МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра комп'ютерних наук

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

на тему: **"РОЗРОБКА АРКАДНОЇ ГРИ BRICK BREAKER З ВИКОРИСТАННЯМ ФРЕЙМВОРКУ UNITY"**

Виконав: студент 4 курсу, групи 6.1229
спеціальності 122 комп'ютерні науки
(шифр і назва спеціальності)
освітньої програми комп'ютерні науки
(назва освітньої програми)
М. С. Єфімов
(ініціали та прізвище)
Керівник доцент кафедри комп'ютерних наук, к.т.н., доцент Борю С. Ю.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)
Рецензент завідувач кафедрою програмної інженерії, доцент, к.ф.-м.н. Лісняк А. О.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя – 2023

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра комп'ютерних наук

Рівень вищої освіти бакалавр

Спеціальність 122 комп'ютерні науки
(шифр і назва)

Освітня програма комп'ютерні науки

ЗАТВЕРДЖУЮ
Завідувач кафедри комп'ютерних наук,
д.т.н., професор

_____ Чопоров С.В.
(підпис)

" ____ " _____ 2023 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Єфімову Микиті Сергійовичу
(прізвище, ім'я та по-батькові)

1. Тема роботи Розробка аркадної гри Brick Breaker з використанням фреймворку Unity

керівник роботи Борю С. Ю., к.т.н., доцент
(прізвище, ім'я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від " 26 " січня 2023року № 102-с

2. Строк подання студентом роботи 6.06.2023

3. Вихідні дані до роботи 1. Постановка задачі.
2. Перелік літератури.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Постановка задачі.
2. Основні теоретичні відомості.
3. Розробка аркадної гри brick breacker

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
презентація

6. Консультанти розділів роботи

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

7. Дата видачі завдання 22.03.2023

К А Л Е Н Д А Р Н И Й П Л А Н

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Розробка плану роботи.	3.04.2023	
2.	Збір вихідних даних.	7.04.2023	
3.	Обробка методичних та теоретичних	16.04.2023	
	джерел.		
4.	Розробка першого та другого розділу.	20.04.2023	
5.	Розробка третього розділу.	10.05.2023	
6.	Оформлення та нормоконтроль	6.06.2023	
	кваліфікаційної роботи бакалавра.		
7.	Захист кваліфікаційної роботи.	20.06.2023	

Студент _____ М.С. Єфімов
(підпис) (ініціали та прізвище)

Керівник роботи _____ С.Ю. Борю
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

Нормоконтролер _____ О.Г. Спиця
(підпис) (ініціали та прізвище)

РЕФЕРАТ

Кваліфікаційна робота бакалавра “Розробка аркадної гри Brick Breaker з використанням фреймворку Unity”: 73 с., 25 рис., 0 табл.,14 джерел, 19 додатків.

АРКАДНА ГРА, ГРАФІЧНИЙ ІНТЕРФЕЙС, ІГРОВИЙ РУШІЙ, КЛАС, ПРОГРАМУВАННЯ, РОЗРОБКА ГРИ, УПРАВЛІННЯ ГРОЮ, СПРАЙТ.

Об’єкт дослідження – досліджування процесу розробки гри, включаючи керування проектом, створення дизайну ігрового світу, звукового оформлення, взаємодії з гравцями.

Мета роботи: Створити гру за допомогою рушія Unity. Основні завдання роботи включають вибір технологій розробки, створення архітектури гри, реалізацію ігрових механік, включаючи управління ракеткою, фізику кулі, реалізацію бонусів та послаблень, підстановкою звукових ефектів для кожної події у грі.

Методи дослідження: Аналіз літературних джерел, розробка прототипу, опитування та збір даних, порівняльний аналіз.

SUMMARY

Bachelor’s qualifying theses "Development of the arcade game Brick Breaker with the virtual framework Unity": 73 p., fig.,25 table, 14 references, 19 supplements.

ARCADE GAME, GUI, GAME ENGINE, CLASSROOM, PROGRAMMING, GAME DEVELOPMENT, GAME CONTROL, SPRITE.

Research Object: The research focuses on the process of game development, including project management, game world design, sound design, and player interaction.

Research Goal: Creating a game using the Unity game engine. The main tasks of the job include selecting development technologies, designing the game architecture, implementing gameplay mechanics such as paddle control, bullet physics, implementing bonuses and power-ups, and integrating sound effects for each action in the game.

Research Methods: Analysis of literature sources, Prototype development, Surveys and data collection, Comparative analysis.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

SDK – software development kit

API – Application Programming Interface

ПЗ – Програмне забезпечення

PS – Playstation

VR – virtual reality

AR – augmented reality

GE – Game engine

IDE – Integrated Development Environment

ВСТУП

За останні два десятиліття у світі з'явився і дуже стрімко розвинувся новий напрям – комп’ютерні ігри. Темп розвитку пояснюється просто: стрімкий розвиток комп'ютерних технологій та інтернету на кінці 20 століття. Через це вони стали одним з найдоступнішими та найпопулярнішими видами розваг у сьогоденні. Тому навіть індустрія виробників комлектуючих до комп'ютера зараз орієнтується на ігровий ринок, для прикладу компанія розробників графічних процесорів NVIDIA, тому що з кожним роком якість та вимогу до складових комп’ютера ростуть з кожним поколінням ігор. Тому можна казати з повною упевненістю, що комп’ютерні ігри напряму впливають на розвиток технологій.

Одним з ігрових рушіїв, та фреймворком для розробки ігор є Unity. На даний момент він є найпопулярнішим ігровим рушієм для розробки ігор, але вже в найближчому майбутньому його замістить більш складний але більш різноманітний Unreal Engine. Хоча на даний момент усе одно Unity є більш привабливим для розробників початківців через свою простоту.

Жанр акрадних ігор у сьогоденні є найпопулярнішим жанром з усіх через свою простоту. Їх головною метою є витрачення часу з задоволенням, хоча також більшість таких ігор можуть піти на користь для логічного розвитку, покрашення дрібної моторики рук, чи покращення короткострокової пам’яті. .[13]

На даний момент більшість аркадних ігор розробленні за допомогою Unity.[14]

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Поняття аркадної гри

Аркадні відеоігри приймають вхідні дані гравця від своїх органів управління, обробляють їх за допомогою електричних або комп'ютеризованих компонентів та відображають вихідні дані на електронному моніторі або аналогічному дисплеї. Всі аркадні відеоігри оплачуються монетами або приймають інші платіжні кошти, розміщуються в аркадних автоматах та розміщуються в ігрових автоматах поряд з іншими видами аркадних ігор. До початку 2000-х років аркадні відеоігри були найбільшим і найбільш технологічним сегментом індустрії відеоігор.[13]

Перші прототипи ігор Galaxy Game та Computer Space у 1971 році встановили принципи роботи аркадних ігор, а Pong від Atari у 1972 році була визнана першою успішною комерційною аркадною відеогрою. Поліпшення в комп'ютерних технологіях та дизайні ігрового процесу призвели до золотого віку аркадних відеоігор, точні дати яких обговорюються, але варіюються від кінця 1970-х до середини 1980-х років. Цей золотий вік включає Space Invaders, Pac-Man і Donkey Kong. Індустрія ігрових автоматів відродилася з початку 1990-х до середини 2000-х, включаючи Street Fighter II, Mortal Kombat і Dance Dance Revolution.

Багато аркадних ігор мають короткі рівні, прості та інтуїтивно зрозумілі схеми управління та швидко зростаючу складність. Класична формула успішної аркадної відеоігри: «легко навчитися, важко освоїти”» разом з парадигмою «декілька життів, рівень що поступово ускладнюється». Це пов'язано з середовищем грального автоматів, де гравець, по суті, орендує гру доти, доки його ігровий аватар може залишатися живим. Ігри на консолях або ПК можна назвати аркадними іграми, якщо вони мають ці якості або є прямими портами аркадних ігор. Хоча зараз використовування виразу «Аркадна гра» кардинально змінилось, зараз даний вираз використовується

для менш реалістичної гри, в якій дуже просто витратити свій час. Наприклад зараз ніхто не буде називати таку гру як «Mario Cart» гоночним симулятором на відміну від «Forza». Також під аркадними іграми можуть мати на увазі не складні головоломки чи платформери по типу: 2048, три в ряд, Super Mario, Sonic і т.п. Отже на сьогодні можна зробити висновок, що аркадні ігри – це жанр відеоігор, який включає в себе прості, легкі та швидкі геймплейні механіки, звичайно, з орієнтацією на найширший загал гравців. Ці ігри зазвичай характеризуються швидкістю, рефлексами та легкістю в освоєнні, що робить їх дуже привабливими для гравців.

Аркадні гоночні ігри часто мають складні для аркадних автоматів симулятори руху, спрощений фізичний рушій і короткий час навчання порівняно з більш реалістичними симуляторами гоночними. Автомобілі можуть різко повертати без гальмування або недостатньої повертаємості, а ІІ-суперники іноді запрограмовані таким чином, щоб вони завжди знаходилися поруч із гравцем з ефектом гумової стрічки(rubber band, Dynamic game difficulty balancing). Інші типи аркадних ігор включають музичні ігри (зокрема, ритм-ігри), а також мобільні та казуальні ігри з інтуїтивно зрозумілим керуванням та короткими сеансами.

1.2Аналіз жанрів аркадних ігор

Жанр гри – це категорія або тип ігрового досвіду, який визначається характерними елементами геймплею, настроєм, тематикою та механіками гри. Жанр гри визначає, яким чином гравець взаємодіє з грою, які цілі маються перед ним та які виклики він повинен подолати.

Аркадні ігри поділяються на такі жанри:

– Платформери: Це один з найпоширеніших видів аркадних ігор, в яких гравець керує головним героєм, який пересувається по платформам, стрибаючи, біжачи та подолаючи перешкоди.

Приклади таких ігор включають Super Mario Bros та Sonic the Hedgehog;

- Шутер: Ці ігри зосереджені на швидких реакціях та точному прицілюванні. Гравець керує персонажем або транспортним засобом, щоб вистрілювати ворогів та уникати їхніх атак. Приклади таких ігор включають Space Invaders та Galaga;
- Рейсери: Ці ігри зосереджені на швидкості та вмінні керувати транспортним засобом. Гравець бере участь у гонках, змагається з іншими гравцями або просто намагається досягти фінішу швидше за інших. Приклади таких ігор включають Need for Speed та Mario Kart;
- Brick breakeги: Ці ігри базуються на класичній грі Arkanoid, де гравець керує платформою, щоб відбивати кулю та розбивати блоки. Метою гри є видалення всіх блоків, збиваючи їх кулею без втрати останнього життя;
- Ігри-головоломки: Ці ігри пропонують гравцям різні головоломки та завдання, які потребують логічного мислення та вирішення проблем. Приклади таких ігор включають Tetris та Bejeweled;
- Бойовики: Ці ігри зосереджені на боротьбі та бійцях. Гравець керує героєм, який здатний до виконання різноманітних атак та прийомів. Приклади таких ігор включають Street Fighter та Mortal Kombat.

1.3Аналіз ринку аркадних ігор

Ще до 2000-х років дуже частим явищем було конвертувати або емулювати популярні ігри з аркадних автоматів на домашні чи портативні консолі. Більшість перших ігор на ROM-картриджі для консолі Atari 2600 були імпортовані з аркадних автоматів від самої ж Atari. Розробники аркадних ігор, які не займались виробництвом домашніх консолей, визнали ліцензування своїх ігор виробникам консолей успішною бізнес-моделлю, оскільки

конкуренти-виробники консолей боротимуться за права на більш популярні ігри.

У 1994 з’явилась перша аркадна гра на телефон – тетріс.[15] У 1997 році вже Nokia презентувала свою аркадну гру Змійка. З її виходом вона була інстальована на білішості телефонів Nokia. І з тих пір вона стала одною з найвпізнаваніших ігор у світі.

Навіть на сьогодення акрадна гра тетріс є найпродаванішою грою з більш ніж 520 мільйонами копій.

На сьогоднішній день ринок аркадних ігор продовжує залишатися живим та популярним серед гравців у всьому світі. Аркадні ігри приваблюють широкий спектр аудиторії завдяки своїй простоті, швидкості та веселому геймплею. Давайте зробимо аналіз ринку аркадних ігор на данний момент.

Мобільні ігри: Мобільні аркадні ігри стали одним з найпопулярніших сегментів геймінгу. Завдяки поширенню смартфонів та планшетів, гравці мають можливість грати в аркади на ходу. Цей сегмент ринку пропонує безліч безкоштовних аркадних ігор з простим управлінням та короткими сеансами гри. Деякі з найуспішніших мобільних аркадних ігор включають "Angry Birds", "Candy Crush Saga" та "Subway Surfers".

Ретро-стиль: Ретро-стиль аркадних ігор завжди мав свою особливу атмосферу та прихильників. Відтворення класичних аркадних ігор або створення нових проектів в ретро-стилі зі старомодною графікою та звуком привертає увагу гравців, які ностальгують за іграми минулого. Такі ігри, як "Pac-Man" та "Tetris", досі мають широку популярність серед гравців будь-якого віку.

Онлайн та мультиплеєр: Розвиток онлайн-ігор впливає і на ринок аркадних ігор. Гравці все частіше шукають можливостей грати в аркади разом з друзями або змагатися з іншими гравцями у режимі реального часу. Онлайн-елементи, такі як командні змагання, лідерські дошки та спільне проходження гри, надають ігровому процесу більшої соціальної взаємодії та забезпечують тривалість геймплею.

Використання VR та AR: Розширена реальність (AR) та віртуальна реальність (VR) відкривають нові можливості для аркадних ігор. Гравці можуть погрузнути в ігровий світ та зануритися у неповторний досвід завдяки іммерсивним технологіям. Ігри, які використовують VR або AR елементи, надають новий рівень взаємодії та створюють захоплюючий геймплей для гравців.

Елементи розвитку персонажа: Деякі сучасні аркадні ігри поєднують в собі елементи розвитку персонажа. Гравці можуть вдосконалювати своїх героїв, отримувати нові навички та виконувати завдання для отримання нагород. Це надає більше глибини ігровому процесу та стимулює гравців до подальшого прогресування.

Оглядаючи ринок аркадних ігор на данний момент, можна сказати, що він розмаїтий та висококонкурентний. Розробники постійно шукають нові способи зацікавити та задовольнити гравців, використовуючи різноманітні геймплейні механіки та технології. Популярність мобільних ігор, ретро-стилю, онлайн-елементів, VR та AR, а також елементів розвитку персонажа свідчить про постійний розвиток та інновації в цій галузі.

1.4 Огляд популярних аркадних ігор

Дослідження спрямоване на аналіз успішних та визнаних проектів, що мають велику кількість прихильників серед геймерської спільноти. Розглядаються ігри з різноманітними жанрами, захоплюючими геймплейними механіками та оригінальними концепціями. Цей огляд надає можливість зрозуміти основні фактори успіху цих ігор та здобути цінний досвід для подальшої розробки власної аркадної гри.

1.4.1 Candy crash saga

Безкоштовна відеогра, присвячена зіставленню плиток, випущена King 12 квітня 2012, спочатку для Facebook; відбулися інші версії для iOS, Android, Windows Phone і Windows 10. У грі гравці проходять рівні, змінюючи місцями кольорові цукерки на ігровому полі, щоб скласти збіг із трьох або більше однакових кольорів, видаляючи ці цукерки з дошки та замінюючи їх на нові, що потенційно може створити нові збіги. Збіг чотирьох або більше цукерок створює унікальні цукерки, які діють як бонуси з більшими здібностями для очищення ігрового поля. На дошках є різні цілі, які необхідно виконати за фіксовану кількість ходів, наприклад, зібрати певну кількість цукерок певного типу.(рис. 1.1)

Рисунок 1.1 – Candy Crash

1.4.2 Subway Surfers

Мобільна гра нескінченний раннер, розроблена спільно Kiloo та SYBO Games, приватними компаніями з Данії. Він доступний на платформах Android, iOS, HarmonyOS, Kindle та Windows Phone і використовує ігровий движок Unity. У грі гравці беруть на себе роль молодих графіті-художників, які, спіймавши за «поміткою» залізничної ділянки метро, біжать залізницею сліди, щоб втекти від інспектора та його собаки. Під час бігу вони хапають по дорозі золоті монети, бонуси та багато інших предметів, одночасно уникаючи зіткнень із поїздами та іншими об'єктами. Вони також можуть стрибати на поїзди і кататися на ховербордах, щоб уникнути захоплення, доки персонаж не вріжеться в перешкоду, не буде спійманий інспектором або не буде збитий поїздом, після чого гра закінчується.(рис. 1.2)

Рисунок 1.2 – Subway Surfers

1.4.3 Angry Birds

Аркадна відеогра-головоломка 2009 року, розроблена фінським розробником відеоігор Rovio Entertainment. Натхнена в першу чергу ескізом стилізованих безкрилих птахів, гра була вперше випущена для пристроїв iOS та Maemo, починаючи з грудня 2009 року, що спонукало розробника розробити версії для інших смартфонів із сенсорним екраном, в першу чергу пристрої Android, Symbian, Windows Phone та BlackBerry 10 . З того часу серія розширилася і тепер включає ігри для спеціалізованих ігрових консолей і ПК. Ігровий процес обертається навколо гравців, які використовують рогатку, щоб запускати птахів у зелених свиней, розміщених у різних будовах або навколо них, з наміром знищити всіх свиней на ігровому полі. У міру просування гравців у грі стають доступними нові типи птахів.(рис. 1.3)

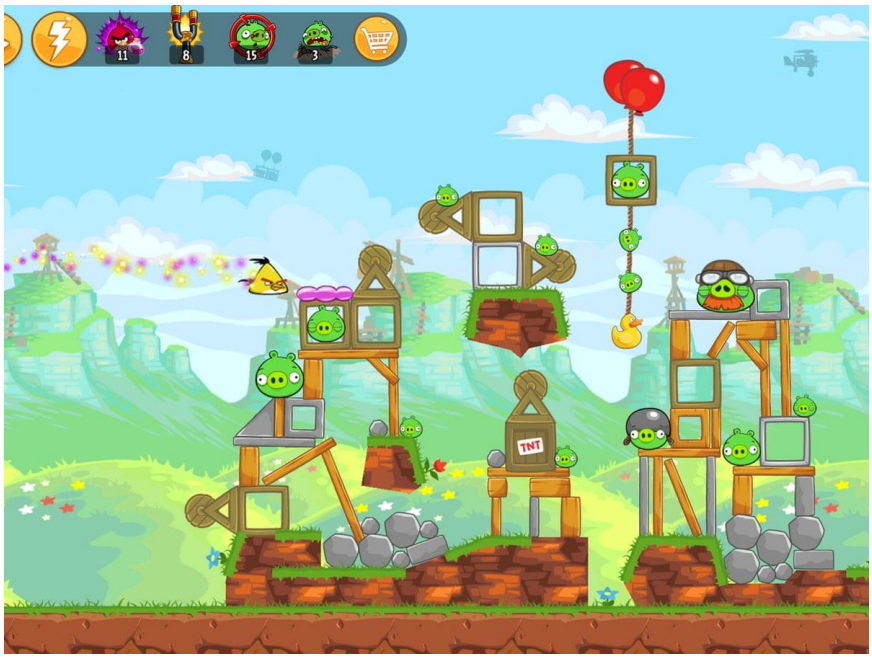

Рисунок 1.3 – Angrybirds

1.4.4 Temple run

3D-гра з нескінченним бігом, розроблена та видана Imangi Studios. Гравець управляє дослідником, який здобув давню реліквію і тікає від демонічних мавпоподібних істот, що переслідують його. Спочатку гра була випущена для пристроїв iOS 4 серпня 2011, а потім була перенесена на системи Android і Windows Phone 8.(рис. 1.4)

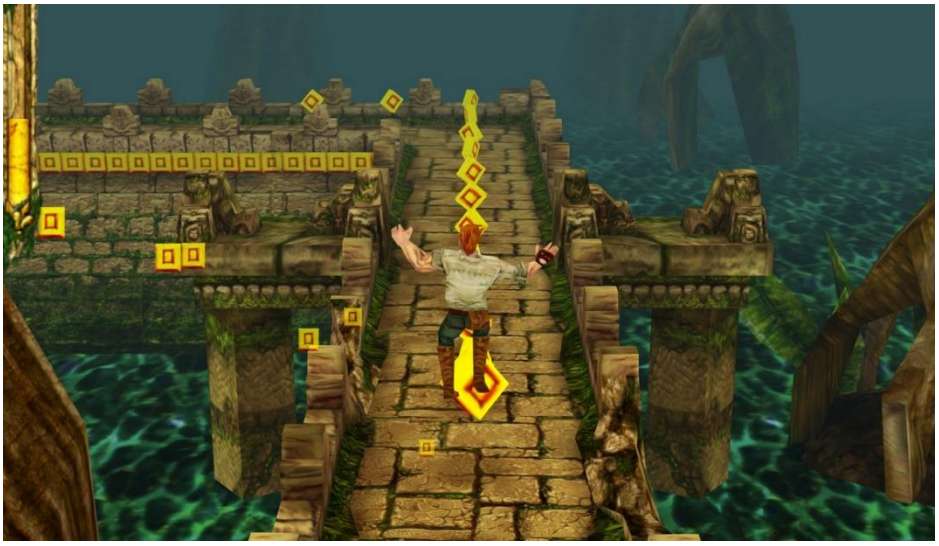

Рисунок 1.4. Temple Run

1.4.5 Brick vs Ball bricker

Гра була розроблена Hangzhou ZhiYu Technology у 2019, гра являє собою один з аналогів гри Arcanoid, мета гри полягає у вибору траєкторії польоту кульки, яка повинна знищити максимальну кількість кубиків.(рис.1.5)

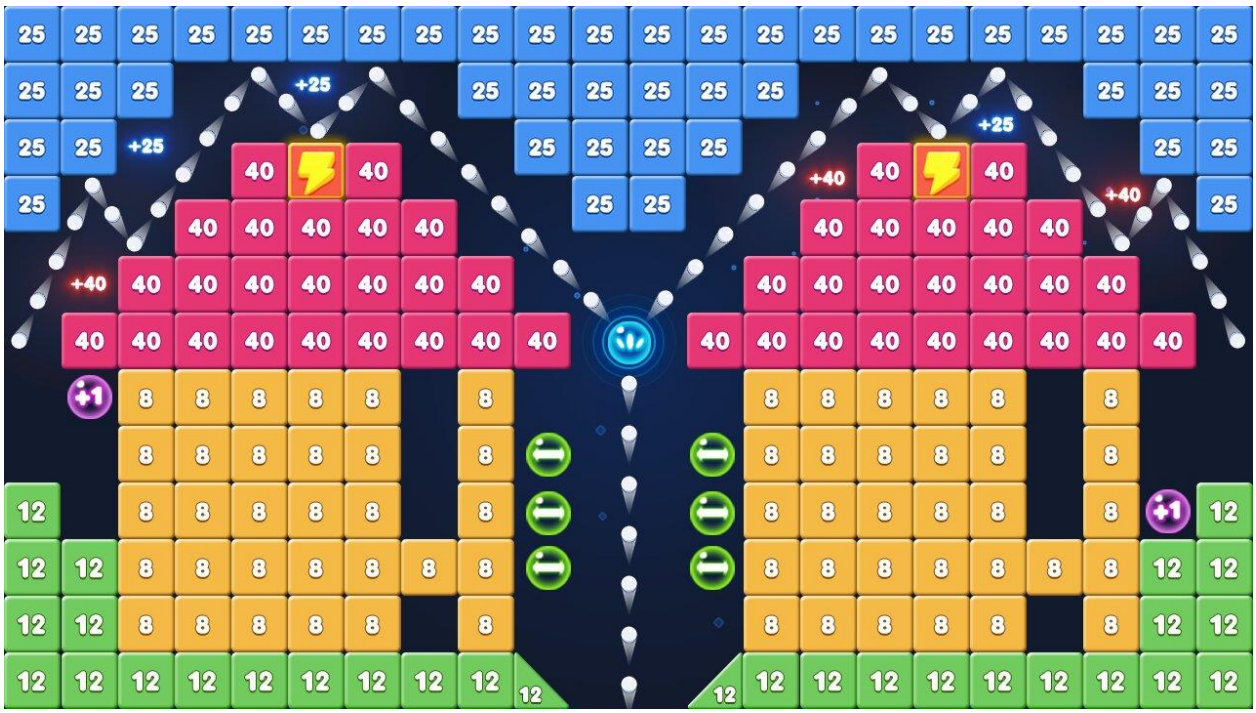

Рисунок 1.5 – Bricks vs Breaker

1.5 Мета проекту

Метою проекту є проведення аналізу засобів розробки комп’ютерних ігор з метою розробки власної комп’ютерної гри у жанрі Arkanoid, що включатиме основні риси цього жанру. Проект також спрямований на ознайомлення цільової аудиторії з цікавим, але не найпоширенішим жанром ігор. Розробка гри дає можливість отримати цінний досвід, який буде корисним не тільки при розробці ігор, але й у багатьох інших областях. Отриманий під час розробки досвід роботи з движком дозволить прискорити роботу над наступними проектами

2 РОЗРОБКА ПРОЕКТА

2.1 Поняття ігрового рушія

Ігровий рушій (Game engine, GE) – це програмне забезпечення, яке використовується для розробки ігор. Воно надає розробникам набір інструментів та функціоналу для створення графіки, фізики, штучного інтелекту, звуку, керування геймплеєм та інших аспектів гри. Ігровий рушій дозволяє ефективно управляти ресурсами, оптимізувати продуктивність і розробляти гри для різних платформ. Він є основою для створення ігрових проектів і забезпечує розробникам потужні інструменти для реалізації їхніх ідей та концепцій в ігровому середовищі.

2.2 Огляд безкоштовних GE

На даний момент нараховується величезна кількість GE, з безкоштовних більше всього виділяються три

2.2.1 Unity

Unity(рис2.1) – це багатоплатформовий ігровий рушій, розроблений Unity Technologies, вперше анонсований і випущений в червні 2005 року на Всесвітній конференції розробників Apple Worldwide Developers Conference як ігровий рушій для Mac OS X. З тих пір двигун поступово розширювався для підтримки різних настільних, мобільних, консольних платформ та платформ віртуальної реальності. Він особливо популярний у розробці мобільних ігор для iOS і Android, вважається простим у використанні для розробників-початківців і популярний та- для розробки інді-ігор.

Рушій можна використовувати для створення тривимірних (3D) та двовимірних (2D) ігор, а також інтерактивних симуляцій та іншого. Цей рушій був прийнятий у галузях, не пов'язаних з відеоіграми, таких як кіно, автомобілебудування, архітектура, машинобудування, будівництво та збройні сили США.

Unity дає користувачам можливість створювати ігри як у 2D, так і в 3D, а рушій пропонує основний API сценаріїв на C# з використанням Mono, як для редактора Unity у вигляді плагінів, так і для самих ігор, а також для перетягування(Drag & Drop). До того, як C# став основною мовою програмування, що використовується для рушія, він раніше підтримував Boo, який був видалений з випуском Unity 5, і засновану на Boo реалізацію JavaScript під назвою UnityScript, яка застаріла в серпні. 2017, після випуску Unity 2017.1 на користь C#.

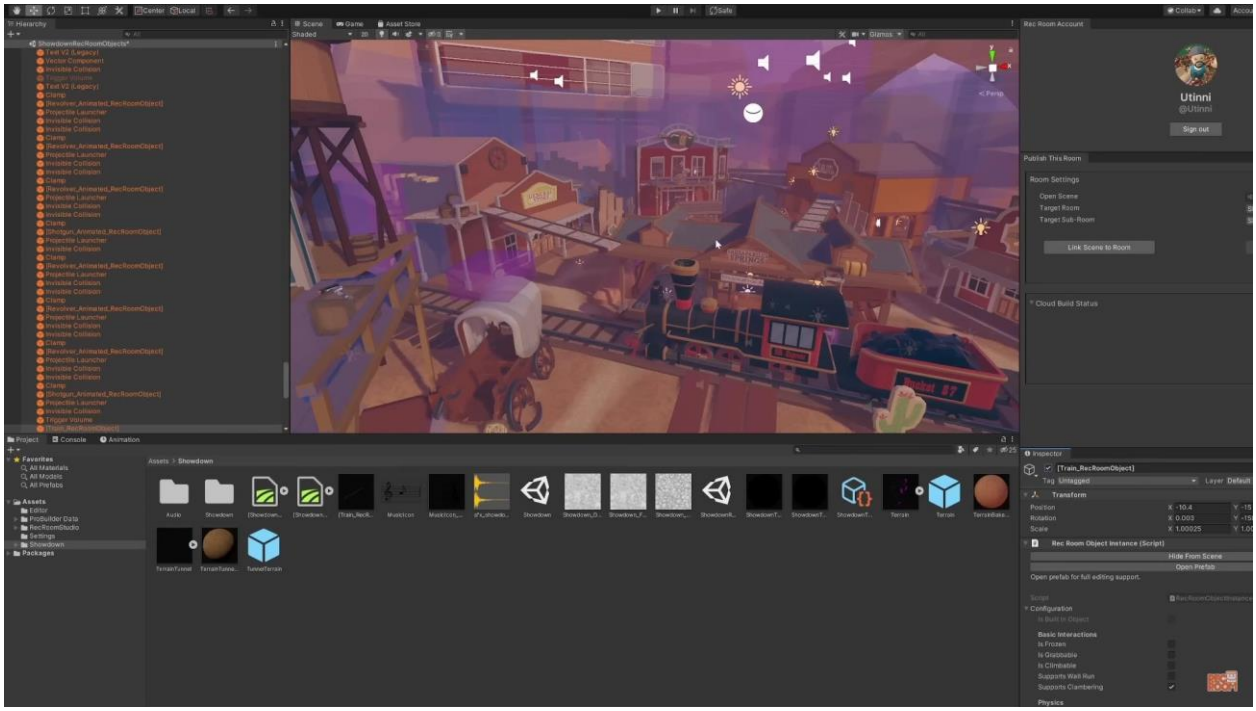

Рисунок 2.1 Unity інтерфейс

2.2.2 Gamemaker Sudio 2

GameMaker(рис. 2.2)-серія кросплатформових ігрових рушіїв, створених Марком Овермарсом в 1999 році і розроблюваних YoYo Games з 2007 року. Остання ітерація GameMaker випущена у 2022 році.

GameMaker дозволяє створювати кросплатформові та мультижанрові відеоігри з використанням мови візуального програмування з функцією перетягування або мови сценаріїв, відомої як Game Maker Language, яку можна використовувати для розробки більш просунутих ігор, які неможливо створити просто так. Використовуючи можливості візуального програмування. GameMaker спочатку був розроблений, щоб дозволити програмістам-початківцям створювати комп'ютерні ігри без особливих знань в області програмування за допомогою цих дій. Останні версії програмного забезпечення також орієнтовані на розвинених розробників.

GameMaker (рис.2.) в першу чергу призначений для створення ігор з 2D-графікою, дозволяючи використовувати растрову графіку, векторну графіку (через SWF), та 2D-скелетну анімацію (через Esoteric Software's Spine), а також велика стандартна бібліотека для малювання графіки та 2D-примітивів. Хоча програмне забезпечення дозволяє обмежене використання 3D-графіки, воно представлене у формі буфера вершин і матричних функцій і як таке не призначене для користувачів-початківців.

Рушій використовує Direct3D у Windows, UWP та Xbox One; OpenGL на macOS та Linux; OpenGL ES на Android та iOS, WebGL або 2D-канвас на HTML5 та пропрієтарні API на консолях.

GameMaker підтримує збірку для Microsoft Windows, macOS, Ubuntu, HTML5, Android, iOS, Amazon Fire TV, Android TV, Raspberry Pi, Microsoft UWP, PlayStation 4, Nintendo Switch та Xbox One; про підтримку PlayStation 5 та Xbox Series X|S було оголошено у лютому 2021 року.

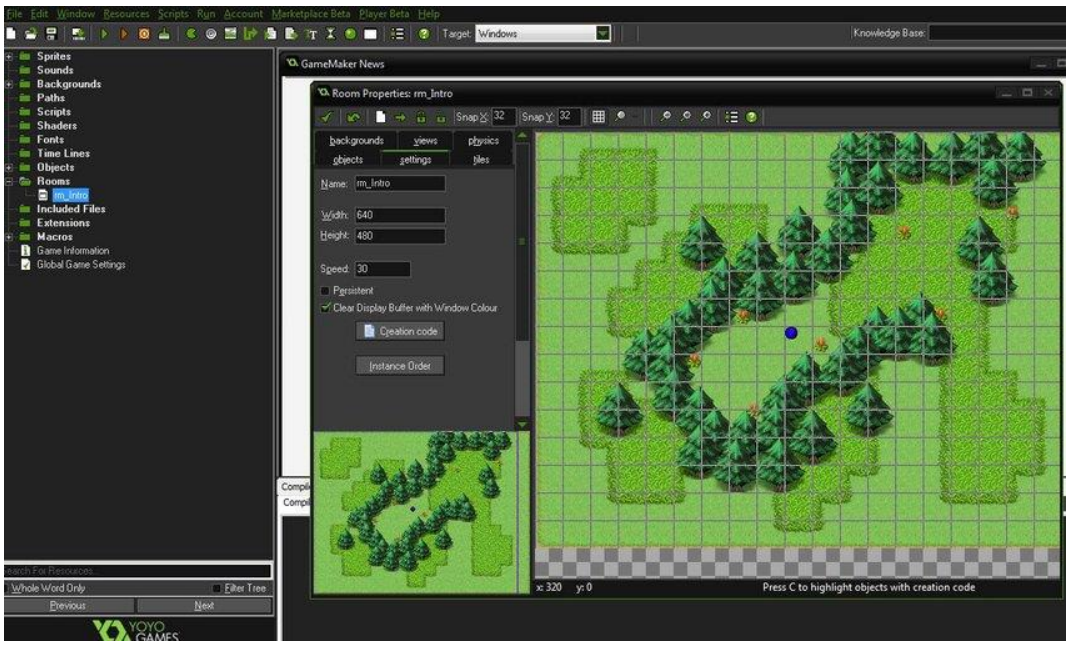

Рисунок 2.2 Gamemaker Sudio 2 інтерфейс

2.2.3 Unreal Engine 4

Unreal Engine (рис.2.3)- Ігровий рушій для комп'ютерної 3D-графіки, розроблений Epic Games, вперше представлений в 1998 році в шутері від першої особи Unreal. Спочатку розроблений для шутерів від першої особи для ПК, він з тих пір використовувався в різних жанрах ігор і був прийнятий в інших галузях, насамперед у кіно та телеіндустрії. Unreal Engine написаний на C++ і має високий рівень мобільності, підтримуючи широкий спектр настільних, мобільних, консольних платформ і платформ віртуальної реальності.

Unreal Engine 4 офіційно підтримує наступні платформи: Windows, macOS, Linux, iOS, Android, Nintendo Switch, PlayStation 4, Xbox One, PlayStation 5, Xbox Series X/S, Stadia, Magic Leap, HTC Vive, Oculus, PlayStation VR , Samsung Gear VR і HoloLens 2.

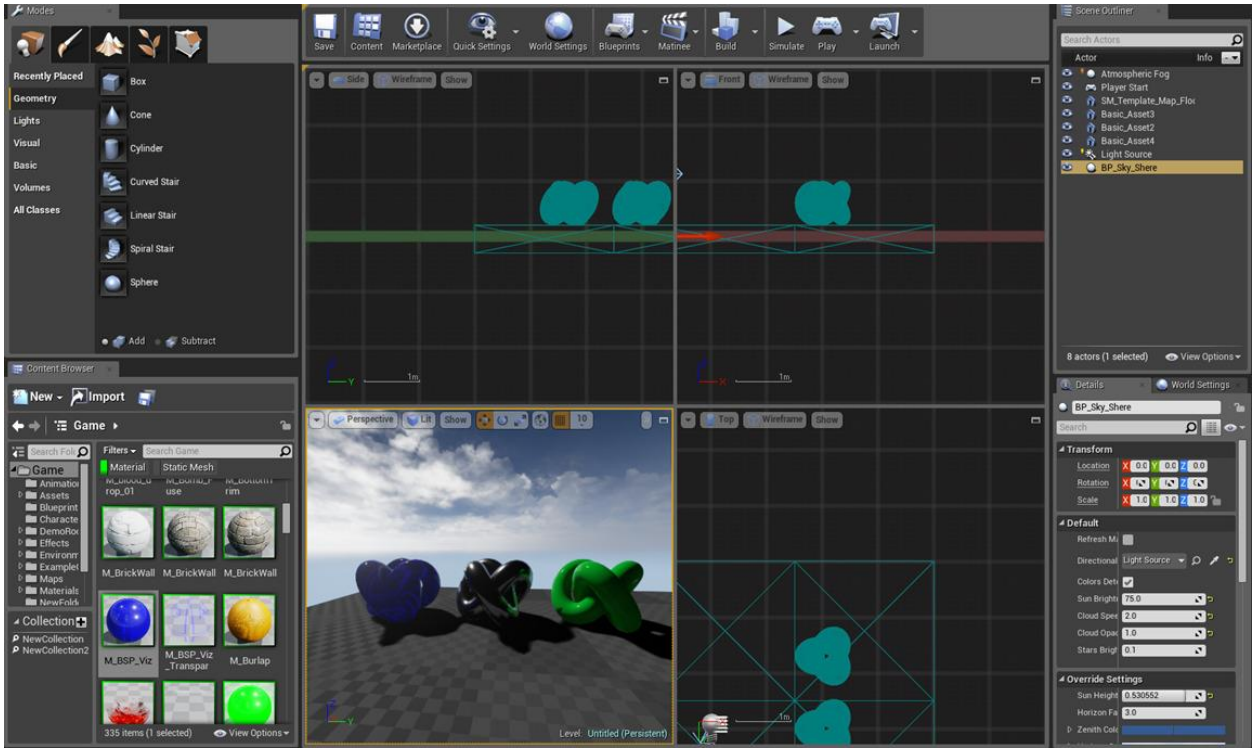

Рисунок 2.3 unreal engine інтерфейс

Якщо підвести підсумки то ми маємо:

Unity часто вважається одним з найкращих двигунів для розробки 2D-ігор, особливо в тому випадку, якщо користувач має досвід у програмуванні та розробці ігор. Unity має широкий спектр інструментів та ресурсів, що дозволяє розробникам створювати якісні та детальні 2D-ігри, що можуть бути запущені на різних платформах.

Unreal Engine 4 також має потужні можливості для розробки 2D-ігор, особливо якщо йдеться про графіку та візуальну якість. Він надає більш продуману систему фізики, а також має розширені засоби для відлагодження, що дозволяє розробникам більш точно контролювати гру.

GameMaker Studio забезпечує швидку та просту розробку 2D-ігор, що робить його ідеальним варіантом для початківців. Він має широкий спектр шаблонів та інструментів, що дозволяє розробникам швидко створювати прості 2D-ігри без складності в програмуванні.

2.3 Огляд популярних подібних проектів

Перед тим, як перейти до розробки власної гри "brick breaker", корисно ознайомитися з існуючими іграми, які належать до подібного жанру. Це дозволить нам зрозуміти основні риси і особливості цих ігор, а також виявити те, що може вплинути на нашу власну розробку.

1. "Brick Breaker" (Breakout):

"Brick Breaker" (рис. 2.4) є однією з найвідоміших ігор у жанрі "brick breaker". Гравець керує ракеткою, яка відбиває кулю для знищення блоків на рівні. У цій грі рівні генеруються автоматично, ігровий процес поступово стає складнішим з кожним рівнем. Розробка гри "Brick Breaker" зазвичай включає створення системи генерації рівнів, фізики відбивання кулі та різноманітних блоків з різними властивостями.

Рисунок 2.4 Brick breaker

2. "DX-Ball":

"DX-Ball" є ще однією популярною грою в жанрі "brick breaker". Вона має схожий геймплей, де гравець відбиває кулю для знищення блоків. Особливістю "DX-Ball" є наявність різних типів блоків, які мають різні властивості та ефекти. При розробці гри "DX-Ball" потрібно враховувати дизайн різноманітних блоків та їх взаємодію з кулею та ракеткою. (рис.2.5)

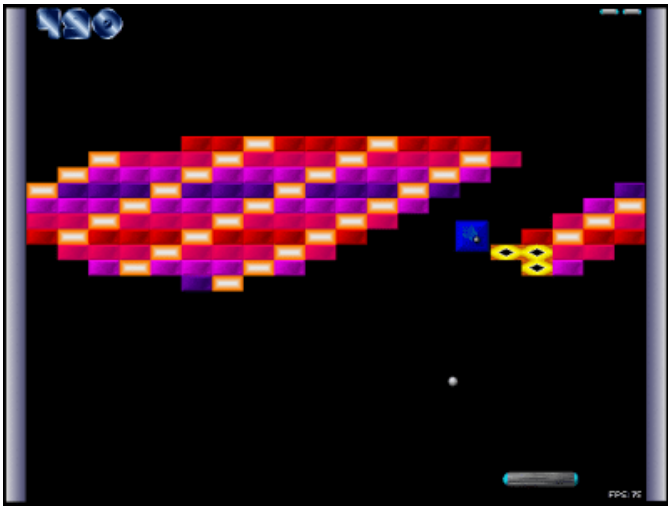

Рисунок 2.5 DX ball

3. "Shatter":

"Shatter" є сучасною версією гри "brick breaker" з додатковими елементами. У цій грі гравець може використовувати різні спеціальні властивості, які допомагають в руйнуванні блоків. "Shatter" також відрізняється відмінною графікою та ефектами. При розробці власної гри можна взяти на увагу такі додаткові елементи та спеціальні властивості, що збагатять геймплей. (рис. 2.6)

Рисунок 2.6 shatter

При розробці гри "brick breaker" ми можемо взяти на увагу ці існуючі ігри та їх розробку, але наша мета – створити унікальний досвід для гравців.

Варто використовувати найкращі практики, пристосовуючи їх до нашої власної концепції гри. За допомогою Unity та мови програмування C# ми матимемо необхідні інструменти для реалізації геймплею, фізики, графіки, звуку та управління.

2.4 Сценарій проекту

Метою гри "brick breaker" є проходження рівнів, де кожен рівень представляє собою нову складність та виклик для гравця. Головною ігровою механікою є відбивання кулі від ракетки та поверхонь рівня для знищення блоків. Гра пропонує взаємодію з гравцем через головне меню, де можна розпочати нову гру, переглянути правила та вийти з гри.

Розроблена система рівнів гри включає шість різних рівнів, прогресивно зростаючих за складністю. Гравець повинен успішно пройти кожен рівень, долаючи виклики та досягаючи максимального рахунку. Завершення гри настає після успішного проходження всіх рівнів.

Гра "brick breaker" є класичною аркадною грою, в якій гравець керує ракеткою для відбивання кулі та знищення блоків на рівні. Наша мета полягає у розробці повноцінної версії гри зі системою рівнів, геймплеєм, взаємодією з гравцем та завершенням гри.

Відтворення класичного геймплею "brick breaker" вимагає розробки різних компонентів, таких як графіка, фізика, звук, логіка гри та управління. Наша робота буде зосереджена на створенні цих компонентів, їх взаємодії та налагодженні геймплею, щоб створити захоплюючий досвід гри для гравців.

Основні елементи гри "brick breaker" включають ракетку, кулю, блоки та рівні гри. Гравець буде керувати ракеткою, намагаючись відбити кулю так, щоб вона зіткнулася з блоками та знищила їх. Гра буде містити систему рівнів, де кожен рівень буде складніший за попередній.

У цьому розділі ми розглянемо розробку сценарію гри "brick breaker". Ми описуємо послідовність подій, взаємодію гравця з грою та специфічні

аспекти геймплею. Наша мета – створити захоплюючий та цікавий геймплей, який заохочуватиме гравців пройти всі рівні та досягти високих результатів.

Для кращого розуміння сценарій можно поділити на такі пункти:

1)Головне меню

– Гравець запускає гру і потрапляє до головного меню. Тут він має наступні варіанти:

– Нова гра: Гравець обирає цей пункт для початку нової гри;

– Правила: Гравець може переглянути правила гри та ознайомитись з ігровим процесом;

– Вихід: Гравець може вийти з гри.

2)Початок гри

– Після обрання "Нова гра" гравець потрапляє на перший рівень гри. На початку рівня розміщені блоки, ракетка та куля;

– Ракетка: Гравець управляє ракеткою, рухаючи її вліво або вправо для відбивання кулі;

– Куля: Куля рухається в напрямку блоків та поверхонь рівня. Гравець повинен відбивати кулю, щоб знищити блоки;

– Блоки: Рівень містить різні блоки, які гравець має знищити відбиваючи кулю від ракетки.

3)Ігровий процес

– Гравець активно взаємодіє з грою, намагаючись знищити всі блоки та досягти якомога більшого рахунку;

– Відбивання кулі: Гравець відбиває кулю від ракетки, намагаючись направити її в блоки та поверхні рівня для їх знищення;

– Знищення блоків: Коли куля зіткнувається з блоком, він зникає, а гравець отримує бали;

– Збільшення складності: З кожним наступним рівнем з'являються нові блоки та змінюється їх розташування, що ускладнює завдання гравця.

4)Завершення гри

– Гра завершується, коли гравець успішно пройшов всі рівні або коли він програв, не змоговши відбити кулю від ракетки.
– Успішне проходження рівнів: Коли гравець успішно проходить всі рівні, він отримує повідомлення про перемогу та може почати нову гру.
– Програш: Якщо гравець не змоговши відбити кулю, він отримує повідомлення про програш та може почати нову гру.

5)Продовження гри

– Якщо гравець програв, він має можливість почати нову гру або повернутись до головного меню.

2.5 Збірка проекту

Ігрові рушії надають можливість збирати проекти для різних платформ, включаючи ПК, консолі та мобільні пристрої. Для ПК та мобільних пристроїв процес збірки зазвичай простий і вимагає лише одного кліка. Однак, для інших платформ може знадобитися встановлення додаткового SDK (набору розробки програмного забезпечення), що дозволить адаптувати гру до вимог і специфіки цих платформ.

2.6 основниі функції проекту

Проект переробки гри Arkanoid використовуючи C# і Unity 2D передбачає впровадження різноманітного функціоналу для покращення геймплею. Перш за все, буде реалізовано систему рівнів, де гравцеві буде запропоновано пройти послідовність складніших рівнів з різноманітними завданнями та блоками. Додадуться нові види блоків з унікальними властивостями та бонусами, що збагатить геймплей. Також планується впровадження системи покращення м'яча та ракетки, де гравці матимуть можливість вдосконалювати їх характеристики під час гри. Крім того, буде

реалізовано систему рекордів, яка дозволить гравцям зберігати свої досягнення та змагатися з іншими гравцями. У проекті будуть використані різні звукові ефекти та візуальні покращення для створення захоплюючої атмосфери гри. Цей проект повинен відповідати деяким вимогам для забезпечення успішної розробки і готовності до впровадження. Основні вимоги для проекту переробки гри Arkanoid включають:

– Функціональні вимоги: Реалізація основних механік гри, включаючи керування ракеткою, рух м'яча, взаємодію з блоками та систему рівнів. Наявність системи бонусів та покращень, системи рекордів і можливості перезапуску гри;

– Графічні вимоги: Створення візуальних елементів, таких як блоки, ракетка, м'яч, фон та ефекти руху. Графіка повинна бути привабливою та відповідати стилю гри;

– Звукові вимоги: Використання звукових ефектів для підсилення геймплею, таких як звук зіткнень, звукові ефекти під час знищення блоків та фонова музика;

– Продуктивність: Забезпечення плавної роботи гри без помітних затримок або лагів, оптимізація ресурсів і швидка реакція на взаємодію гравця;

– Переносимість: Забезпечення можливості запуску гри на різних платформах, таких як ПК, консолі та мобільні пристрої. При необхідності використання платформозалежних SDK;

– Ігрова механіка: Сповнення гри різноманітними та викликаючими цікавість елементами, які збережуть високий рівень геймплею та стимулюватимуть гравця до подальшого проходження гри.

2.6 Інформаційна модель

Інформаційна модель гри "Brick breaker" описує структуру даних та взаємозв'язки між їхніми компонентами. Ця модель визначає, які дані будуть використовуватися в грі, як вони будуть організовані та оброблятися.

Основні елементи інформаційної моделі гри "Brick breaker" включають:

1. Гравець/ракетка (Pad):

– Позиція: координати по осі X, Y, що визначають положення ракетки на екрані;

– Розмір: ширина і висота ракетки для відображення на екрані;

– Стан: активна/неактивна, яка визначає, чи може гравець керувати ракеткою.

2. Куля (Ball):

– Позиція: координати по осі X, Y, що визначають положення кулі на екрані;

– Розмір: діаметр кулі для відображення на екрані;

– Швидкість: вектор швидкості, що визначає напрямок та швидкість руху кулі;

– Стан: активна/неактивна, яка визначає, чи перебуває куля в грі.

3. Блоки (Bricks):

– Позиція: координати по осі X, Y, що визначають положення блоків на екрані;

– Розмір: ширина і висота блоків для відображення на екрані;

– Стан: цілий/пошкоджений/знищений, що визначає стан блоків після зіткнення з кулею.

4. Рівні (Levels):

– Конфігурація блоків: розташування блоків на рівні, їхні типи та кольори;

– Складність: рівень складності рівня, який впливає на швидкість руху кулі, розміщення блоків та інші фактори.

– Система очків (Scoring System):

– Очки гравця: кількість очків, які гравець отримує за знищення блоків;

– Правила нарахування очків: логіка нарахування очків в залежності від типу блоку, рівня складності та інших факторів;

5. Звукова система (Sound System):

– Звукові ефекти: звуки, які відтворюються під час різних подій в грі, наприклад, зіткнення кулі з блоком або ракеткою.

Ці елементи інформаційної моделі визначають основні складові гри "Brick breaker" та забезпечують необхідну базу для розробки скриптів та логіки гри. Інформаційна модель допомагає організувати дані і забезпечує їхню взаємодію для створення унікального геймплею та досвіду для гравців.

2.6 Розробка алгоритму

1. Рух ракетки: Розробляється алгоритм, що дозволяє гравцю керувати ракеткою. Цей алгоритм визначає, як ракетка рухатиметься вгору, вниз, ліворуч або праворуч, відповідно до введення користувача. Також можуть бути враховані обмеження щодо межі руху ракетки на екрані.

2. Рух кулі: Розробляється алгоритм, що визначає шлях руху кулі після її запуску. Цей алгоритм враховує кут напрямку та швидкість кулі, а також можливі зіткнення з іншими об'єктами. При зіткненні з ракеткою або блоком, алгоритм змінює напрямок руху кулі.

3. Зіткнення та взаємодія: Розробляються алгоритми, що визначають наслідки зіткнення кулі з різними об'єктами гри. При зіткненні з блоком алгоритм виконує видалення блоку з екрану та нарахування гравцю очків. Також можуть бути розроблені алгоритми для спеціальних блоків, які надають додаткові можливості або ефекти гравцю.

4. Рівні гри: Розроблюються алгоритми, що визначають структуру та складність кожного рівня гри. Ці алгоритми визначають кількість та розташування блоків на кожному рівні, а також встановлюють умови для завершення рівня та перехід до наступного.

5. Штучний інтелект: В розробці гри "Brick breaker" може бути включений комп'ютерний супротивник, який контролює свою ракетку. Для цього розробляються алгоритми штучного інтелекту, які дозволяють комп'ютеру ефективно відбивати кулю та реагувати на рухи гравця. Це може

включати розробку стратегій руху, прогнозування траєкторії кулі та прийняття рішень щодо оптимального руху ракетки.

В процесі розробки алгоритмів використовуються мова програмування C# та фреймворк Unity. C# надає потужні засоби для роботи з об'єктами, управління потоками та обробки подій. Unity забезпечує інтеграцію графічного движка, фізичного моделювання та інших ресурсів, що дозволяють легко реалізувати графіку, звуки та фізику гри. Застосування C# та Unity у поєднанні з розробленими алгоритмами дозволяють створити функціональну гру "Brick breaker".

3 РОЗРОБКА АРКАДНОЇ ГРИ BRICK BREACKER

3.1 Вибір середовища розробки

Було вирішено використовувати найпопулярніше середовище розробки як Visual Studio(VS), а саме VS 2019.[5] (рис. 3.1)

Visual Studio – це інтегроване середовище розробки (IDE) від Microsoft. Він використовується для розробки комп'ютерних програм, включаючи веб-сайти, веб-додатки, веб-служби та мобільні додатки. Visual Studio використовує платформи розробки програмного забезпечення Microsoft, такі як Windows API, Windows Forms, Windows Presentation Foundation, Windows Store та Microsoft Silverlight. Він може створювати як власний код, і керований код.

Visual Studio включає редактор коду, що підтримує IntelliSense (компонент завершення коду), а також рефакторинг коду. Вбудований налагоджувач працює як налагоджувач рівня вихідного коду і як налагоджувач рівня комп'ютера. Інші вбудовані інструменти включають профільувальник коду, конструктор для створення додатків з графічним інтерфейсом, веб-дизайнер, конструктор класів та конструктор схеми бази даних. Він приймає плагіни, які розширюють функціональність практично на кожному рівні, включаючи додавання підтримки систем керування вихідним кодом (таких як Subversion і Git).) та додавання нових наборів інструментів, таких як редактори та візуальні дизайнери для предметно-орієнтованих мов або наборів інструментів для інших аспектів життєвого циклу розробки програмного забезпечення (наприклад, клієнт Azure DevOps: Team Explorer).

Visual Studio 2019 є однією з найбільш популярних та потужних інтегрованих середовищ розробки (IDE) для програмування на різних мовах програмування, включаючи C#, який використовується в Unity для розробки 2D-ігор.

Ось деякі плюси середовища розробки Visual Studio 2019 для розробки 2D-ігор з Unity:

– Інтеграція з Unity: Visual Studio 2019 підтримує повну інтеграцію з Unity, що дозволяє легко редагувати та налагоджувати код Unity в середовищі розробки Visual Studio без необхідності переходити між програмами;

– Підсвічування синтаксису та автодоповнення: Visual Studio 2019 забезпечує підсвічування синтаксису та автодоповнення, що допомагає зменшити кількість помилок та зробити розробку коду більш продуктивною;

– Інструменти для відлагодження: Visual Studio 2019 має розширені інструменти для відлагодження, що дозволяє розробникам відлагоджувати код Unity більш ефективно, дозволяючи розглядати значення змінних та виконувати код по крокам;

– Підтримка Git: Visual Studio 2019 підтримує Git, що дозволяє розробникам зберігати та керувати кодом Unity в репозиторії Git, співпрацювати з іншими розробниками та здійснювати версіювання коду;

– Розширені можливості програмування: Visual Studio 2019 надає розширені можливості програмування, такі як refactoring, генерування коду, відлагодження динамічних запитів та інші, що дозволяє зменшити час, необхідний для написання коду та збільшити продуктивність розробки;

– Підтримка різних платформ: Visual Studio 2019 дозволяє розробникам створювати ігри для різних платформ, таких як Windows, Mac, iOS, Android та інші;

– Налаштування: Visual Studio 2019 дозволяє розробникам налаштувати середовище розробки відповідно до їх потреб. Розробники можуть вибирати між різними темами, шрифтами, розмірами шрифтів та іншими параметрами для підвищення комфорту роботи;

– Документація: Visual Studio 2019 має добре розроблену документацію, яка допомагає розробникам знайти рішення на будь-яке запитання та проблему. Розробники можуть легко знайти інформацію про класи, функції, методи та інші елементи коду;

– Інтеграція з Azure: Visual Studio 2019 має інтеграцію з хмарними службами Azure, що дозволяє розробникам зберігати та керувати своїми проектами, базами даних та іншими ресурсами в хмарі;

– Розширені можливості плагінів: Visual Studio 2019 має багато плагінів, що дозволяє розширити можливості середовища розробки та зробити роботу з Unity ще більш продуктивною та зручною.

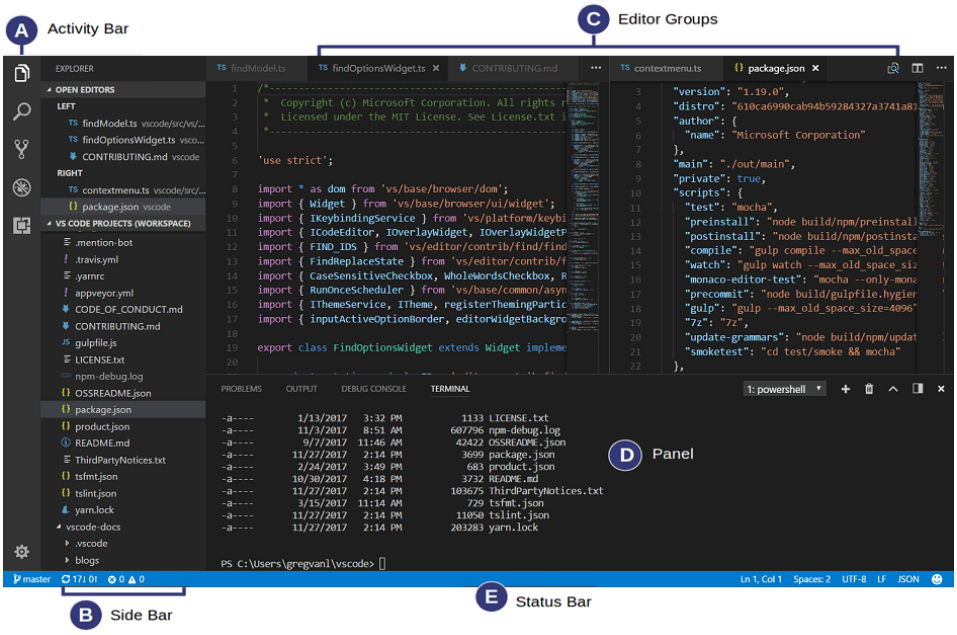

Рисунок 3.1 Visual Studio інтерфейс

3.2 Налаштування конфігурації проекту

У Unity існують налаштування компіляції для кожної платформи, які визначають, як програма буде зібрана та працювати на конкретній платформі. Основні особливості кожної платформи в Build Setting у Unity такі:

1. PC, Mac & Linux Standalone:

– Вибір платформи: Windows, Mac чи Linux;

– Розрядність: 32-bit чи 64-bit;

– Тип білда: Debug або Release;

– Параметри складання: опції складання, які залежать від платформи, наприклад, налаштування екрана та увімкнення/вимкнення функцій.

2. Android:

– Архітектура: вибір архітектури процесора, яка визначає, як програма буде працювати на пристроях Android;

– Тип білда: Debug або Release;

– Маніфест: конфігураційний файл, який визначає інформацію про пристрій, який використовуватиме програму;

– Підпис: сертифікат, який використовується для підписання програми перед завантаженням на Google Play Store.

3. iOS:

– Тип білда: Debug або Release;

– Схема складання: налаштування складання, такі як вибір цільового пристрою та увімкнення/вимкнення функцій;

– Provisioning Profile: файл, який використовується для ідентифікації та перевірки розробника та програми перед його завантаженням на пристрої iOS;

– Ключі та сертифікати: використовуються для підписання та складання програми.

4. WebGL:

– Тип білда: Debug або Release.

– Розмір буфера: розмір буфера, який використовується для завантаження та відтворення веб-сторінки з WebGL-програмою;

– Стиснення текстур: увімкнення/вимкнення стиснення текстур для зменшення розміру файлів, що завантажуються.

5. Universal Windows Platform:

– Тип білда: Debug або Release;

– Тип пакета: вибір типу пакета програми, наприклад APPX або MSIX;

– Розмір буфера: розмір буфера, який використовується для завантаження та відтворення програми;

– Маніфест: конфігураційний файл, який визначає інформацію про пристрій, який використовуватиме програму.

3.3 Графічне оформлення

Основна складова будь-якої комп'ютерної гри – її візуальна складова, і перед тим як говорити про частину графічного оформлення, слід розібратися з основними поняттями, що використовуються в геймдизайні, такими як тайл та спрайт.

Тайл – це повторюваний фрагмент невеликих розмірів, що використовується для створення зображень великих розмірів за допомогою тайлової графіки. Цей метод використовується для створення рівнів у двовимірних та тривимірних іграх.

Спрайт – це графічне зображення об'єкта у двовимірних іграх, яке може містити кілька зображень для створення анімації.

У процесі побудови графічної стилістики мною були взяті спрайти з безкоштовного сайту для пошуку та розробки дизайну ігор kenney.nl[7] (рис. 3.2)

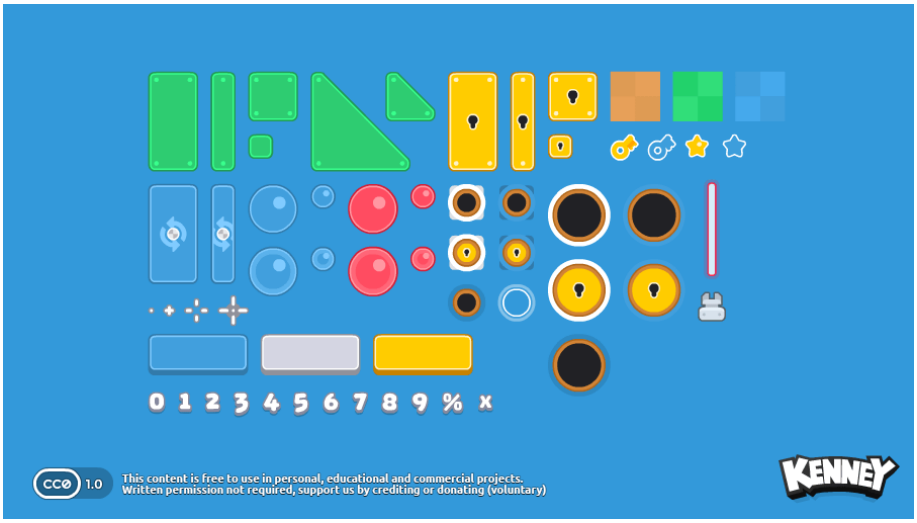

Рисунок 3.2— Спрайти

Для покращення візуального стилю гри мною був використано арт у 8-bit стилі під назвою pixel night взятий з сайту pxfuel.com[16] (рис. 3.3 Задній фон)

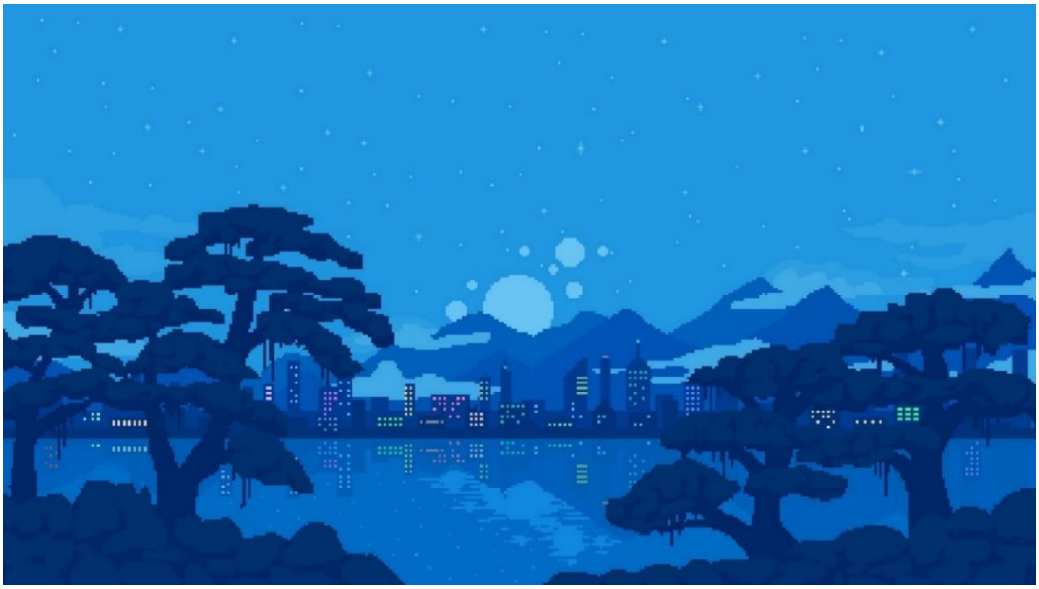

Рисунок 3.3 Задній фон

3.3 Структура гри

Початкове вікно гри містить у собі назву, кнопку старт, правила, рівні та кнопку виходу з гри. (рис. 3.4)

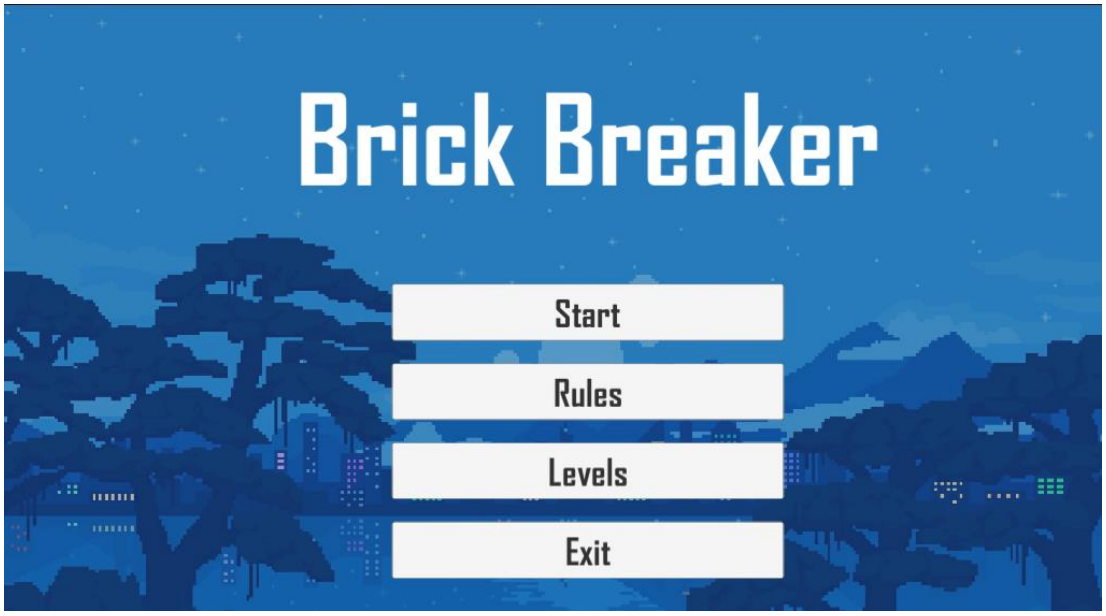

Рисунок 3.4 — Початкове меню

Якщо натиснути кнопку Rules то ви перейдете до вікна з підказками де вказуються правила та пояснення властивостей об’єктів у грі (рис. 3.5)

Рисунок 3.5 — Rules

Якщо ви натиснете кнопку Levels ви перейдете до меню вибору рівнів. На даний момент гра складається з 6 рівнів (у майбутньому може бути більше). (рис. 3.6)

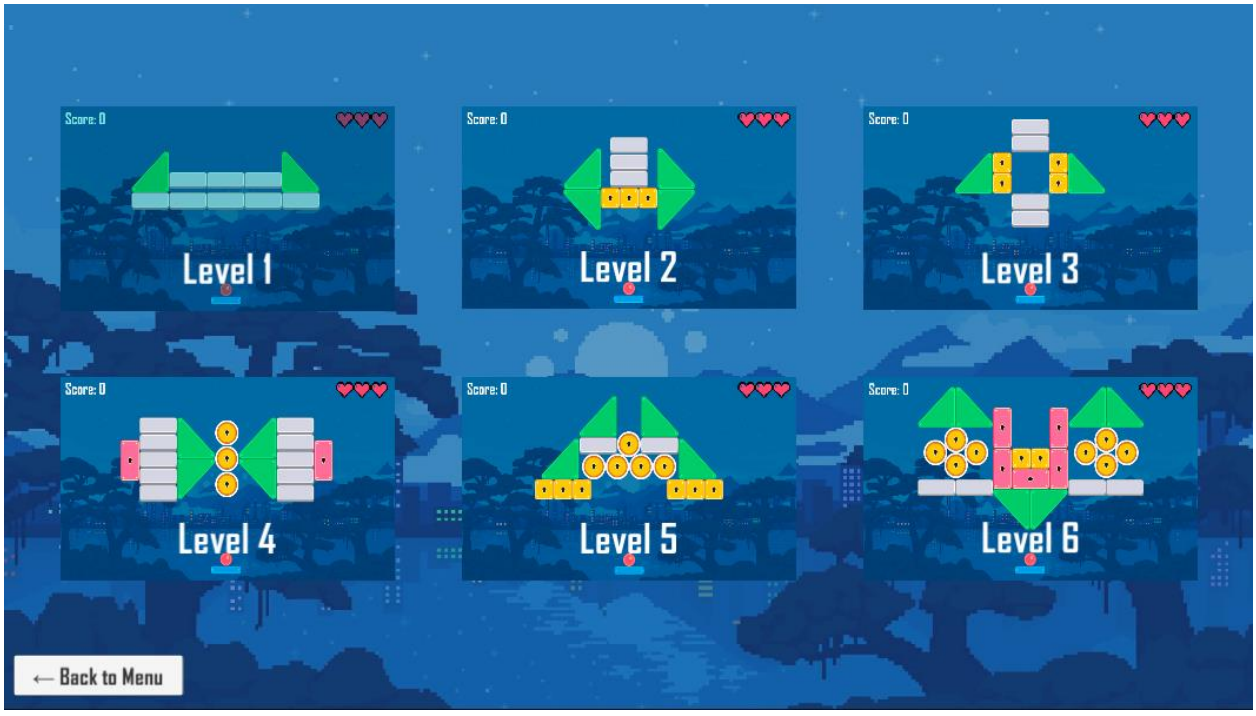

Рисунок 3.6 —Рівні

Управління грою відбувається за допомогою миші, гравець має три одиниці життя(HP) які витрачаються якщо куля впала до низу(рис. 3.7). Кожен блок має свою текстуру, свою міцність та звук знищення, з блоків з різним шансом може випасти або посилення або послаблення (дивитись рисунок 3.5), коли у гравця закінчилось HP викликається меню з кнопками почати гру з першого рівня, спробувати пройти рівень ще раз, чи вийти у головне меню, (рис. 3.8)

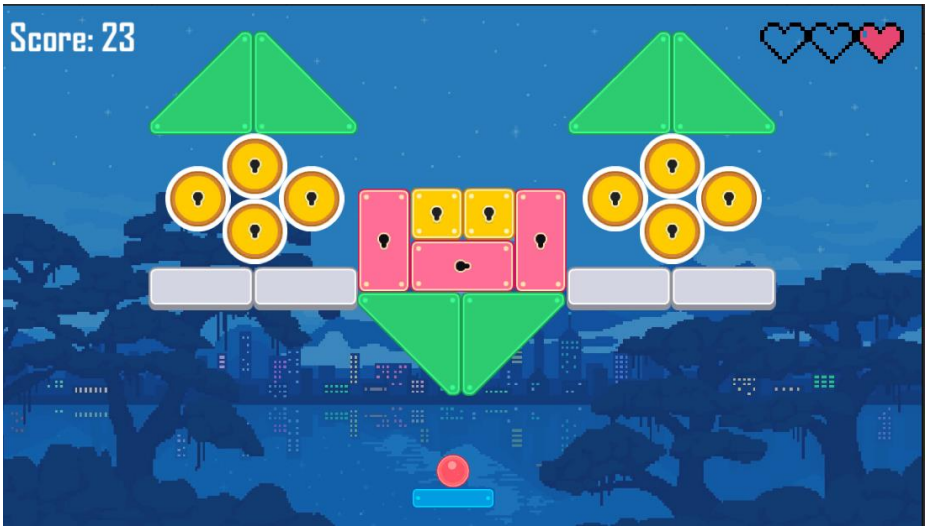

Рисунок 3.7— Втрачено два життя

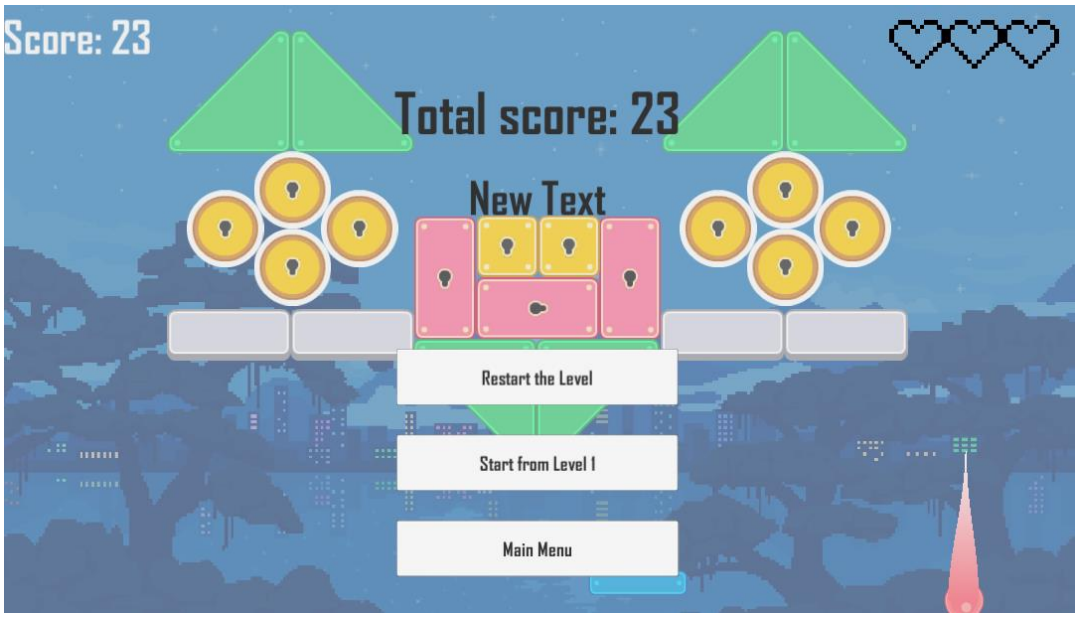

Рисунок 3.8 —Меню програшу

3.4 Створення класів

3.4.1 Game Manager

Після розробки спрайтів була створена папка Scripts у якій знаходяться усі скрипти та вихідний код до гри. Код GameManager головний скрипт у цьому проекту, та є основою усієї гри. Цей код відповідає за керування грою, оновлення рахунку, обробку паузи та збереження найкращого результату гравця.

Game Manager оголошує змінні, які використовуються для збереження та відображення інформації про гру, таку як текст рахунку, поточний рахунок, стан паузи та панель паузи. bestResult використовується для збереження найкращого результату гравця.

Метод Awake() викликається при створенні об'єкта на сцені. У цьому методі перевіряється, чи існує вже інший об'єкт GameManager на сцені. Якщо такий об'єкт існує, поточний об'єкт вимикається (gameObject.SetActive(false)) або може бути знищений (Destroy(gameObject)). bestResult ініціалізується зі значенням, яке зберігається в PlayerPrefs (зберігається найкращий результат гравця).

Метод AddScore() додає задане значення до поточного рахунку гравця. Рахунок оновлюється на екрані, а найкращий результат зберігається в PlayerPrefs.

3.4.2 Ball

Цей код відповідає за керування поведінкою кулі в грі, включаючи її рух, розмір, взаємодію з іншими об'єктами та спеціальні ефекти.

– Змінні: фізичне тіло (Rigidbody2D), швидкість (initialSpeed, currentSpeed), розмір (ballSize), стани (isStarted, isMagnetActive, isExplosive), зовнішні ефекти та звуки.

– Методи: Метод Awake()викликається при створенні об'єкта на сцені. У цьому методі виконується отримання компонента AudioSource, який використовується для відтворення звуків кулі. Метод Start() викликається після методу Awake() та при створенні об'єкта на сцені. У цьому методі встановлюються початкові значення розміру кулі, поточної швидкості, позиції по осі Y та стану вибуху (isExplosive). Метод Update() викликається кожен кадр гри. У цьому методі перевіряється, чи куля рухається (isStarted). Якщо так, то її швидкість оновлюється відповідно до поточної швидкості.ActivateExplode(): Активує ефект вибуху (explodeEffect) та відтворює звук. MultiplySpeed(float speedCoeff): Множить поточну швидкість на заданий коефіцієнт.MagnetActivate(): Активує магнітний ефект. BallScale(float ballScale): Змінює розмір кулі, включаючи її візуальний розмір та ширину сліду. DuplicateBall(): Дублює кулю. StartBall(): Задає початкову швидкість та напрямок руху кулі. Explode(): Знаходить блоки, що перетинаються з радіусом вибуху та знищує їх.

– Обробники подій:private void OnCollisionEnter2D(Collision2D collision): Обробляє зіткнення кулі з іншими об'єктами. Відтворює звук зіткнення та реагує на певні умови, такі як активний магнітний ефект (isMagnetActive) або активований режим вибуху (isExplosive).

Цей код відповідає за керування поведінкою кулі в грі, включаючи її рух, розмір, взаємодію з іншими об'єктами та спеціальні ефекти.

3.4.3 Blocks

Цей скрипт відповідає за функціональність блоків гри, їх життя, візуальні ефекти та вплив на геймплей (очки, підсилення).

– Змінні та посилання: Змінна points визначає кількість очків, які гравець отримує при знищенні блоку. GameManager та LevelChanger є посиланнями на об'єкти відповідних класів, які використовуються для управління грою.

– Змінні блоку: lifeBlock визначає кількість життів блоку, яку можна зменшувати при зіткненні з іншими об'єктами. Змінна invisible вказує, чи блок на початку гри невидимий. Масив blockDamaged містить спрайти, які використовуються для зображення блоку в пошкодженому стані. sRender є посиланням на компонент SpriteRenderer, який керує відображенням спрайту блоку.

– Ефекти та звуки: particlePrefab є префабом, який створюється при знищенні блоку для створення ефекту частинок. pickUps містить масив об'єктів, які представляють можливі підсилення, які можуть випасти після знищення блоку. audioSource є компонентом AudioSource, який відтворює звукові ефекти блоку. blockDestroyedSound є аудіокліпом, який відтворюється при знищенні блоку. sM є посиланням на об'єкт класу SoundManager, який керує звуковими ефектами в грі.

– Методи:Awake(): Метод, що викликається при створенні об'єкта.

Встановлює посилання на необхідні компоненти та об'єкти. Start(): Метод, що викликається перед першим кадром гри. Викликає метод BlockCreated() у LevelChanger та встановлює невидимість блоку, якщо це необхідно. OnCollisionEnter2D(Collision2D collision): Метод, що викликається при зіткненні з іншим об'єктом. Зменшує кількість життів блоку та встановлює пошкоджений спрайт. Якщо кількість життів стає менше або дорівнює нулю, то викликає метод DestroyBlock() для знищення блоку. DestroyBlock(): Метод, що викликається для знищення блоку. Збільшує рахунок гравця, відтворює звук знищення блоку, створює ефект частинок та випадково генерує підсилення pickUps. PickUpChance(): Метод, що випадково вибирає підсилення з pickUps та створює його об'єкт у позиції блоку.

3.4.4 GameDataSO

Даний скрипт використовується для зберігання ігрових даних. Він містить дві публічні змінні типу int: s та h. Скрипт успадковується від базового

класу ScriptableObject і має атрибут [CreateAssetMenu], що дозволяє створювати екземпляри цього скрипту як ресурси у редакторі Unity. Такий підхід дозволяє централізовано зберігати дані та обмінюватися ними між різними компонентами та сценами у проекті.

3.4.5 GameOver

Цей код відповідає за відображення результата гри та керування певними елементами геймплею.

– Імпортує необхідні простори імен.

– Оголошує клас GameOver, який успадковується від MonoBehaviour.

– Оголошує змінні totalScore, bestScore, gm і hearts для зберігання посилань на інші об'єкти або компоненти.

– У методі Awake() знаходить і зберігає посилання на об'єкти GameManager і HeartsBar.

У методі Start() встановлює текст для totalScore і bestScore на основі значень змінних gm.score і gm.bestResult. Також виконує деякі додаткові дії, такі як вимкнення hearts та gm.scoreText.

3.4.6 HeartsBar

Цей скрипп відповідає за відображення та керування сердечками (здоров'ям) гравця, включаючи їх зміну, видалення та перевірку на умову поразки

– Імпортує необхідні простори імен.

– Оголошує клас HeartsBar, який успадковується від MonoBehaviour.

– Оголошує змінні hearts, fullHeart, emptyHeart, health, heartsNumber, startHeartsNumber, ball, gameOverPanel, gm і totalScore.

– У методі Start() знаходить і зберігає посилання на об'єкт GameManager, та вимикає всі сердечка, крім перших heartsNumber.

– Метод MinusHeart() зменшує значення health і оновлює відображення сердечок відповідно до health та heartsNumber. Викликає метод GameOver().

– Метод AddRemoveHeart() додає або видаляє сердечка залежно від значення addHeart.

– Метод HeartsStart() встановлює початкові значення для змінних isDead, isPaused, score, Time.timeScale і heartsNumber. Змінює відображення сердечок відповідно до heartsNumber.

– Метод GameOver() перевіряє, чи досягнуто умови поразки (якщо health менше або дорівнює 0). Встановлює isDead в true, активує панель gameOverPanel, зупиняє час гри (Time.timeScale = 0f) і встановлює текст totalScore на основі значення gm.score.

3.4.7 LevelChanger

Цей скрипт відповідає за зміну рівня гри після того, як усі блоки на поточному рівні були знищені.

– Імпортує необхідні простори імен.

– Оголошує клас LevelChanger, який успадковується від MonoBehaviour.

– Оголошує змінні blocksCount, gm і heartsBar.

– У методі Start() знаходить і зберігає посилання на об'єкти GameManager і HeartsBar.

– Метод BlockCreated() збільшує значення blocksCount при створенні нового блоку.

– Метод BlockDestroyed() зменшує значення blocksCount при знищенні блоку і перевіряє, чи досягнуто умови, коли всі блоки знищені. Якщо blocksCount менше або дорівнює 0, викликається метод Wait(), який очікує певний час (time) і потім завантажує наступний рівень.

Метод Wait() є асинхронним і використовує WaitForSeconds для затримки протягом певного часу (time). Після затримки, отримує індекс активного рівня

за допомогою SceneManager.GetActiveScene().buildIndex і завантажує наступний рівень за допомогою SceneManager.LoadScene(index + 1).

3.4.8 LostBall

Цей код відповідає за обробку втрати м'яча гравцем, включаючи зменшення кількості м'ячів, виконання певних дій при втраті останнього м'яча, відтворення звуку, відновлення м'яча і знищення об'єкта-тригера, який зіткнувся з м'ячем.

– Імпортує необхідні простори імен.

– Оголошує клас LostBall, який успадковується від MonoBehaviour.

– Оголошує змінні health, balls, ball, ballsLength і audioSource.

– У методі Awake() отримує посилання на компонент AudioSource, що прикріплений до цього об'єкта.

– У методі Start() знаходить і зберігає посилання на об'єкт HeartsBar.

– У методі Update() знаходить і зберігає масив усіх об'єктів типу Ball і оновлює значення ballsLength залежно від кількості м'ячів.

– У методі OnTriggerEnter2D() спрацьовує, коли об'єкт зіткнувся з коллайдером. Перевіряє, чи зіткнений об'єкт має тег "ball". Якщо так, зменшує ballsLength і виконує деякі додаткові дії, такі як відтворення звуку, виклик методу MinusHeart() з health, виклик методу BallRestart(), знищення м'яча (якщо ballsLength більше 0) тощо. Якщо зіткнений об'єкт не має тегу "ball", то його також знищує.

– Метод BallRestart() знаходить і зберігає посилання на об'єкт Ball, що прикріплений до сцени, і викликає його метод Restart().

– Метод Wait() є асинхронним і використовує WaitForSeconds для затримки протягом певного часу (delayInSecs). Після затримки, знищує компонент Ball з цього об'єкта.

3.4.9 Player

Цей код відповідає за рух гравця вгору і вниз за допомогою миші або автоматично, в залежності від встановленого режиму autoplay. Також він включає функціонал зміни масштабу гравця.

– Імпортує необхідні простори імен.

– Оголошує клас Player, який успадковується від MonoBehaviour.

– Оголошує змінні yPosition, xMax, ball і autoplay.

– У методі Start() зберігає початкову позицію по осі Y гравця.

– У методі Update() перевіряє, чи час гри не зупинений (Time.timeScale == 0f). Якщо так, повертається з методу.

– Якщо включений режим autoplay, то отримує позицію м'яча і встановлює нову позицію гравця на основі позиції м'яча. Обмежує нову позицію за допомогою Mathf.Clamp в діапазоні від -xMax до xMax.

– Якщо режим autoplay не включений, то отримує позицію миші в пікселях (Input.mousePosition), перетворює її в позицію в світі (Camera.main.ScreenToWorldPoint) і встановлює нову позицію гравця на основі позиції миші. Обмежує нову позицію за допомогою Mathf.Clamp в діапазоні від -xMax до xMax.

Метод PadScale() змінює масштаб гравця по осі Y на основі заданого значення newSize.

3.4.10 Scenes

Цей код відповідає за зміну сцен, перезавантаження поточної сцени та вихід з гри. Він також має залежність від SoundManager, який використовується для відтворення звуків.

– Імпортує необхідний простір імен.

– Оголошує клас Scenes, який успадковується від MonoBehaviour.

– Оголошує змінну sm типу SoundManager.

– У методі Start() знаходить і зберігає посилання на об'єкт типу SoundManager.

– Метод ChangeScene(string scene) викликається при зміні сцени і приймає назву сцени, на яку потрібно перейти. Використовує SceneManager.LoadScene(scene) для зміни сцени на задану. Якщо назва сцени дорівнює "Main", виводить повідомлення "hi".

– Метод RestartScene() викликається при перезапуску поточної сцени. Отримує індекс активної сцени за допомогою SceneManager.GetActiveScene().buildIndex і потім викликає SceneManager.LoadScene(index), щоб перезавантажити сцену.

– Метод ExitGame() викликається при виході з гри. Використовує Application.Quit() для закриття додатка.

3.4.11 SoundManager

Цей код відповідає за керування звуком в грі. Він має можливість вимкнути звук, відтворювати звуки за допомогою AudioClip і зберігати фоновий звук на всіх сценах.

– Імпортує необхідний простір імен.

– Оголошує клас SoundManager, який успадковується від MonoBehaviour.

– Оголошує змінну audioSource типу AudioSource.

– Оголошує змінну bgSound типу GameObject.

– У методі Awake() знаходить компонент AudioSource на цьому об'єкті і зберігає його в змінну audioSource.

– Знаходить об'єкт з назвою "Big in Japan" за допомогою GameObject.Find. Якщо об'єкт не знайдено, створює його за допомогою Instantiate(bgSound) і зберігає його в змінну bgSoundOne. Встановлює флаг DontDestroyOnLoad(bgSoundOne), щоб об'єкт не був знищений при зміні сцени.

– Метод Mute() встановлює значення true для властивості mute об'єкта audioSource, щоб вимкнути звук.

– Метод PlaySound(AudioClip sound) відтворює звук, переданий як параметр sound, за допомогою audioSource.PlayOneShot(sound).

3.5 Класи модифікаторів

3.5.1 PickUpExplode

Цей скрипт відповідає за поведінку об'єкту "PickUpExplode" у грі. При запуску сцени, він знаходить компонент "SoundManager" у сцені, який відповідає за відтворення звукових ефектів. При зіткненні з об'єктом гравця, викликається метод OnTriggerEnter2D, який перевіряє, чи об'єкт гравця має тег "Player". Якщо так, то відтворюється звуковий ефект, передаючи йому заздалегідь заданий звук "pickup".

Після відтворення звуку, викликається метод ApplyEffect(), який знаходить усі об'єкти типу "Ball" у сцені за допомогою методу FindObjectsOfType<Ball>(). Для кожного знайденого об'єкту "Ball" викликається метод ActivateExplode(), що активує вибуховий ефект.

У кінці методу OnTriggerEnter2D() об'єкт "PickUpExplode" знищується за допомогою методу Destroy(gameObject), щоб видалити його зі сцени після підбирання.

3.5.2 PickUpSpeed

Цей скрипт відповідає за поведінку об'єкту "PickUpSpeed" у грі. При запуску сцени, він знаходить компонент "SoundManager" у сцені, який відповідає за відтворення звукових ефектів. При зіткненні з об'єктом гравця, викликається метод OnTriggerEnter2D, який перевіряє, чи об'єкт гравця має тег "Player". Якщо так, то відтворюється звуковий ефект, передаючи йому заздалегідь заданий звук "pickup".

Після відтворення звуку, викликається метод ApplyEffect(), який знаходить усі об'єкти типу "Ball" у сцені за допомогою методу FindObjectsOfType<Ball>(). Для кожного знайденого об'єкту "Ball" викликається метод MultiplySpeed(), який множить поточну швидкість об'єкту на коефіцієнт швидкості, переданий в змінну speedCoeff.

У кінці методу OnTriggerEnter2D() об'єкт "PickUpSpeed" знищується за допомогою методу Destroy(gameObject), щоб видалити його зі сцени після підбирання.

3.5.3 PickUpScore

Цей скрипт відповідає за поведінку об'єкту "PickUpScore" у грі. При запуску сцени, він знаходить компонент "SoundManager" у сцені, який відповідає за відтворення звукових ефектів. При зіткненні з об'єктом гравця, викликається метод OnTriggerEnter2D, який перевіряє, чи об'єкт гравця має тег "Player". Якщо так, то відтворюється звуковий ефект, передаючи йому заздалегідь заданий звук "pickup".

Після відтворення звуку, викликається метод ApplyEffect(), який знаходить об'єкт "GameManager" у сцені за допомогою методу FindObjectOfType<GameManager>(). Далі, метод AddScore(pickUpPoints) викликається на знайденому об'єкті "GameManager" і передає йому кількість очків "pickUpPoints" для додавання до загального рахунку гравця.

У кінці методу OnTriggerEnter2D() об'єкт "PickUpScore" знищується за допомогою методу Destroy(gameObject), щоб видалити його зі сцени після підбирання.

3.5.4 PickUpPadScale

Цей скрипт відповідає за поведінку об'єкту "PickUpPadScale" у грі. При запуску сцени, він знаходить компонент "SoundManager" у сцені, який відповідає за відтворення звукових ефектів. При зіткненні з об'єктом гравця, викликається метод OnTriggerEnter2D, який перевіряє, чи об'єкт гравця має тег "Player". Якщо так, то відтворюється звуковий ефект, передаючи йому заздалегідь заданий звук "pickup".

Після відтворення звуку, викликається метод ApplyEffect(), який знаходить об'єкт "Player" у сцені за допомогою методу FindObjectOfType<Player>(). Далі, метод PadScale(newSize) викликається на знайденому об'єкті "Player" і передає йому новий розмір "newSize" для зміни розміру платформи гравця.

У кінці методу OnTriggerEnter2D() об'єкт "PickUpPadScale" знищується за допомогою методу Destroy(gameObject), щоб видалити його зі сцени після підбирання.

3.5.5 PickUpMagnet

Цей скрипт відповідає за поведінку об'єкту "PickUpMagnet" у грі. При запуску сцени, він знаходить компонент "SoundManager" у сцені, який

відповідає за відтворення звукових ефектів. При зіткненні з об'єктом гравця, викликається метод OnTriggerEnter2D, який перевіряє, чи об'єкт гравця має тег "Player". Якщо так, то відтворюється звуковий ефект, передаючи йому заздалегідь заданий звук "pickup".

Після відтворення звуку, викликається метод ApplyEffect(), який знаходить усі об'єкти "Ball" у сцені за допомогою методу FindObjectsOfType<Ball>(). Далі, в циклі foreach проходиться по кожному знайденому об'єкту "Ball" і викликається метод MagnetActivate(), що активує магнітну властивість для кожного з них.

У кінці методу OnTriggerEnter2D() об'єкт "PickUpMagnet" знищується за допомогою методу Destroy(gameObject), щоб видалити його зі сцени після підбирання.

3.5.6 PickUpHealth

Цей скрипт відповідає за поведінку об'єкту "PickUpHealth" у грі. При запуску сцени, він знаходить компонент "SoundManager" у сцені, який відповідає за відтворення звукових ефектів. При зіткненні з об'єктом гравця, викликається метод OnTriggerEnter2D, який перевіряє, чи об'єкт гравця має тег "Player". Якщо так, то відтворюється звуковий ефект, передаючи йому заздалегідь заданий звук "pickup".

Після відтворення звуку, викликається метод ApplyEffect(), який знаходить об'єкт "HeartsBar" у сцені за допомогою методу FindObjectOfType<HeartsBar>(). Після цього викликається метод AddRemoveHeart(addHeart), який додає або видаляє серце з "HeartsBar", залежно від значення змінної addHeart.

У кінці методу OnTriggerEnter2D() об'єкт "PickUpHealth" знищується за допомогою методу Destroy(gameObject), щоб видалити його зі сцени після підбирання.

3.5.8 PickUpDuplicateBall

Цей скрипт відповідає за поведінку об'єкту "PickUpDuplicateBall" у грі. При запуску сцени, він знаходить компонент "SoundManager" у сцені, який відповідає за відтворення звукових ефектів. При зіткненні з об'єктом гравця, викликається метод OnTriggerEnter2D, який перевіряє, чи об'єкт гравця має тег "Player". Якщо так, то відтворюється звуковий ефект, передаючи йому заздалегідь заданий звук "pickup".

Після відтворення звуку, викликається метод ApplyEffect(), який знаходить усі об'єкти "Ball" у сцені за допомогою методу FindObjectsOfType<Ball>(). Потім для кожного об'єкта "Ball" викликається метод DuplicateBall(), який дублює кулю, створюючи новий об'єкт.

У кінці методу OnTriggerEnter2D() об'єкт "PickUpDuplicateBall" знищується за допомогою методу Destroy(gameObject), щоб видалити його зі сцени після підбирання.

3.5.7 PickUpBallScale

Цей скрипт відповідає за ефект підбирання об'єкту "PickUpBallScale" у грі. При запуску сцени, він шукає компонент "SoundManager" у сцені, який відповідає за відтворення звукових ефектів.

У методі OnTriggerEnter2D перевіряється, чи об'єкт гравця має тег "Player". Якщо так, то відтворюється звуковий ефект, заданий заздалегідь у змінній "pickup", за допомогою методу PlaySound з об'єкту SoundManager.

Далі, метод ApplyEffect знаходить усі об'єкти "Ball" у сцені за допомогою методу FindObjectsOfType<Ball>(). Для кожного об'єкту "Ball" викликається метод BallScale(), який змінює масштаб кулі, встановлюючи новий розмір.

У кінці методу OnTriggerEnter2D(), об'єкт "PickUpBallScale" знищується за допомогою методу Destroy(gameObject), щоб видалити його зі сцени після підбирання.

3.6 Звукове оформлення

У розділі розробки звуку у грі Ball Break вирішено було зосередитись на створенні ефективної системи звуків, яка покращує іммерсію та геймплей гравця. було використано клас SoundManager для керування всіма звуковими елементами гри. Цей клас включає методи для відтворення різних звуків, таких як зіткнення м'яча з блоком або звук втрати життя.. Крім того, використано клас AudioSource для відтворення звуків і призначено його різним об'єктам в грі, таким як м'яч і платформа(рис.3.9). Мета полягає в створенні приємного та реалістичного звукового досвіду, який доповнює геймплей та робить гру більш захоплюючою для гравця. за допомогою редактора 8 та 16 bit звуків на сайті sfxr.me були створені такі звуки для відбивання кулі, знищення об'єктів, та отримання різних бонусів. (рис. 3.10)

Рисунок 3.9— Звуки у грі

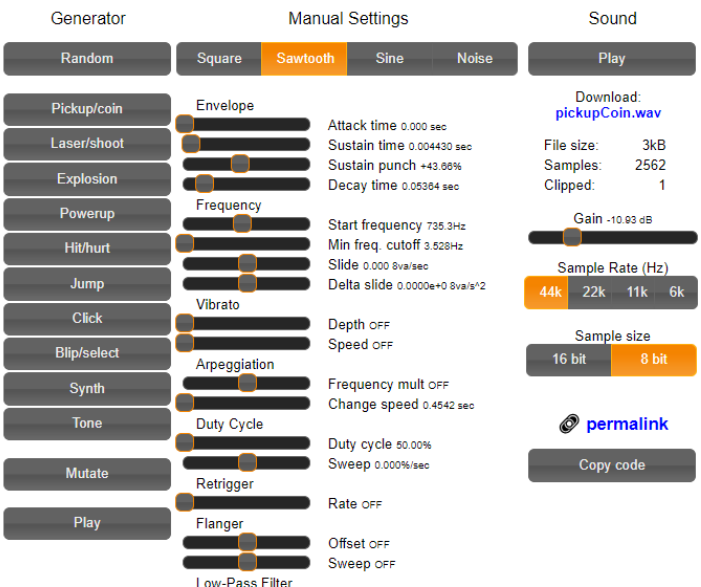

Рисунок 3.10 —Редаткор 8 та 16 bit звуків sfxr.me

ВИСНОВКИ

У цій кваліфікаційній роботі було розроблено аркадну гру Brick breaker.

Також проаналізовано середовища для розробки двовимірних ігор та розроблено двовимірну адаптацію аркадної гри «Arkanoid» за допомогою фреймворка Unity. У аналізі ігрових рушіїв, було розглянуто три найпопулярниших фреймворки такі як: Unity, Unreal Engine та GameMaster. Під час розробки були розглянуті ключові концепції та інструменти, які використовуються при створенні ігор на платформі Unity, такі як створення ігрових об'єктів, прикріплення компонентів, конфігурація фізики, використання анімацій, звукових ефектів, візуалізаційних ефектів та написання скриптів на мові C#.

Мета роботи: Створити гру за допомогою рушія Unity для декількох ігрових платформ.

Методи дослідження:

- Аналіз літературних джерел,
- розробка прототипу,
- опитування та збір даних,
- порівняльний аналіз.

Результати роботи:

✓ Розроблена гра на Unity

✓ Написані скрипти для створення об’єктів та їх знищення

✓ Проект протестований та налагодження проведено.

ПЕРЕЛІК ПОСИЛАНЬ

1. Koster R. A Theory of Fun for Game Design. Paraglyph Press, 2004, 256 с.
2. Norton T. Learning C# by Developing Games with Unity 3D Beginner's Guide. Packt Publishing, London UK 2013. 292 с.
3. Jackson S. Mastering Unity 2D Game Development. Packt Publishing – ebooks Account, Washington DC, USA, 2014. 500 с.
4. Jon Manning, Tim Nugent, Paris Buttfield-Addison.Unity 2D Game Development Cookbook 1st Edition. Packt Publishing, Colorado 2019. 256 с
5. Softermii URL: https://www.softermii.com/blog/top-ides-for-software-development (дата звернення: 28.04.2023)
6. sfxr.me URL: https://sfxr.me/ (дата звернення: 26.05.2023)
7. kenney.nl URL: https://kenney.nl/ (дата звернення : 24.05.2023)
8. Mike Geig, Unity Game Development in 24 Hours 1st edition : Sams Publishing, USA 2019. 434 с.
9. Matt McIntyre, The 20 Most Popular Video Games of All gamedesigning (дата звернення: 23.04.2023) URl: https://www.gamedesigning.org/career/video-game-engines/
10. Metamersiv URL: https://www.metaverseme.io/blog/evolution-of-gaming (дата звернення: 22.04.2023)
11. MattMcIntyre,03.04.2023,GaminggorillaURL:https://gaminggorilla.com/most-popular-video-games/ (дата звернення:23.04.2023)
12. JeffDrake,17.03.2023,thegamer.com URL: https://www.thegamer.com/unity-game-engine-great-games/ (дата звернення:23.04.2023)
13. Livescience URL:https://www.livescience.com/56481-strange-history-of-tetris.html (дата звернення: 23.04.2023)
14. pxfuel.com URL: https://www.pxfuel.com/en/desktop-wallpaper-snwhm (дата звернення:23.04.2023)

Додатки

Додаток А (ball)

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

public class Ball : MonoBehaviour
{
    [Header("Ball's characteristics")]
    public Rigidbody2D rb;
    public float initialSpeed;
    public float currentSpeed;
    public float ballSize;
    public PickUpSpeed bSpeed;
    public PickUpBallScale bScale;

    [Header("Ball's position")]
    public float yPosition;
    public float xDelta;
    public float yDelta;
    bool isStarted;
    bool isMagnetActive;

    [Header("Explode")]
    public float explodeRadius;
    public bool isExplosive;
    public GameObject explodeEffect;

    [Header("Pad")]
    public Player pad;
    public TrailRenderer trailRenderer;

    [Header("Sounds")]
    AudioSource audioSource;
    //public AudioClip explodeSoundBall;
    public AudioClip ballHitSmth;
    private void Awake()
    {
        audioSource = GetComponent<AudioSource>();
    }
    private void Start()
    {
        ballSize = 1f;
        currentSpeed = initialSpeed;
        yPosition = -3.93f;
        isExplosive = false;
    }
    private void Update()
    {
        if (isStarted)
        {
            rb.velocity = rb.velocity.normalized * currentSpeed;
        }
        else
```

```
        {
            transform.position = new Vector3(pad.transform.position.x + xDelta,
yPosition, 0);
            if (Input.GetMouseButtonDown(0))
            {
                StartBall();
            }
        }
    }
    public void ActivateExplode()
    {
        explodeEffect.SetActive(true);
        audioSource.Play();
        isExplosive = true;
    }
    public void MultiplySpeed(float speedCoeff)
    {
        currentSpeed *= speedCoeff;
    }
    public void MagnetActivate()
    {
        isMagnetActive = true;
    }
    public void BallScale(float ballScale)
    {
        isStarted = true;
        StartBall();
        ballSize = ballScale;
        transform.localScale = new Vector3(ballSize, ballSize, ballSize);
        trailRenderer.startWidth = ballSize / 2;
    }
    public void DuplicateBall()
    {
        Ball originalBall = this;
        Ball newBall = Instantiate(originalBall);
        newBall.StartBall();
    }
    void StartBall()
    {
        float randomX = Random.Range(0, 0);
        Vector2 direction = new Vector2(randomX, 4f);
        Vector2 force = direction.normalized * currentSpeed;
        rb.velocity = force;
        isStarted = true;
    }
    private void OnDrawGizmos()
    {
        Gizmos.color = Color.red;
        Gizmos.DrawWireSphere(transform.position, explodeRadius);
    }
    public void Restart()
    {
        currentSpeed = initialSpeed;
        isMagnetActive = false;
        isExplosive = false;
        explodeEffect.SetActive(false);
        ballSize = 1f;
        trailRenderer.startWidth = 0.5f;
        transform.localScale = new Vector3(ballSize, ballSize, ballSize);
        isStarted = false;
        transform.position = new Vector3(pad.transform.position.x, yPosition, 0);
        rb.velocity = rb.velocity.normalized * currentSpeed;
    }
    void Explode()
    {
```

```
        int layerMask = LayerMask.GetMask("Block");
        Collider2D[] colliders = Physics2D.OverlapCircleAll(transform.position,
explodeRadius, layerMask);
        foreach (Collider2D col in colliders)
        {
            Blocks block = col.GetComponent<Blocks>();
            if (block == null)
            {
                Destroy(gameObject);
            }
            else
            {
                block.DestroyBlock();
            }
        }
    }
    private void OnCollisionEnter2D(Collision2D collision)
    {
        audioSource.PlayOneShot(ballHitSmth);
        if (isMagnetActive && collision.gameObject.CompareTag("Player"))
        {
            //audioSource.Play();
            xDelta = transform.position.x - pad.transform.position.x;
            float newYPosition = transform.position.y;
            isStarted = false;
            rb.velocity = Vector2.zero;
            transform.position = new Vector3(pad.transform.position.x + xDelta,
newYPosition, 0);

        }
        if (isExplosive && collision.gameObject.CompareTag("block"))
        {
            Explode();
        }
    }

}
```

Додаток Б (blocks)

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
public class Blocks : MonoBehaviour
{
    [Header("Block's points")]
    public int points;
    GameManager gamemanager;
    LevelChanger lc;

    [Header("Block's lives")]
    public int lifeBlock;
    public bool invisible;

    [Header("Block's sprites")]
    public Sprite[] blockDamaged;
    public SpriteRenderer sRender;

    [Header("Pick-ups and effects")]
    public GameObject particlePrefab;
    public GameObject[] pickUps;

    [Header("Sounds")]
    AudioSource audioSource;
```

```
    public AudioClip blockDestroyedSound;
    SoundManager sM;
    private void Awake()
    {
        audioSource = GetComponent<AudioSource>();
        sM = FindObjectOfType<SoundManager>();
        gamemanager = FindObjectOfType<GameManager>();
        sRender = GetComponent<SpriteRenderer>();
        lc = FindObjectOfType<LevelChanger>();
    }
    private void Start()
    {
        lc.BlockCreated();
        if (invisible)
        {
            sRender.enabled = false;
        }
    }
    private void OnCollisionEnter2D(Collision2D collision)
    {
        audioSource.Play();
        if (invisible)
        {
            sRender.enabled = true;
            invisible = false;
            return;
        }
        lifeBlock--;
        for (int i = lifeBlock; i > 0; i--)
        {
            if (lifeBlock == i)
            {
                sRender.sprite = blockDamaged[i - 1];
            }
        }

        if (lifeBlock <= 0)
        {
            sM.PlaySound(blockDestroyedSound);
            DestroyBlock();
        }
    }
    public void DestroyBlock()
    {
        lc.BlockDestroyed();
        gamemanager.AddScore(points);
        Destroy(gameObject);
        Instantiate(particlePrefab, transform.position, Quaternion.identity);
        PickUpChance();
    }
    public void PickUpChance()
    {
        int randomValue = Random.Range(0, 30);
        for (int i = 0; i < pickUps.Length; i++)
        {
            if (randomValue == i)
            {
                Instantiate(pickUps[i], transform.position, Quaternion.identity);
            }
        }
    }
}
```

Додаток В (GameDataSO)

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
[CreateAssetMenu]
public class GameDataSO : ScriptableObject
{
    public int s;
    public int h;

}
```

Додаток Г (GameManager)

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

public class GameManager : MonoBehaviour
{
    public Text scoreText;
    public int score;
    public bool isPaused;
    public GameObject pausePanel;
    public int bestResult;
    private void Awake()
    {
        GameManager[] gameManagers = FindObjectsOfType<GameManager>();
        for (int i = 0; i < gameManagers.Length; i++)
        {
            if (gameManagers[i].gameObject != gameObject)
            {
                //Destroy(gameObject);
                gameObject.SetActive(false);
                break;
            }
        }
        bestResult = PlayerPrefs.GetInt("BestScore");
    }

    private void Update()
    {
        scoreText.text = "Score: " + score.ToString();
        DontDestroyOnLoad(gameObject);
        if (Input.GetKeyDown(KeyCode.Escape))
        {
            if (isPaused)
            {
                pausePanel.SetActive(false);
                Time.timeScale = 1f;
                isPaused = false;
            }
            else
            {
                pausePanel.SetActive(true);
                Time.timeScale = 0f;
                isPaused = true;
            }
        }
    }
    public void AddScore(int addScore)
    {
```

```
        score += addScore;
        scoreText.text = "Score: " + score.ToString();
        PlayerPrefs.SetInt("BestScore", score);
    }

}
```

Додаток Ґ (GameOver)

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

public class GameOver : MonoBehaviour
{
    public Text totalScore;
    public Text bestScore;
    GameManager gm;
    HeartsBar hearts;
    private void Awake()
    {
        gm = FindObjectOfType<GameManager>();
        hearts = FindObjectOfType<HeartsBar>();
    }
    void Start()
    {
        totalScore.text = "Total score: " + gm.score.ToString();
        bestScore.text = "Best score: " + gm.bestResult.ToString();
        hearts.gameObject.SetActive(false);
        gm.scoreText.enabled = false;
    }

}
```

Додаток Д (HeartsBar)

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

public class HeartsBar : MonoBehaviour
{
    [Header("Hearts images")]
    public Image[] hearts;
    public Sprite fullHeart;
    public Sprite emptyHeart;

    [Header("Lives")]
    public int health; //- здоровье = заполненные сердечки
    public int heartsNumber; //количество ВИДИМЫХ сердечек на канвасе (заполненных и
незаполненных)
    [SerializeField] private int startHeartsNumber;

    [Header("Ball")]
    public Ball ball;

    [Header("Game Over")]
    public GameObject gameOverPanel;
    GameManager gm;
    public Text totalScore;
    public bool isDead;

    private void Start()
    {
```

```
        gm = FindObjectOfType<GameManager>();
        for (int j = 0; j < hearts.Length; j++) //т.к. всего сердец 5, но видно только 3
в начале игры, надо их сделать НЕвидимыми все
        {
            hearts[j].enabled = false;
        }
        HeartsStart();
    }

    public void MinusHeart()
    {
        health--;
        for (int i = 0; i < heartsNumber; i++)
        {
            if (i < health)
            {
                hearts[i].sprite = fullHeart;
            }
            else
            {
                hearts[i].sprite = emptyHeart;
            }
            if (i < heartsNumber)
            {
                hearts[i].enabled = true;
            }
            else
            {
                hearts[i].enabled = false;
            }
        }
        GameOver();
    }
    public void AddRemoveHeart(int addHeart)
    {
        if (addHeart > 0)
        {
            if (health < heartsNumber)
            {
                hearts[heartsNumber - 1].sprite = fullHeart;
                health++;
            }
            else if (health == heartsNumber)
            {
                heartsNumber++;
                hearts[heartsNumber - 1].enabled = true;
                health++;
            }
        }
        else
        {
            MinusHeart();
        }

    }
    public void HeartsStart()
    {
        isDead = false;
        gm.isPaused = false;
        gm.score = 0;
        Time.timeScale = 1f;
        heartsNumber = startHeartsNumber;
        health = startHeartsNumber;
        for (int k = 0; k < heartsNumber; k++) //а после этого сделать видимыми только
нужное количество (указывается в инспекторе)
```

```
        {
            hearts[k].sprite = fullHeart;
            hearts[k].enabled = true;
            hearts[k + 1].enabled = false;
            hearts[k + 2].enabled = false;
        }
    }

    public void GameOver()
    {
        if (health <= 0)
        {
            isDead = true;
            gameOverPanel.SetActive(true);
            Time.timeScale = 0f;
            totalScore.text = "Total score: " + gm.score.ToString();
        }
    }
}
```

Додаток Е (LevelChanger)

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

public class LevelChanger : MonoBehaviour
{
    public int blocksCount;
    GameManager gm;
    HeartsBar heartsBar;

    private void Start()
    {
        gm = FindObjectOfType<GameManager>();
        heartsBar = FindObjectOfType<HeartsBar>();

    }
    public void BlockCreated()
    {
        blocksCount++;
    }

    public void BlockDestroyed()
    {
        blocksCount--;
        if (blocksCount <= 0)
        {

            //PlayerPrefs.SetInt("CurrentScore", gm.score);
            //PlayerPrefs.SetInt("Hearts", heartsBar.health);
            StartCoroutine(Wait(0.5f));
        }
    }
    IEnumerator Wait(float time)
    {
        yield return new WaitForSeconds(time);
        int index = SceneManager.GetActiveScene().buildIndex;
        SceneManager.LoadScene(index + 1);
    }
}
```

Додаток Є (PickUpBallScale)

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

public class PickUpBallScale : MonoBehaviour
{
    public float ballScale;
    SoundManager sM;
    public AudioClip pickup;
    private void Awake()
    {
        sM = FindObjectOfType<SoundManager>();
    }
    void ApplyEffect()
    {
        Ball[] balls = FindObjectsOfType<Ball>();
        foreach (Ball bal in balls)
        {
            bal.BallScale(ballScale);
        }
    }
    private void OnTriggerEnter2D(Collider2D collision)
    {
        if (collision.gameObject.CompareTag("Player"))
        {
            sM.PlaySound(pickup);
            ApplyEffect();
            Destroy(gameObject);
        }
    }
}
```

Додаток Ж (PickUpDuplicateBall)

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

public class PickUpDuplicateBall : MonoBehaviour
{
    SoundManager sM;
    public AudioClip pickup;
    private void Awake()
    {
        sM = FindObjectOfType<SoundManager>();
    }
    void ApplyEffect()
    {
        Ball[] balls = FindObjectsOfType<Ball>();
        foreach (Ball bal in balls)
        {
            bal.DuplicateBall();
        }
    }
    private void OnTriggerEnter2D(Collider2D collision)
    {
        if (collision.gameObject.CompareTag("Player"))
        {
            sM.PlaySound(pickup);
            ApplyEffect();
            Destroy(gameObject);
        }
    }
```

```
}
```

Додаток З (PickUpExplode)

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

public class PickUpExplode : MonoBehaviour
{
    SoundManager sM;
    public AudioClip pickup;
    private void Awake()
    {
        sM = FindObjectOfType<SoundManager>();
    }
    void ApplyEffect()
    {
        Ball[] balls = FindObjectsOfType<Ball>();
        foreach (Ball bal in balls)
        {
            bal.ActivateExplode();
        }
    }
    private void OnTriggerEnter2D(Collider2D collision)
    {
        if (collision.gameObject.CompareTag("Player"))
        {
            sM.PlaySound(pickup);
            ApplyEffect();
            Destroy(gameObject);
        }
    }
}
```

Додаток И (PickUpHealth)

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

public class PickUpHealth : MonoBehaviour
{
    public int addHeart;
    SoundManager sM;
    public AudioClip pickup;
    private void Awake()
    {
        sM = FindObjectOfType<SoundManager>();
    }
    void ApplyEffect()
    {
        HeartsBar hearts = FindObjectOfType<HeartsBar>();
        hearts.AddRemoveHeart(addHeart);
    }
    private void OnTriggerEnter2D(Collider2D collision)
    {
        if (collision.gameObject.CompareTag("Player"))
        {
            sM.PlaySound(pickup);
            ApplyEffect();
            Destroy(gameObject);
        }
    }
}
```

Додаток Й (PickUpManagment)

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

public class PickUpMagnet : MonoBehaviour
{
    SoundManager sM;
    public AudioClip pickup;
    private void Awake()
    {
        sM = FindObjectOfType<SoundManager>();
    }
    void ApplyEffect()
    {
        Ball[] balls = FindObjectsOfType<Ball>();
        foreach (Ball bal in balls)
        {
            bal.MagnetActivate();
        }
    }
    private void OnTriggerEnter2D(Collider2D collision)
    {
        if (collision.gameObject.CompareTag("Player"))
        {
            sM.PlaySound(pickup);
            ApplyEffect();
            Destroy(gameObject);
        }
    }
}
```

Додаток К (PickUpPadScale)

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

public class PickUpPadScale : MonoBehaviour
{
    public float newSize;
    SoundManager sM;
    public AudioClip pickup;
    private void Awake()
    {
        sM = FindObjectOfType<SoundManager>();
    }
    void ApplyEffect()
    {
        Player pad  = FindObjectOfType<Player>();
        pad.PadScale(newSize);
    }
    private void OnTriggerEnter2D(Collider2D collision)
    {
        if (collision.gameObject.CompareTag("Player"))
        {
            sM.PlaySound(pickup);
            ApplyEffect();
            Destroy(gameObject);
        }
    }
}
```

Додаток Л (PickUpScore)

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
public class PickUpScore : MonoBehaviour
{
    public int pickUpPoints;
    SoundManager sM;
    public AudioClip pickup;
    private void Awake()
    {
        sM = FindObjectOfType<SoundManager>();
    }
    void ApplyEffect()
    {
        GameManager gM = FindObjectOfType<GameManager>();
        gM.AddScore(pickUpPoints);
    }
    private void OnTriggerEnter2D(Collider2D collision)
    {
        if (collision.gameObject.CompareTag("Player"))
        {
            sM.PlaySound(pickup);
            ApplyEffect();
            Destroy(gameObject);
        }
    }
}
```

Додаток М (PickUpSpeed)

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

public class PickUpSpeed : MonoBehaviour
{
    public float speedCoeff;
    SoundManager sM;
    public AudioClip pickup;
    private void Awake()
    {
        sM = FindObjectOfType<SoundManager>();
    }
    void ApplyEffect()
    {
        Ball[] balls = FindObjectsOfType<Ball>();
        foreach (Ball bal in balls)
        {
            bal.MultiplySpeed(speedCoeff);
        }
    }
    private void OnTriggerEnter2D(Collider2D collision)
    {
        if (collision.gameObject.CompareTag("Player"))
        {
            sM.PlaySound(pickup);
            ApplyEffect();
            Destroy(gameObject);
        }
    }
}
```

Додаток Н (Player)

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

public class Player : MonoBehaviour
{
    float yPosition;
    public float xMax;
    [Tooltip("Ссылка на мяч")]
    public Ball ball;

    [Tooltip("Режим автоматической игры для проверки")]
    public bool autoplay;

    void Start()
    {
        yPosition = transform.position.y;
    }

    void Update()
    {
        if (Time.timeScale == 0f)
        {
            return;
        }

        if (autoplay)
        {
            Vector3 ballPos = ball.transform.position;
            Vector3 newPadPos = new Vector3(ballPos.x, yPosition, 0);
            newPadPos.x = Mathf.Clamp(newPadPos.x, -xMax, xMax);
            transform.position = newPadPos;
        }
        else
        {
            Vector3 mousePixelPoint = Input.mousePosition;
            Vector3 mouseWorldPosition = Camera.main.ScreenToWorldPoint(mousePixelPoint);
            Vector3 padNewPosition = new Vector3(mouseWorldPosition.x, yPosition, 0);
            padNewPosition.x = Mathf.Clamp(padNewPosition.x, -xMax, xMax);
            transform.position = padNewPosition;
        }
    }
    public void PadScale(float newSize)
    {
        transform.localScale = new Vector3(1, transform.localScale.y * newSize, 1);
    }

}
```

Додаток О (Scenes)

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

public class Scenes : MonoBehaviour
{
    SoundManager sm;
    private void Start()
    {
```

```
        sm = FindObjectOfType<SoundManager>();
    }
    public void ChangeScene(string scene)
    {
        SceneManager.LoadScene(scene);
        if (scene == "Main")
        {
            print("hi");
        }
    }
    public void RestartScene()
    {
        int index = SceneManager.GetActiveScene().buildIndex;
        SceneManager.LoadScene(index);
    }
    public void ExitGame()
    {
        Application.Quit();
    }
}
```

Додаток П (SoundManager)

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

public class SoundManager : MonoBehaviour
{
    AudioSource audioSource;
    public GameObject bgSound;
    private void Awake()
    {

        audioSource = GetComponent<AudioSource>();
        var bgSoundOne = GameObject.Find("Big in Japan");
        if (bgSoundOne == null)
        {
            bgSoundOne = Instantiate(bgSound);
            DontDestroyOnLoad(bgSoundOne);
        }

    }
    public void Mute()
    {
        audioSource.mute = true;
    }
    public void PlaySound(AudioClip sound)
    {
        audioSource.PlayOneShot(sound);
    }

}
```