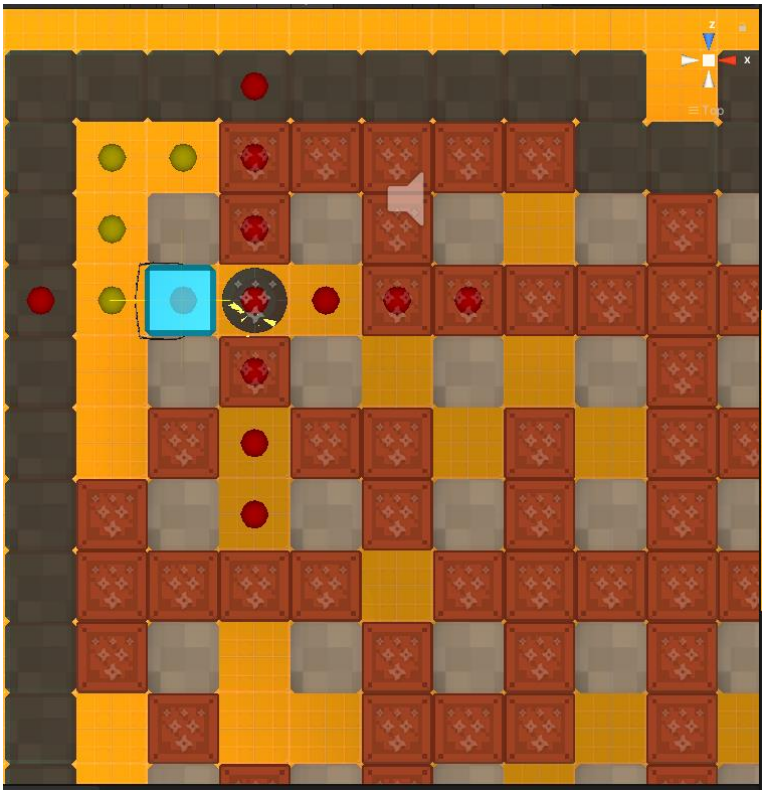

Рис.2.19. Переміщення ботом до безпечної позиції

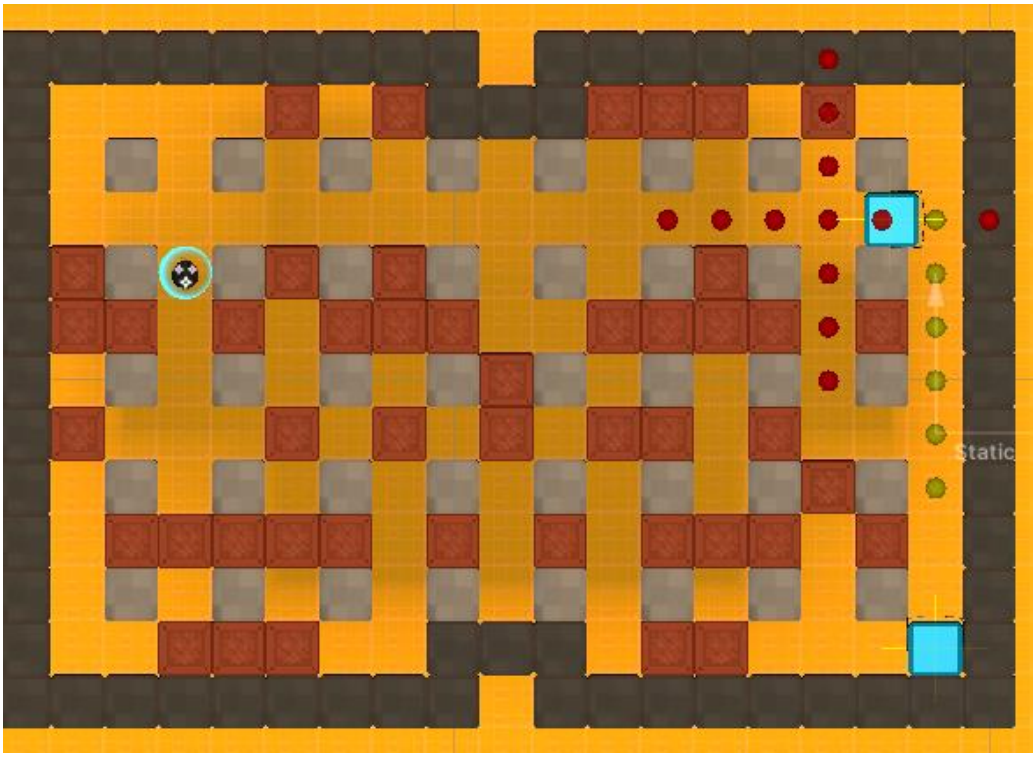

Рис.2.20. Знаходження ботом та переміщення на небезпечну позицію для гравця

2.6. Обґрунтування та організація вхідних та вихідних даних програми

Аналізуючи те, що гра буде кросплатформеною, ми мали змогу підібрати зрозумілий інтерфейс, який повністю забезпечує зручне використання гри у користувача. Для керування гри на компютері доцільно буде використовувати мишку та клавіатуру. На смартфоні, звичайно, сенсорний тап. Щодо девайсу інтуїтивне управління ігрою за допомогою тапів та інтерфейсу, який буде підлаштовуватися для гравця в ігровій сцені для керування персонажем.

Вхідні дані, які застосунок отримує від користувача, тобто вводяться завдяки сенсорному екрану на смартфоні на ПК, відповідно через мишку та клавіатуру.

Вихідні дані, які генеруються під час роботи гри, при обробці даних, які отримує гравець – це реакція інтерфейсу на натискання кнопок та клавіш або тапу по екрану. В ігровій сцені – це переміщення позиції гравця або створення бомб.

2.7. Опис розробленого програмного продукту

Всі додатки на Android встановлються через APK або AAB. AAB використовується для встановлення з PlayMarket. В ході тестування, було встановлено гру через APK. Для цього не потрібно було нічого, окрім самого девайсу на Android.

У ході тестування мобільного додатку було виявлено, що гра працює коректно. Але на Samsung J5, який був придбаний у 2015 році, гра запустилась та працювала коректно, проте показники FPS були дуже низькі від 5 до 10. На всіх інших девайсах ніяких проблем з підключенням, або ж іншими показниками виявлено не було.

На Windows після розробки білда проєкту, була створена папка, в якій ми містимо наступні файли, показані на Рис.2.21. У нас є декілька папок та файлів. Для початку роботи застосунку ми використовуємо “Bomb.exe”. Загальна вага всіх файлів складає 100Мб.

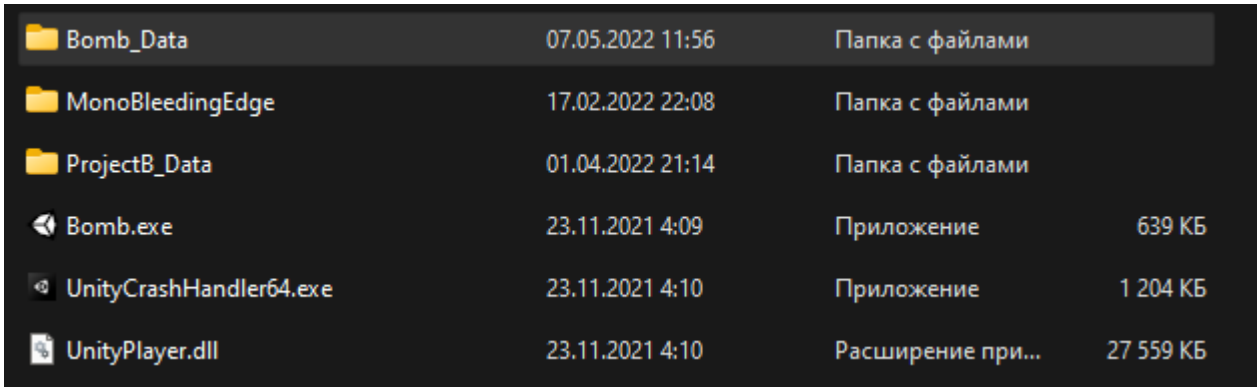

Рис.2.21.Результат білду проєкту під Windows.

2.7.1. Використані технічні засоби

При розробці програми була використана ОЕМ з наступними характеристиками:

– Процесор Intel Core i5-7300HQ;

– Відеопроцесор Nvidia GeForce GTX 1050 (4 ГБ GDDR5);

– Оперативна пам’ять 16 ГБ;

– SSD Samsung EVO 970.

Тестування проводилося на таких смартфонах:

– Meizu 16 X;

– Samsung S8;

– Huawei P Smart Plus;

– Samsung G5;

– Lenovo Y520;

– Lenovo Y530;

– Lenovo ideapad gaming 3;

– Xiaomi 11Pro.

2.7.2. Використані програмні засоби

– Unity;

– JetBrains Rider;

– Gimp;

– Fork Git;

– Blender;

– Git.

Гра створена в Unity. У процесі тестування також було викростано багато смартфонів. Для тестування на різних версіях використовувався також і смартфон із системою IOS Iphone 12 mini через емулятор Unity Remote 5.

JetBrains Rider – кросплатформене інтегроване середовище розробки програмного забезпечення для платформи .NET. Підтримуються мови програмування C#, VB.NET та F# [32].

GIMP – растровий графічний редактор, що вільно розповсюджується, програма для створення та обробки растрової графіки та із частковою

підтримкою роботи з векторною графікою. Є безкоштовним аналогом відомого всім Photoshop.

Fork – візуальний гіт, дуже зручний та з можливістю створювати свої шоткати для швидкої роботи з гітом.

Blender — програмний пакет для створення тривимірної комп'ютерної графіки, що включає засоби моделювання, анімації, рендерінгу, після-обробки відео. Як на мене найкращій засіб моделювання на сьогоднішній час.

Git – вражаюча система контролю версій, за допомогою якої не можливо створювати проекти. Спираючись на досвід роботи з Git, можемо прийти до висновку про відсутність недоліків цієї системи [30].

2.7.3. Виклик та завантаження програми

Після загрузки файлу APK, необхідно пройти етапи встановлення додатку на смартфон за рекомендаціями від Android. Наступним етапом буде запуск додатку на девайсі.

На Windows завантаження програми відбудеться через клік по “Bomb.exe”.

2.7.4. Опис інтерфейсу користувача

Інтерфейс гри був створений зручним та зрозумілим для гравців. Інтуїтивно зрозумілий інтерфейс не заважає та є мінімалістичним. Далі представлені приклади інтерфейсу застосунку кваліфікаційної роботи.

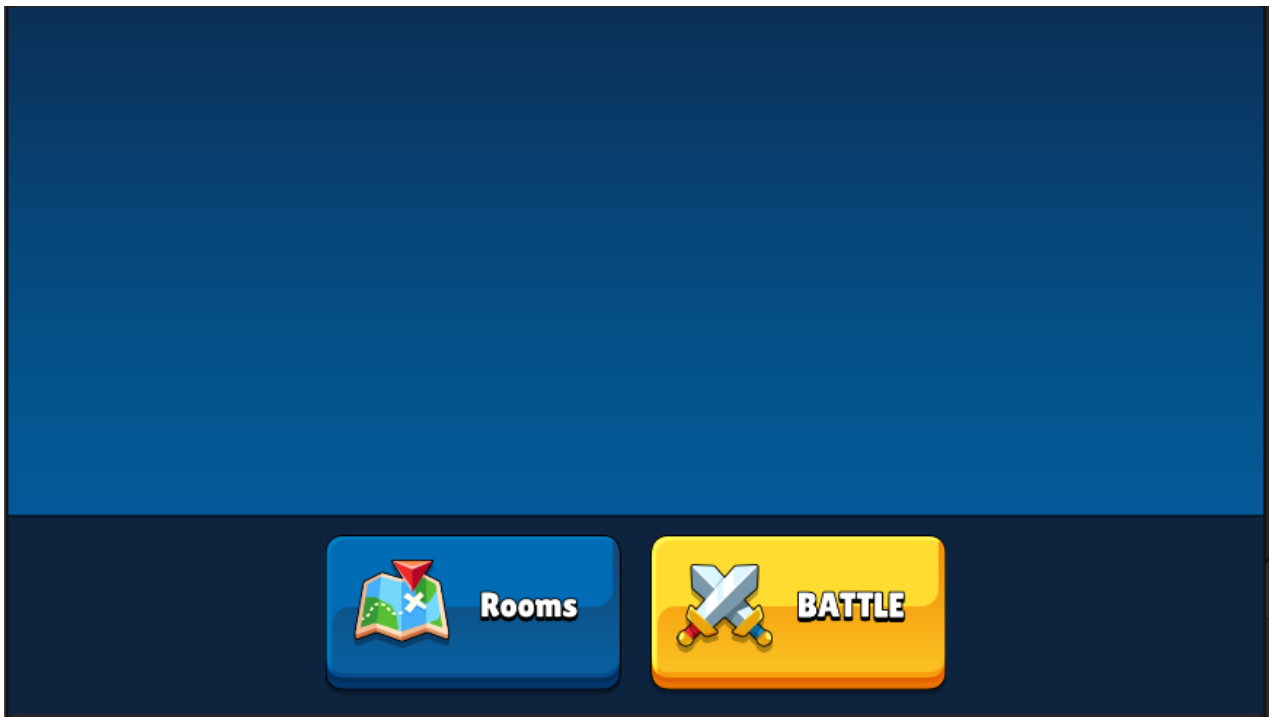

Рис.2.22. Головне меню

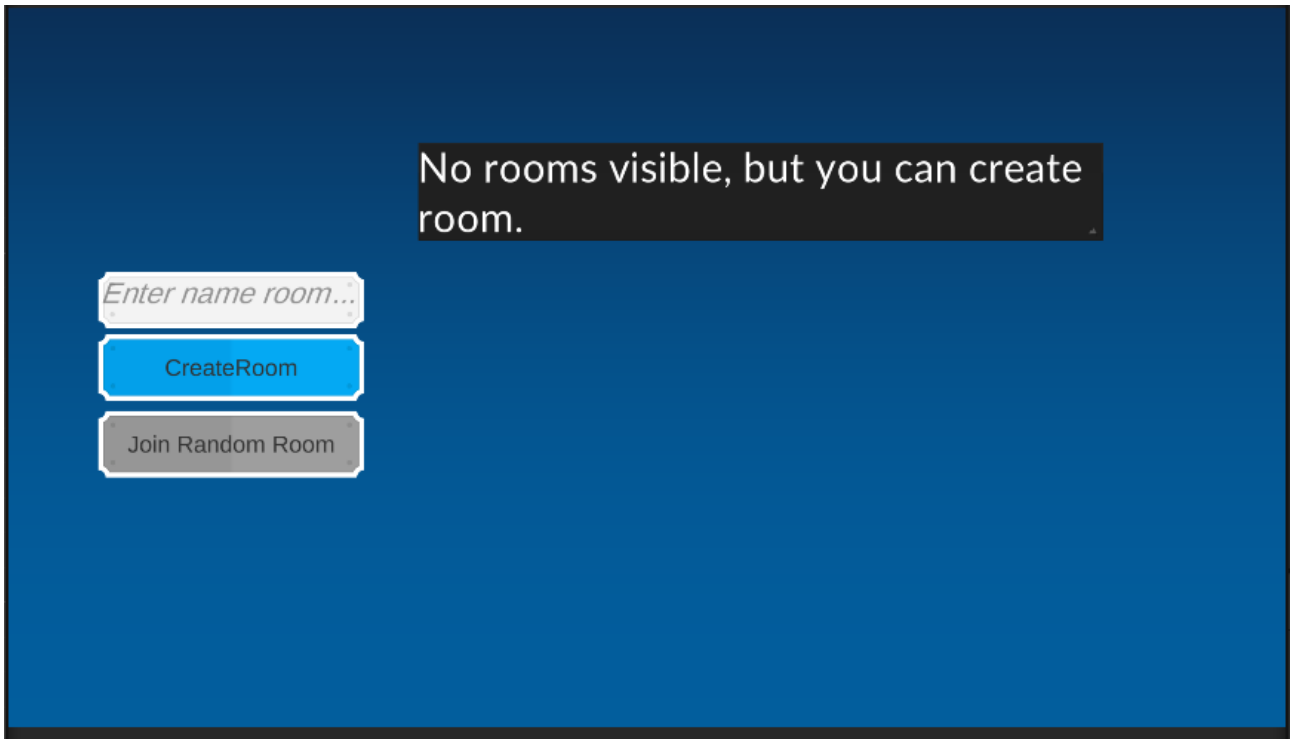

Рис.2.23. Панель пошуку ігор з відсутніми кімнатами

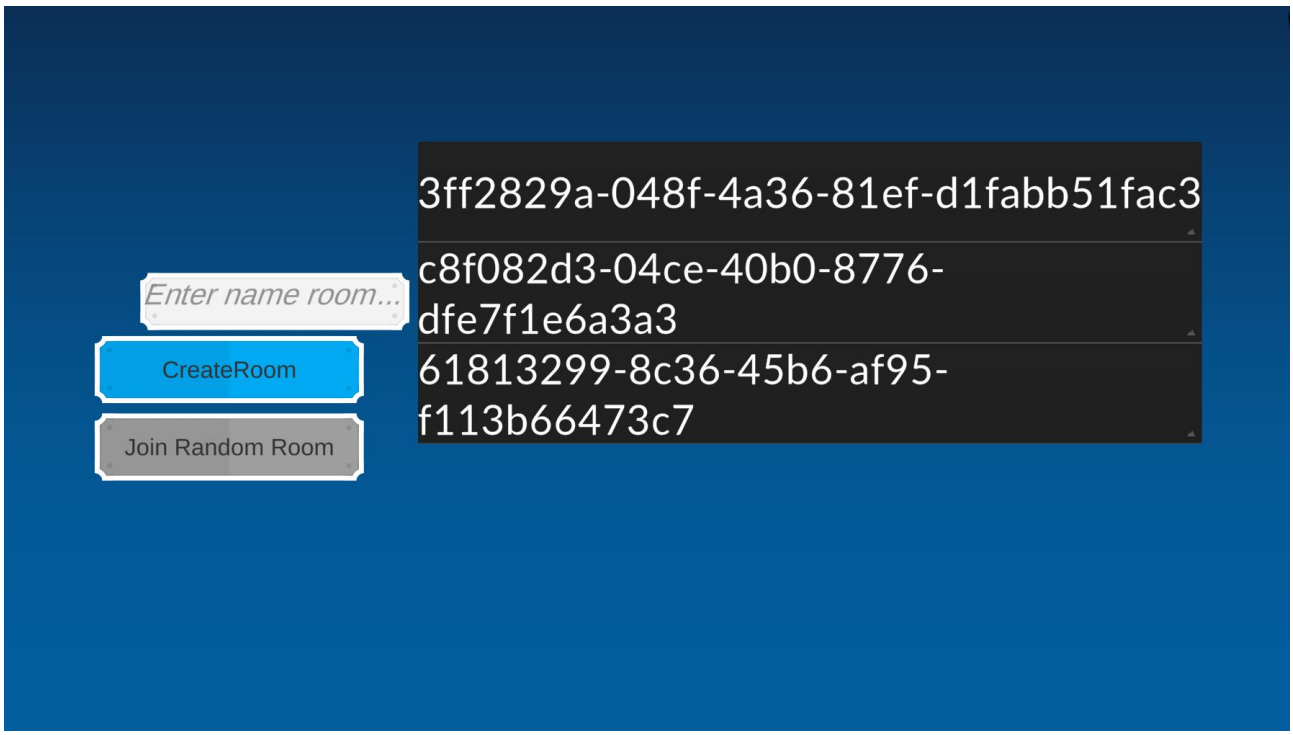

Рис.2.24. Панель пошуку ігор з створеними кімнатами

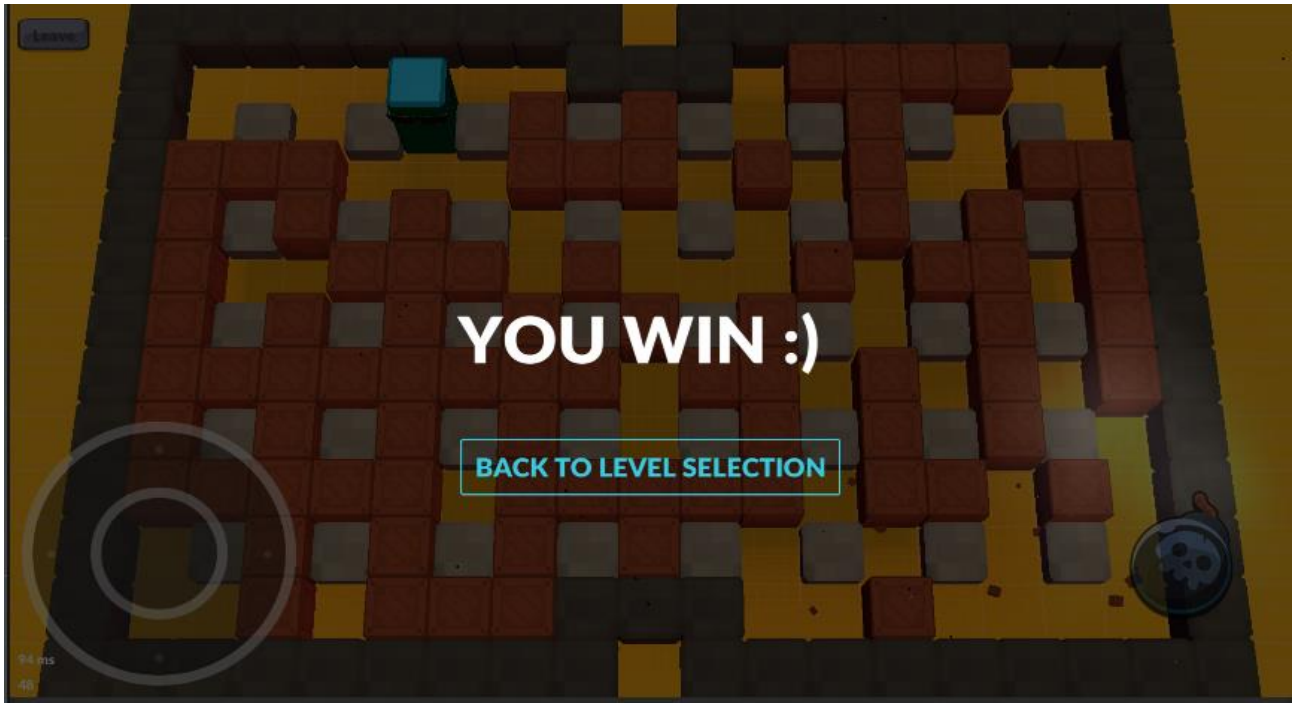

Рис.2.25. Екран виграшу

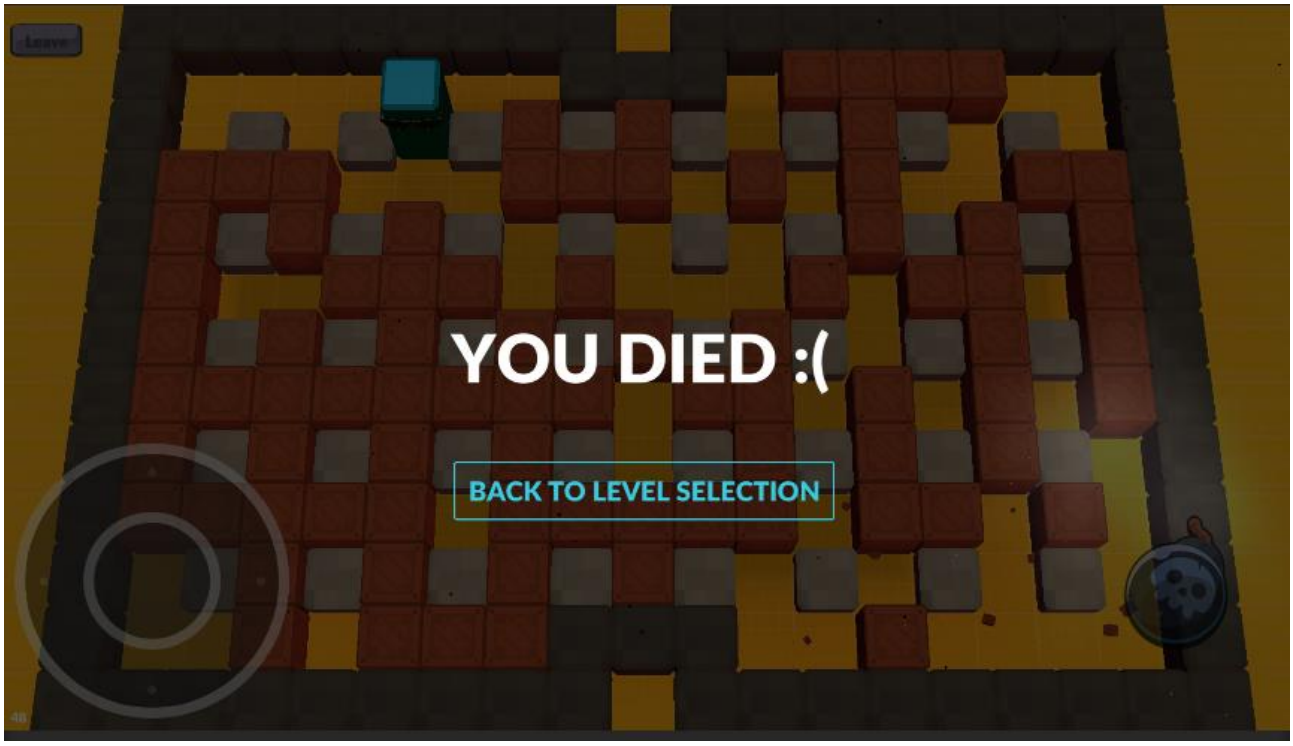

Рис.2.26.Екран програшу

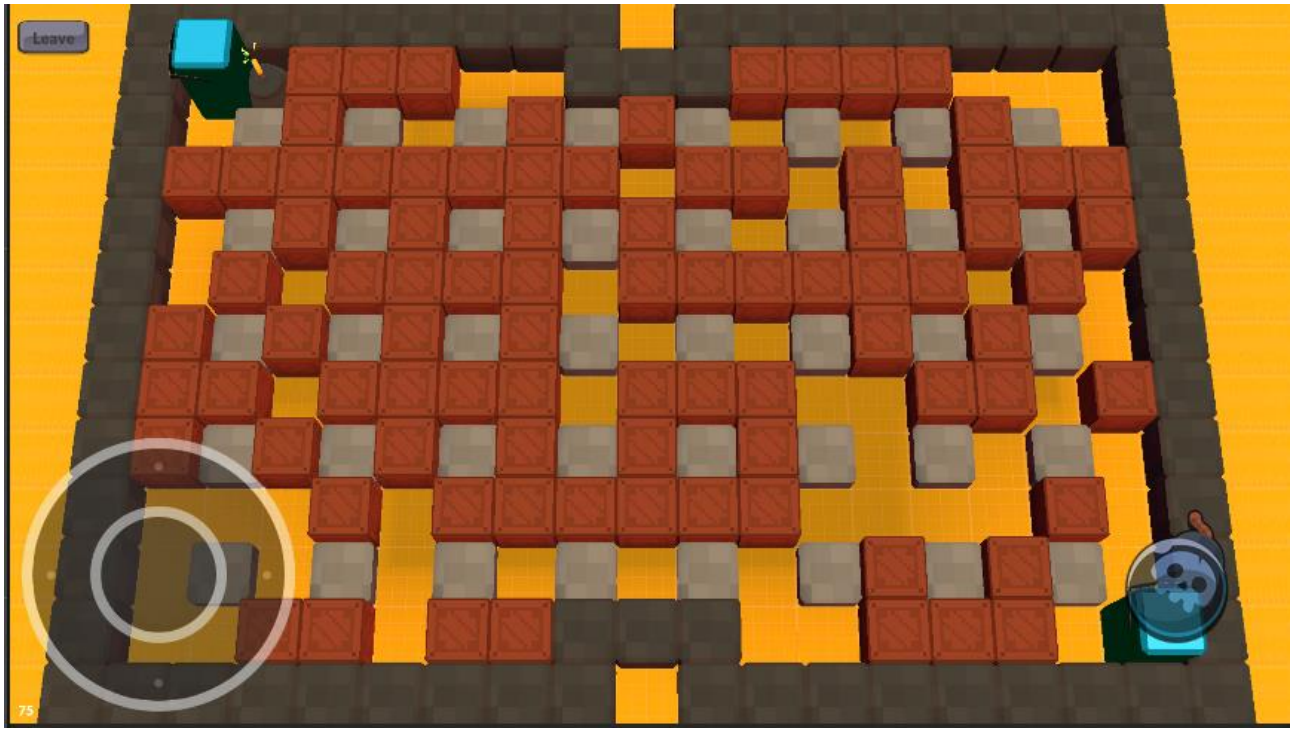

Рис.2.27. Ігрова сцена на Android

Джойстик та кнопка спауна бомб на Windows відсутні.

Інтерфейс в Unity створюється за допомогою елементу Canvas.

Canvas — це область, всередині якої мають бути всі елементи інтерфейсу користувача [23]. Canvas — це ігровий об’єкт із компонентом Canvas, і всі елементи інтерфейсу мають бути дочірніми для так. Створення нового елемента інтерфейсу користувача, наприклад, зображення за допомогою меню GameObject > UI > Image, автоматично створює Canvas, якщо в сцені ще немає Canvas. Елемент інтерфейсу користувача створюється як дочірній для цього Canvas. Область Canvas відображається у вигляді прямокутника в режимі перегляду сцени. Це дозволяє легко розташовувати елементи інтерфейсу користувача без необхідності постійно відображати Game View. Canvas використовує об’єкт EventSystem, щоб допомогти системі обміну повідомленнями — це область, всередині якої мають бути всі елементи інтерфейсу користувача.

Елементи інтерфейсу користувача на Canvas малюються в тому ж порядку, в якому вони з’являються в ієрархії Рис.2.28.

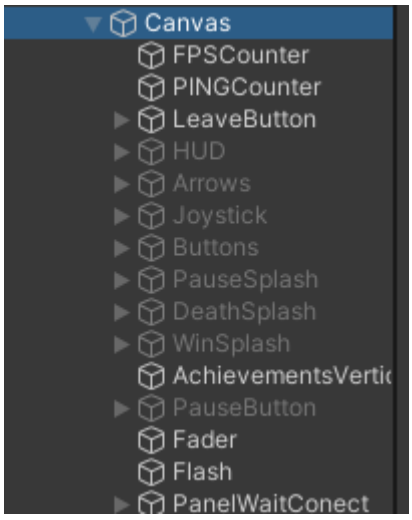

Рис.2.28. Приклад використання Canvas

Першою малюється перша дитина, наступною – друга і так далі. Якщо два елементи інтерфейсу перекриваються, останній з’явиться поверх попереднього. Щоб змінити, який елемент відображатиметься поверх інших елементів, просто змініть порядок елементів в ієрархії, перетягнувши їх.

Адаптивний UI в Unity завдяки Anchored Position [25]. Anchored Position — це положення опорної точки RectTransform з урахуванням опорної точки

прив’язки. Орієнтирною точкою якоря є положення якорів. Якщо якоря не розташовані разом, Unity оцінює чотири позиції прив’язки, використовуючи розташування опори як довідник.

РОЗДІЛ 3
ЕКОНОМІЧНІЙ РОЗДІЛ

3.1. Розрахунок трудомісткості та вартості розробки програмного продукту

Початкові дані:

1. передбачуване число операторів програми – 2566;
2. коефіцієнт складності програми – 1,5;
3. коефіцієнт корекції програми в ході її розробки – 0,09;
4. годинна заробітна плата програміста – 160 грн/год;

Годинна заробітна плата Unity 3D розробника була вирахувана виходячи з даних «Української спільноти програмістів (DOU)». Середньоукраїнська заробітна плата такого розробника з досвідом роботи близько року дорівнює 800 доларів США у місяць [34]. При курсі валют НБУ на початок червня 2022 року один американський долар дорівнює 29,25 грн, тому середня зарплата в гривнях дорівнює 23400 грн. При восьмигодинному робочому дні (176 робочих годин в місяць в середньому) середня заробітна плата за годину буде становити 160 грн.

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. Кількість виконавців – 1
8. Вартість машино-години ЕОМ – 10 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:

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

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):

$$Q = q \cdot C \cdot (1 + p), \tag{3.2}$$

де q – передбачуване число операторів (2566);

C – коефіцієнт складності програми (1,5);

p – коефіцієнт кореляції програми в ході її розробки (0,09).

$Q = 2566 \cdot 1{,}3 \cdot (1 + 0{,}09) = 3636;$

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

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

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

K – коефіцієнт кваліфікації програміста, обумовлений стажем роботи з даної спеціальності (0,9);

$t_u = \frac{2566 \cdot 1{,}2}{85 \cdot 0{,}9} = 40{,}3$, людино-годин.

Витрати праці на розробку алгоритму рішення задачі:

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

$t_a = \frac{2566}{20 \cdot 0{,}9} = 142{,}5$, *людино-годин.*

Витрати на складання програми по готовій блок-схемі:

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

$t_n = \frac{2566}{25 \cdot 0{,}9} = 114$, *людино-годин.*

Витрати праці на налагодження програми на ЕОМ:

за умови автономного налагодження одного завдання:

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

$t_{\text{отл}} = \frac{2566}{5 \cdot 0.9} = 570{,}2$, *людино-годин,*

за умови комплексного налагодження завдання:

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

$t_{\text{отл}}^{\text{к}} = 1{,}2 \cdot 570{,}2 = 684{,}26$, *людино-годин,*

Витрати праці на підготовку документації:

$$t_д = t_{др} + t_{до}; \quad (3.8)$$

де $t_{др}$ – трудомісткість підготовки матеріалів і рукопису;

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

$t_{\partial} = \frac{2566}{20 \cdot 0{,}9} = 142{,}5$, *людино-годин,*

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

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

$t_{до} = 0{,}75 \cdot 142{,}5 = 106{,}91$, *людино-годин.*

$t_{\partial} = 142{,}5 + 106.91 = 249{,}41$, *людино-годин.*

Отримаємо трудомісткість розробки програмного забезпечення:

t = 50 + 40,3+ 142,5 + *114 + 570,2 +249,41 = 1166,11, людино-годин.*

У результаті ми розрахували, що в загальній складності необхідно 1166,11 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.

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

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

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

де *t* – загальна трудомісткість, людино-годин;

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

З урахуванням того, що середня годинна зарплата програміста становить 150 грн/год, то отримаємо:

$З_{ЗП} = 1166{,}11 \cdot 150 = 170916{,}5$, 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.

$З_{МВ} = 570{,}2 \cdot 10 = 5702$ грн.

Звідси витрати на створення програмного продукту:

$К_{ПО} = 170916{,}5 + 5702 = 176618{,}5$ 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 программного продукту:

$T = \frac{1166{,}11}{1 \cdot 176} = 6{,}6$ *міс.*

Висновки. Додаток має вартість 176618,5 грн. Ймовірний очікуваний час розробки – 6,6 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 додатку буде витрачено 1166,11 людино-годин.

ВИСНОВКИ

Під час виконання кваліфікаційної роботи був розроблений практичний та надійний застосунок, використовуючи сучасні підходи та інструменти для кросплатформеної гри на Unity.

Перевагами використання сучасних підходів до створення програми є подальша легкість впровадження, за необхідності, додаткових функцій, за рахунок модульної архітектури.

Додаток має сучасні технології, зрозумілий та зручний інтерфейс, швидкий запуск та підключення. Сучасна графіка та приємне управління в додатку є одним із найбільших плюсів. Photon в свою чергу за рахунок своєї архітектури надає стабільний пінг та гарне підключення.

Застосунок має два режими роботи, онлайн та офлайн за рахунок бота який, підключається при відсутності суперника онлайн.

В результаті отримано застосунок, який має практичну цінність, тому, що в грі можна гарно провести час зі своїми друзями. Подалі застосунок легко можна монетизувати через рекламу, введення кастомізації та підсилювачив які, будуть змінювати ігрову логіку.

Через те, що додаток працює на платформах Windows та Android існує можливість того, що додаток буде популярнішим серед цільової аудиторії. Додаток розроблений на мові C#, яка зарекомендувала себе вже давно. За рахунок того, що додаток створенно на Unity, подалі розширення платформ, на які, можна буде встановити застосунок може бути розширено.

Під час виконання даного дипломного проекту були виконані наступні етапи створення програмного продукту:

- аналіз предметної області задачі, що розв’язується;
- обрання раціональної архітектури та технології створення додатку;
- написання програмного коду додатку;

Під час виконання кваліфікаційної роботи також було визначено трудомісткість розробленого програмного продукту (1166,11 людино-годин),

проведений підрахунок вартості роботи по створенню програми (176618,5 грн) та розраховано час на його створення (6.6 міс).

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Unity Game Development Cookbook: Essentials for Every Game 1st Edition Автори: Джон Меннінг Тім, Ньюджент Періс, Батфілд Еддісон
2. Unity in Action: Multiplatform Game Development in C# with Unity 5 Автори: Джо Хокінг Джессі Шелл
3. Introduction to Game Design, Prototyping, and Development Автор: Джеремі Гібсон Бонд
4. Nice Touch documenation https://nice-touch-docs.moremountains.com/API/
5. Nice Vibration documentation https://nice-vibrations-docs.moremountains.com/#does-it-work-everywhere-
6. Multiplayer and Networking documentation documentation https://docs.unity3d.com/Manual/UNet.html
7. Atributs in Unity documentation https://docs.unity3d.com/ScriptReference/TooltipAttribute.html
8. Microsft quickstart in Unity documentation https://docs.microsoft.com/en-us/visualstudio/gamedev/unity/get-started/getting-started-with-visual-studio-tools-for-unity?view=vs-2022&pivots=windows
9. C# in Unity documentation https://docs.microsoft.com/en-us/archive/msdn-magazine/2014/august/unity-developing-your-first-game-with-unity-and-csharp
10. Microsft quickstart in Photon documentation https://docs.microsoft.com/en-us/gaming/playfab/sdks/photon/quickstart
11. Photon introduction documentation https://doc.photonengine.com/en-us/pun/current/getting-started/pun-intro
12. Photon Features overview documentation https://doc.photonengine.com/en-us/pun/current/getting-started/feature-overview
13. Photon setup and connect documentation https://doc.photonengine.com/en-us/pun/current/getting-started/initial-setup
14. Photon C# callbacks documentation https://doc.photonengine.com/en-us/pun/current/getting-started/dotnet-callbacks
15. Photon migration node documentation https://doc.photonengine.com/en-us/pun/current/getting-started/migration-notes
16. Photon app and lobby stats documentation https://doc.photonengine.com/en-us/pun/current/lobby-and-matchmaking/appandlobbystats
17. Photon regions documentation https://doc.photonengine.com/en-us/pun/current/connection-and-authentication/regions
18. Photon matchmakibg guide https://doc.photonengine.com/en-us/pun/current/lobby-and-matchmaking/matchmaking-and-lobby
19. Quickstart to TextMesh Pro https://learn.unity.com/tutorial/working-with-textmesh-

pro#:~:text=TextMesh%20Pro%20is%20an%20easy,to%20any%20project%27s%20user%20interface.
20. TextMesh Pro settings https://docs.lunalabs.io/docs/playable/code/unity-plugins/tmp
21. Unity Editor guide https://learn.unity.com/tutorial/get-started-in-the-unity-editor
22. Create user UI in Unity documentation https://docs.unity3d.com/Manual/UIToolkits.html
23. Canvas in Unity documentation https://docs.unity3d.com/Packages/com.unity.ugui@1.0/manual/UICanvas.html
24. Rect Transform in Unity documentation https://docs.unity3d.com/Packages/com.unity.ugui@1.0/manual/class-RectTransform.html
25. Animation in Unity documentation https://docs.unity3d.com/Manual/AnimationSection.html
26. Unity quick search https://docs.unity3d.com/Packages/com.unity.quicksearch@1.0/manual/index.html
27. Tag component https://gametorrahod.com/tag-component/
28. Game object coversioin https://gametorrahod.com/gameobject-conversion-ecosystem-code-tour/
29. TextMesh Pro anatomy https://gametorrahod.com/textmeshpro-anatomy/
30. Unity Event serialization https://gametorrahod.com/unityevent-serialization-research/
31. Quality in Unity documentation https://docs.unity3d.com/Manual/class-QualitySettings.html
32. Integrated development environment (IDE) support https://docs.unity3d.com/2020.2/Documentation/Manual/ScriptingToolsIDEs.html
33. Draw call batching documentation https://docs.unity3d.com/2019.4/Documentation/Manual/DrawCallBatching.html
34. DOU https://dou.ua/?utm_source=google&utm_medium=cpc&utm_campaign=search-brand&gclid=CjwKCAjwqauVBhBGEiwAXOepkT0MwxHh446-SO3crkQwq9LdD0FqNhAFx4IzbssoJV-a_ldFX1ThRBoCJyYQAvD_BwE
35. Metanit https://metanit.com/sharp/tutorial/3.7.php

ДОДАТОК А

КОД ПРОГРАМИ

```
using System.Collections.Generic;
using AnimationEvent;
using Extential;
using Photon.Pun;
using Photon.Realtime;
using TMPro;
using UnityEngine;
using UnityEngine.EventSystems;
using UnityEngine.UI;

namespace Lobby
{
   public class LobbyController : MonoBehaviourPunCallbacks
   {
      [SerializeField]
      private TMP_InputField _tmpInputField;

      [SerializeField]
      private Button m_Battle;

      [SerializeField]
      private Button m_CreateRoom;

      [SerializeField]
      private Button m_JoinRandomRoom;

      [SerializeField]
      private EventSystem m_EventSystem;

      [SerializeField]
      private WaitPanelAnim m_LoadingPanel;

      [SerializeField]
      private GameObject m_LobbyPanel;

      [SerializeField]
      private RoomPanel m_RoomsPanelPrefab;
```

```
[SerializeField]
private Transform m_Content;

private List<RoomInfo> m_Rooms = new List<RoomInfo>();

private const string GameScene = "GameScene";

private void Start()
{
    m_EventSystem.enabled = false;
    m_LoadingPanel.gameObject.SetActive(true);
    m_LobbyPanel.gameObject.SetActive(false);
    PhotonNetworkSetup();
    SubscribeButton();
}

private void SubscribeButton()
{
    LoadWaitPanel();
    m_CreateRoom.onClick.AddListener(CreateRoom);
    m_JoinRandomRoom.onClick.AddListener(JoinGame);
    m_Battle.onClick.AddListener(JoinGame);
}

private void LoadWaitPanel()
{
    m_EventSystem.enabled = false;
    m_LoadingPanel.SetUp();
    m_LoadingPanel.gameObject.SetActive(true);
    m_LobbyPanel.gameObject.SetActive(false);
}

private void UpdateRooms()
{
    m_Content.transform.Clear();

    foreach (var roomInfo in m_Rooms)
    {
        var prefab = Instantiate(m_RoomsPanelPrefab, m_Content);
        prefab.SetUp(roomInfo, this);
    }
```

```
    if (m_Rooms.Count != 0)
    {
        return;
    }

    {
        var prefab = Instantiate(m_RoomsPanelPrefab, m_Content);
        prefab.SetUpInfo(LobbyConstants.NoRoom);
    }
}

private void AnimComplete()
{
    m_LoadingPanel.StartAnim();
    m_EventSystem.enabled = true;
}

private void CreateRoom()
{
    LoadWaitPanel();
    string nameRoom = _tmpInputField.text.Length == 0 ? _tmpInputField.text : PhotonNetwork.NickName;

    PhotonNetwork.CreateRoom(nameRoom, new RoomOptions
    {
        IsVisible = true,
        MaxPlayers = 2,
    });
}

private void JoinGame()
{
    LoadWaitPanel();
    PhotonNetwork.JoinRandomRoom();
}

private static void PhotonNetworkSetup()
{
    PhotonNetwork.NickName = "Player" + Random.Range(1, 500);
    PhotonNetwork.AutomaticallySyncScene = true;
    PhotonNetwork.GameVersion = "1.0.0";
    PhotonNetwork.ConnectUsingSettings();
}
```

```
        public override void OnRoomListUpdate(List<RoomInfo> roomList)
        {
            m_Rooms = roomList;
            UpdateRooms();
        }

        public void JoinRoom(string nameRoom)
        {
            PhotonNetwork.JoinRoom(nameRoom);
        }

        public override void OnConnectedToMaster()
        {
            m_LobbyPanel.SetActive(true);
            AnimComplete();
            PhotonNetwork.JoinLobby();
        }

        public override void OnJoinedRoom()
        {
            PhotonNetwork.LoadLevel(GameScene);
        }

        public override void OnJoinRandomFailed(short returnCode, string message)
        {
            CreateRoom();
        }
    }
}
using Photon.Realtime;
using TMPro;
using UnityEngine;
using UnityEngine.UI;

namespace Lobby
{
    public class RoomPanel : MonoBehaviour
    {
        [SerializeField]
        private Button m_Button;
```

```
        [SerializeField]
        private TMP_Text m_RoomName;

        private LobbyController m_LobbyController;
        private string m_NameRoom;

        public void SetUp(RoomInfo roomInfo, LobbyController lobbyController)
        {
            m_NameRoom = roomInfo.Name;
            m_LobbyController = lobbyController;

            m_RoomName.SetText(m_NameRoom);
            m_Button.onClick.AddListener(JoinRoom);
        }

        public void SetUpInfo(string text)
        {
            m_RoomName.SetText(text);
        }

        private void JoinRoom()
        {
            m_LobbyController.JoinRoom(m_NameRoom);
        }
    }
}
using System.Collections;
using ExitGames.Client.Photon;
using MoreMountains.TopDownEngine;
using Photon.Pun;
using Photon.Realtime;
using UnityEngine;

namespace GameManager
{
    public class MapController : MonoBehaviour, IOnEventCallback
    {
        [SerializeField]
        private ExplodudesCrate[] m_ExplodudesCrates;

        private void Awake()
        {
```

```
    PhotonPeer.RegisterType(typeof(MapDate), 244, MapDate.Serialize, MapDate.DeSerialize);
}

private void OnEnable()
{
    PhotonNetwork.NetworkingClient.EventReceived += OnEvent;
}

private void OnDisable()
{
    PhotonNetwork.NetworkingClient.EventReceived -= OnEvent;
}

private void Start()
{
    if (PhotonNetwork.IsMasterClient)
        SetUpMap();
}

public void SendSyncDate(Player player)
{
    MapDate date = new MapDate();

    RaiseEventOptions options = new RaiseEventOptions
    {
        TargetActors = new[] { player.ActorNumber }
    };

    SendOptions sendOptions = new SendOptions
    {
        Reliability = true
    };

    date.CratesDate = new BitArray(m_ExplodudesCrates.Length);

    for (int i = 0; i < m_ExplodudesCrates.Length; i++)
    {
        date.CratesDate.Set(i, m_ExplodudesCrates[i].IsEnable);
    }

    PhotonNetwork.RaiseEvent(43, date, options, sendOptions);
}
```

```
    public void OnEvent(EventData eventData)
    {
        switch (eventData.Code)
        {
            case 43:
                var date = (MapDate)eventData.CustomData;
                OnSendDateRecursive(date);
                break;
        }
    }

    private async void OnSendDateRecursive(MapDate eventData)
    {
        for (int i = 0; i < m_ExplodudesCrates.Length; i++)
        {
            var active = eventData.CratesDate.Get(i);
            m_ExplodudesCrates[i].SetActive(active);
        }
    }

    private void SetUpMap()
    {
        for (int i = 0; i < m_ExplodudesCrates.Length; i++)
        {
            var random = Random.Range(0, 3);

            if (random == 0)
            {
                m_ExplodudesCrates[i].SetActive(false);
            }
        }
    }
}
using System.Collections.Generic;
using Photon.Pun;
using Photon.Realtime;
using UnityEngine;
using UnityEngine.SceneManagement;
using UnityEngine.UI;
```

```
namespace GameManager
{
    public class PhotoGameManager : MonoBehaviourPunCallbacks
    {
        [SerializeField]
        private Button m_LeaveRoomButton;

        [SerializeField]
        private List<Transform> m_ListSpawnPosition;

        [SerializeField]
        private GameObject m_LoadingPanel;

        [SerializeField]
        private MapController m_MapController;

        private List<Player> m_Characters = new List<Player>();
        private object invoke;
        private GameObject player;
        private AiBomber bot;

        private void Start()
        {
            m_LeaveRoomButton.onClick.AddListener(OnLeftRoom);

            SpawnPlayers();
        }

        private void SpawnBot()
        {
            m_LoadingPanel.SetActive(false);

            bot = PhotonNetwork.Instantiate("AiMinimalGridCharacter", m_ListSpawnPosition[1].position,
Quaternion.identity).GetComponent<AiBomber>();
            bot.Target = player;
        }

        private void SpawnPlayers()
        {
            if (PhotonNetwork.CurrentRoom.PlayerCount == 1)
            {
```

```
                player = PhotonNetwork.Instantiate("MinimalGridCharacter", m_ListSpawnPosition[0].position,
Quaternion.identity);
                Invoke(nameof(SpawnBot), 6f);
                m_LoadingPanel.SetActive(true);
            }
            else
            {
                PhotonNetwork.Instantiate("MinimalGridCharacterSecond", m_ListSpawnPosition[1].position,
Quaternion.identity);
                m_LoadingPanel.SetActive(false);
                CancelInvoke(nameof(SpawnBot));

                if (bot != null)
                {
                    Destroy(bot.gameObject);
                }
            }
        }

        public override void OnPlayerEnteredRoom(Player newPlayer)
        {
            AddPlayer(newPlayer);

            if (PhotonNetwork.IsMasterClient)
            {
                m_MapController.SendSyncDate(newPlayer);
            }

            if (PhotonNetwork.CurrentRoom.PlayerCount == 2)
            {
                m_LoadingPanel.SetActive(false);
                CancelInvoke(nameof(SpawnBot));

                if (bot != null)
                {
                    Destroy(bot.gameObject);
                }
            }
        }

        public override void OnPlayerLeftRoom(Player player)
        {
```

```
            Debug.LogFormat("Player{0} entered room", player.NickName);
        }

        public override void OnLeftRoom()
        {
            SceneManager.LoadScene(0);
        }

        private void AddPlayer(Player character)
        {
            m_Characters.Add(character);
        }
    }
}
using Photon.Pun;
using UnityEngine;
using UnityEngine.SceneManagement;
using UnityEngine.UI;

namespace Lobby
{
    public class LeaveRoomButton : MonoBehaviour
    {
        private Button _button;

        private void Start()
        {
            _button = GetComponent<Button>();

            if (_button == null)
            {
                return;
            }

            _button.onClick.AddListener(LeaveRoomButtonClick);
        }

        private void Leave()
        {
            PhotonNetwork.LeaveRoom();
        }
```

```
    public void LeaveRoomButtonClick()
    {
        SceneManager.LoadScene(0);
        Leave();
    }
  }
}
```

ДОДАТОК Б

ВІДГУК КЕРІВНИКА ЕКОНОМІЧНОГО РОЗДІЛУ

ДОДАТОК В

Перелік файлів на магнітному носії

Ім’я файлу	Опис
Пояснювальні документи	
Кваліфікаційна_робота_Савостяненко.docx	Пояснювальна записка до кваліфікаційної роботи. Документ Word.
Кваліфікаційна_робота_Савостяненко.pdf	Пояснювальна записка до кваліфікаційної роботи в форматі PDF
Програма	
Савостяненеко.rar	Архів. Містить коди програми і скомпільовану програму під Windows
Савостяненко.apk	Скомпільована програма під Android
Презентація	
Савостяненко.ppt	Презентація кваліфікаційної роботи

ВІДГУК

на кваліфікаційну роботу бакалавра
на тему:
"Розробка кросплатформеної
мережевиї гри на Unity з використанням Photon"
студента групи 121-18-1 Савостяненко Володимир Івонович

Розроблена в кваліфікаційній роботі програма призначена для розповсюдження гри з метою зацікавлення нею аудиторії.

Як інструмент для проектування і реалізації було використане середовище розробки Unity за допомогою мови програмування C#.

Практична значимість створення даної інформаційної системи полягає в можливості грати в гру на різних платформах та подальшому зацікавленні аудиторії.

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 бала «добре», а студент Савостяненко Володимир Іванович 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:
"Розробка кросплатформеної
мереживої гри на Unity з використанням Photon "
студента групи 121-18-1 Савостяненко Володимира Іванович

Кваліфікаційну роботу на тему "Розробка кроссплатформеної

мереживої гри на Unity з використанням Photon" виконано в повному обсязі, відповідно до технічного завдання.

Мета кваліфікаційної роботи: метою дипломного проекту є розробка 3D гри за допомогою Unity системи у середовищі Jetbrains Rider .

У пояснювальній записці розглянуто необхідність створення і сфера застосування розробленої, 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 Савостяненка Володимира Івановича за напрямом 121 Інженерія програмного забезпечення.

Список літератури, наведений в роботі, налічує 35 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 «відмінно», а студент Савостяненко Володимир Іванович заслуговує присвоєння йому кваліфікації бакалавра з інженерії програмного забезпечення.

Рецензент кваліфікаційної роботи