

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи освітнього ступеня «магістр»
за спеціальністю 121 «Інженерія програмного забезпечення»
(освітня програма «Інженерія програмного забезпечення»)

на тему:

**«Розробка гри "SortyBall" з використанням штучного
інтелекту»**

Виконав студент групи ІПЗм-22-1
БОГДАНЕЦЬ Віталій Миколайович

Керівник роботи:
КРАВЧЕНКО Світлана Миколаївна

Рецензент:
ЛОКТИКОВА Тамара Миколаївна

Житомир – 2024

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

«ЗАТВЕРДЖУЮ»

Зав. кафедри інженерії
програмного забезпечення

Тетяна ВАКАЛЮК

«02» травня 2024 р.

ЗАВДАННЯ
на кваліфікаційну роботу

Здобувач вищої освіти: **БОГДАНЕЦЬ Віталій Миколайович**

Керівник роботи: **КРАВЧЕНКО Світлана Миколаївна**

Тема роботи: **«Розробка гри "SortyBall" з використанням штучного інтелекту»**,

затверджена Наказом закладу вищої освіти від **«02» травня 2024 р. 174/с**

Вихідні дані для роботи: **Предметом дослідження є розробка нейромережі, створення 2D моделей та їх анімація анімація, реалізація власного програмного забезпечення. Об'єктом дослідження є історія жанру аркада, казуальна; алгоритми та технології для розробки комп'ютерних ігор; моделі нейромереж.**

Консультанти з магістерської кваліфікаційної роботи із зазначенням розділів, що їх стосуються:

Розділ	Консультант	Завдання видав	Завдання прийняв
1	Кравченко С.М.	02.05.2024р.	02.05.2024р.
2	Кравченко С.М.	02.05.2024р.	02.05.2024р.
3	Кравченко С.М.	02.05.2024р.	02.05.2024р.

РЕФЕРАТ

Кваліфікаційна робота передбачає науковий дослід та розробку гри з застосуванням технологій, моделей та методів, які в процесі реалізації будуть обгрунтовані правильністю вибору.

Пояснювальна записка до кваліфікаційної роботи “Розробка гри "SortyBall" з використанням штучного інтелекту ” складається із вступу, трьох розділів, загальних висновків, списку використаних джерел і містить 90 сторінки тексту, 45 рисунки. Список використаних джерел містить 14 найменувань.

Під час розробки проекту перевірено та проведено дослідження інших ігрових розробок в жанрі Casual та доведена правильність вибору на реалізації. Було спроектовано структуру програмної частини, наведені всі сцени роботи додатку, було зроблено повний графічний контент, зроблена програмна частина для всіх елементів гри, проведено тестування ігрової частини проекту.

КЛЮЧОВІ СЛОВА: ГРА, ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ, ШТУЧНИЙ ІНТЕЛЕКТ, АРКАДА, КАЗУАЛЬНА, ІГРОВИЙ РУШІЙ, UNITY 2D UNITY 3D, НЕЙРОМЕРЕЖА.

ABSTRACT

The qualification work involves scientific research and development of the game using technologies, models and methods, which in the process of implementation will be justified by the correctness of the choice.

The explanatory note to the qualification work "Development of the game "SortyBall" using artificial intelligence" consists of an introduction, three sections, general conclusions, a list of sources used and contains 90 pages of text, 45 figures. The list of used sources contains 14 items.

During the development of the project, other game developments in the Casual genre were checked and researched, and the correctness of the choice was proven for implementation. The structure of the software part was designed, all the scenes of the application were shown, the complete graphic content was made, the software part was made for all game elements, and the game part of the project was tested.

KEYWORDS: GAME, SOFTWARE, ARTIFICIAL INTELLIGENCE, ARCADE, CASUAL, GAME ENGINE, UNITY 2D UNITY 3D, NEURAL NETWORK.

ЗМІСТ

РЕФЕРАТ	3
ВСТУП	7
РОЗДІЛ 1. ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ	9
1.1. Нейромережі та їх роль в ігровому світі.....	9
1.2. Опис існуючого аналогу програмного продукту	17
1.2.1. Аналіз інструментальних засобів графіки.....	30
1.3. Задачі для виконання	31
1.4. Обґрунтування вибору платформи для розробки додатку.....	33
Висновки до першого розділу	38
РОЗДІЛ 2. ПРОЕКТУВАННЯ ТА РОЗРОБКА ВЕБ-ДОДАТКУ	40
2.1. Аналіз реалізації системи використання продукту та вихідних даних.....	40
2.1.1. Діаграма використання продукту користувачем.....	40
2.1.2. Діаграми класів	41
2.2. Представлення дизайн документу	47
2.2.1. Розробка та геймплей гри	47
2.2.2. Сцени та їх вигляд	48
2.2.3 Геймплей користувачів, текстури арени, підготовка графічних елементів.	49
2.2.4 Управління персонажем та аналіз його функцій	50
2.3. Процес реалізація програмного коду.....	50
Висновки до другого розділу	56
РОЗДІЛ 3. ПОРЯДОК РОБОТИ ТА ТЕСТУВАННЯ ПРОГРАМНОГО КОМПЛЕКСУ	57
3.1. Покрокова робота та інтерфейс додатку.	57
3.2. Тестування роботи програми	63
Висновки до третього розділу.....	64
ВИСНОВКИ	65
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	67
ДОДАТКИ.....	68

ВСТУП

Актуальність теми. Актуальність теми розробки ігор що з розвитком інформаційних технологій необхідно створювати нові програмні продукти, що будуть розвивати ігровий ринок та індустрію розваг в цілому. Залежно від категорії гри вона(тобій повчальна, розважальна на мислення і тд), гра буде заходити людям різних вікових категорій.

На сьогоднішній день є багато технічних можливостей для створення багатоплатформених ігор для різних ігрових девайсів, що дозволяє запускати програмний продукт у сегменті ринку, що росте і розвивається найшвидше у світі. Також важливим сегментом в розвитку продукту є економічна складова. Навіть при невеликих інвестиціях розробники будуть отримувати хороший прибуток від успішно проданих проектів. Це надихає талановитих людей доєднуватись до світу ігор, щоб досягти в ній та заробити собі на фінансову незалежність.

Мета дослідження. Створення кросплатформеної мультиплеєрна 2D гри на Unity 3D з застосуванням штучного інтелекту з елементами аркади та використанням нейромережі, вдосконалення знань розробки та набуття нових навичок, реалізація анімації, 2D-графіки, написання програмного коду , здобуття знань в нейромережі.

Для досягнення даної мети на потрібно буде такі завдання:

- дослідити ринок ігрової індустрії;
- дати характеристику основним програмним засобам для створення ігор;
- проаналізувати основні етапи створення комп'ютерної гри;
- на основі даних аналізу, розробити комп'ютерну гру.
- Реалізація мультиплеєра;
- Тестування ПЗ.

Об'єктом дослідження є технології та методи розробки кросплатформового ігрового додатку з використання Unity3D.

Предметом дослідження є концепція технології створення гри , створення

3D моделей персонажів на структури карти, їх анімація та розроблення нейромережі і створення головного меню даної гри.

Методи аналізу. Концепція технології створення гри, реалізація можливості мультиплеєра, та опис розроблених ігрових механік для можливості більш активної взаємодії гравця з створеним продуктом та функціями які були описані в ході розробки.

Інформаційна база дослідження літературні джерела з розробки ПЗ, доповіді результатів експериментів нейромережі та її реалізації.

Наукова новизна дослід дає змогу поєднання декількох жанрів ігор націлених на широку аудиторію, також реалізування нейромережі.

РОЗДІЛ 1. ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1. Нейромережі та їх роль в ігровому світі

Ну насамперед нейромережа – це один із напрямків штучного інтелекту, мета якого змодельовати аналітичні механізми, які здійснюються людським мозком. Вони складаються з вузлів, які пов’язані між собою та називаються **нейронами**. Завдання які вирішує типова нейромережа – кваліфікація, передбачення та розпізнання.

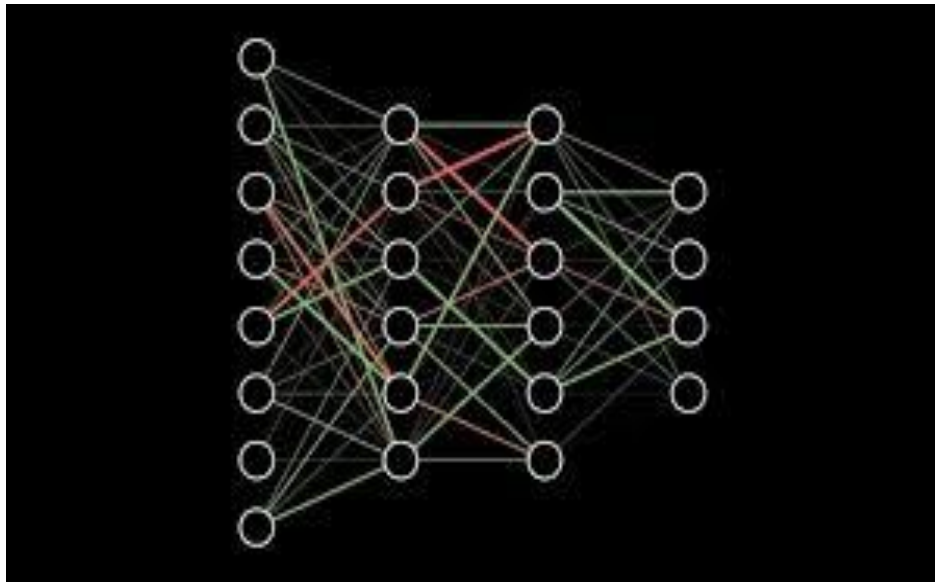


Рис. 1.1 – Приклад сітки нейромережі

Існує два основних типи навчання нейронних мереж:

1. Отримання правильної інформації: це вид отримання правильної інформації отриманий від потоку даних інформації.

2. Отримування інформації без правильних варіантів: це вид інформації який змушує нейромережу вчитися та шукати рішення по проведеному аналізу.

Роль нейромереж виходить за рамки надання “розумних” персонажів для взаємодії, наприклад, вони можуть брати участь в генерації контенту, дизайні локацій, створенні анімацій і персонажів, таким чином забезпечуючи унікальний досвід для кожного гравця і кожного проходження.

Принцип роботи нейромережі можна порівняти з нервовою системою людини, яка самонавчається за рахунок накопиченого досвіду і використовує

його для того, щоб не робити помилок.



Рис.1.2 - Аналіз користувача нейромережею

Нейромережі здійснюють інтелектуальний аналіз кожного користувача. Набір алгоритмів штучного інтелекту автоматично формує шаблони на основі зібраних даних і застосовує їх до гравців.

Штучні нейронні мережі (ШНМ) являють собою обчислювальні системи, створені з використанням вивчення принципів роботи людського мозку. Вони сконструйовані із взаємопов'язаних штучних нейронів, які функціонують разом для обробки інформації. ІНС здатні розв'язувати складні та нетривіальні завдання.

Принцип дії нейронних мереж:

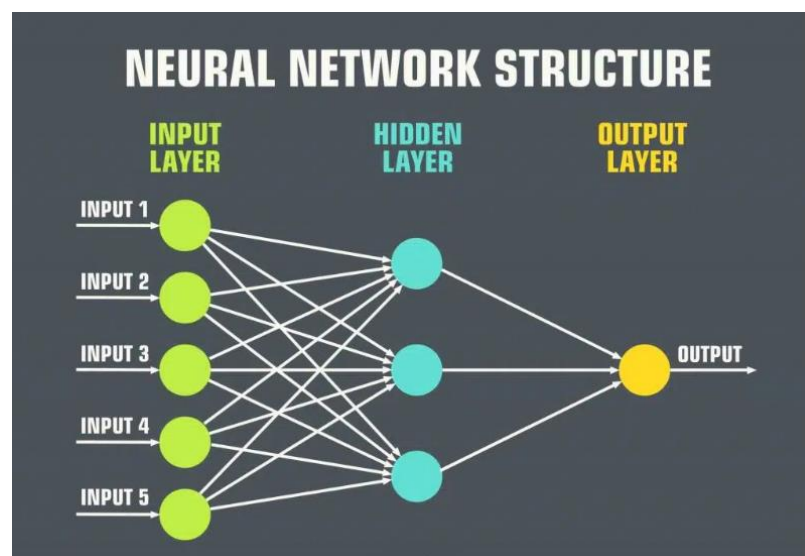


Рис.1.2 – Принцип роботи нейронних мереж

Принцип функціонування нейронних мереж заснований на навчанні. На вхід нейронної мережі надходять дані, які являють собою різні параметри або характеристики завдання. Нейрони всередині мережі з'єднані між собою ваговими зв'язками, які мають певні значення. Процес навчання починається з подачі навчальних даних на вхід мережі. У процесі навчання ваги зв'язків між нейронами поступово адаптуються під впливом навчальних прикладів, змінюються з метою мінімізації помилок моделі. Такий принцип адаптації ваг дає змогу нейронній мережі навчатися на різноманітних даних, що робить її універсальним інструментом для розв'язання різних завдань. Сутністю навчання полягає у здатності нейронної мережі виділяти складні закономірності із вхідних даних.

Крім того, ІНС здатні працювати в нових ситуаціях. Нейронна мережа, навчена на певному завданні, здатна застосовувати отриманий досвід для розв'язання аналогічних або пов'язаних завдань. Наявні закономірності використовуються для прийняття рішень у нових ситуаціях, що робить нейронні мережі потужним інструментом для аналізу, класифікації та прогнозування.

Види нейромереж:

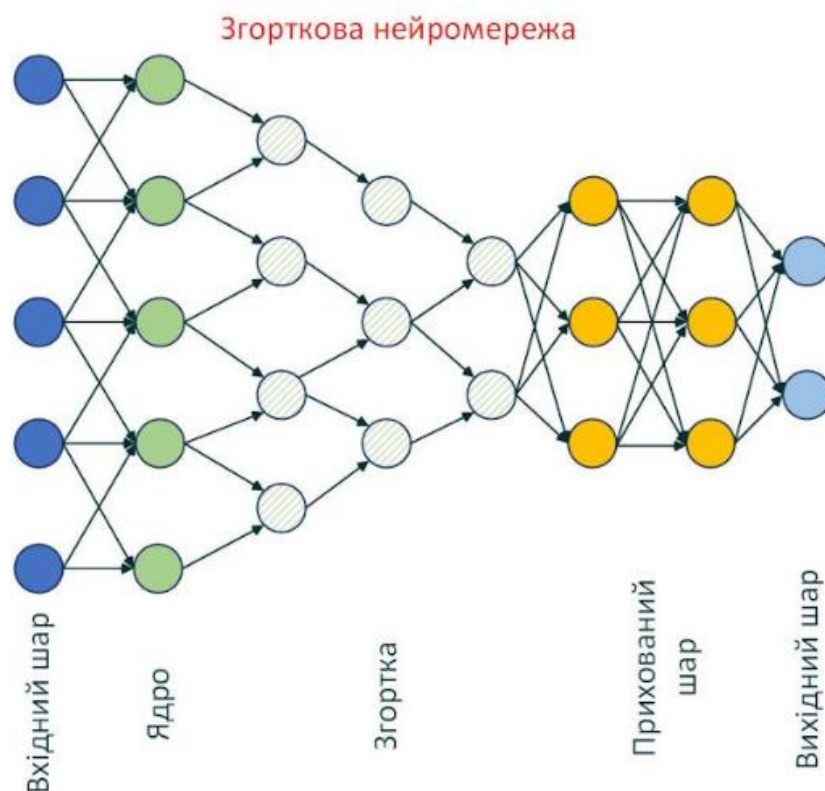


Рис.1.3 – Згорткова нейромережа

Рекурентна нейронна мережа

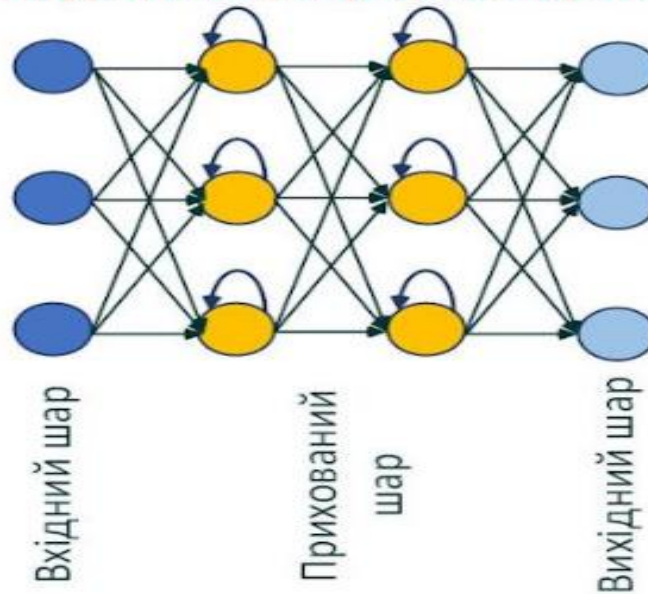


Рис.1.4 – Рекурентна нейронна мережа

Існує багато видів нейромереж, кожна з яких спеціалізована на вирішенні певних завдань. Наприклад, згорткові нейронні мережі (CNN) призначені для ефективного обробки зображень. Вони виділяють важливі ознаки і дають змогу розпізнавати об'єкти. Рекурентні нейронні мережі (RNN), призначені для роботи з послідовними даними, зберігають інформацію про попередні кроки і використовують її для складнішого аналізу.

Використання різних типів нейронних мереж зумовлено їхніми спеціалізованими можливостями, що дає змогу ефективно розв'язувати різноманітні задачі в галузі штучного інтелекту та машинного навчання.

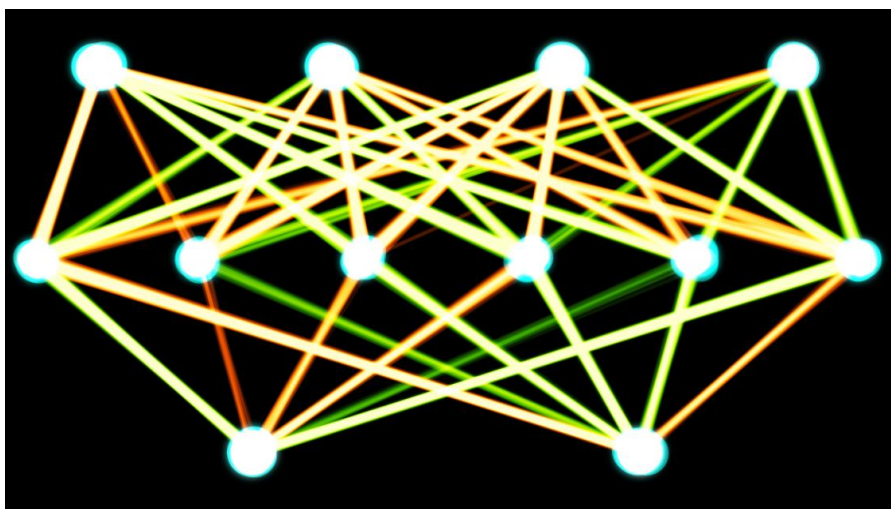


Рис.1.5 - Одношарова штучна нейронна мережа прямого поширення

Нейронна мережа прямого поширення, нейромережа прямого розповсюдження — вид нейронної мережі, в якій сигнали поширюються в одному напрямку, починаючи від вхідного шару нейронів, через приховані шари до вихідного шару і на вихідних нейронах отримується результат опрацювання сигналу. В мережах такого виду немає зворотніх зв'язків. Прикладом нейронної мережі прямого поширення є перцептрон Розенблатта, від якого і беруть свій початок нейромережі прямого розповсюдження.

Застосування нейронних мереж при аналізі даних:

Нейронні мережі мають дивовижну універсальність і здатні вирішувати широкий спектр завдань:

Розпізнавання образів: нейронні мережі успішно застосовуються для розпізнавання образів у зображеннях. Це може включати в себе завдання, такі як розпізнавання осіб, об'єктів або навіть більш складних структур в медичних зображеннях.

Обробка природної мови: в області обробки природної мови (NLP), нейронні мережі використовуються для створення систем автоматичного перекладу, аналізу текстів, генерації мови і навіть відповідей на питання. Вони здатні імітувати розуміння і використання людської мови.

Аналіз даних: нейромережі ефективні під час аналізу великих обсягів даних. Вони можуть виявляти закономірності, тенденції та приховані взаємозв'язки, що робить їх цінними інструментами в галузі аналітики даних.

Прогнозування: здатність нейронних мереж передбачати результати на основі наявних даних робить їх ефективними інструментами для прогнозування, у фінансах, метеорології або інших галузях.

Застосування нейронних мереж у створенні оптимальних методологій розробки ПЗ:

У розробці програмного забезпечення, нейронні мережі привносять сучасні методи оптимізації процесів:

Аналіз даних про минулі проєкти: нейромережі можуть аналізувати великі

обсяги даних з попередніх проєктів, як-от час виконання завдань, баг-репорти, і використання ресурсів. Екстракція значущих патернів із цих даних допомагає визначити успішні стратегії, а також ті аспекти, де виникли складнощі.

Прогнозування необхідних ресурсів: на підставі даних з минулих проєктів, нейронні мережі передбачають необхідні ресурси (час, бюджет, персонал) для поточного проєкту. Це дає змогу розробникам і керівникам компанії ухвалювати рішення, спрямовані на оптимальний розподіл ресурсів і скорочення часу розробки.

Ухвалення рішень для досягнення оптимальних результатів: нейромережі допомагають при ухваленні обґрунтованих рішень на етапах проєктування і розробки, з огляду на наявні ресурси та аналіз попередніх досвідів. Це сприяє формуванню більш ефективних методологій розробки, мінімізації ризиків і підвищенню загальної ефективності процесів.

Застосування нейронних мереж у створенні технічного завдання:

Аналіз великого обсягу інформації: нейромережі обробляють дані з попередніх проєктів, виокремлюють ключові моменти, вимоги та особливості технічних рішень. Це дає змогу формувати деталізовані технічні завдання, що ґрунтуються на реальних даних і успішних дослідах.

Відбір ключових вимог: нейромережі аналізують текстову інформацію з попередніх технічних завдань, виділяють ключові вимоги та параметри проєктів. Це полегшує процес створення технічного завдання, дає змогу зосередитися на найважливіших аспектах проєкту.

Надання цінних рекомендацій: на основі даних про попередні проєкти, нейромережі надають цінні рекомендації щодо структури та змісту технічного завдання. Це підвищує якість і повноту технічних завдань, покращує комунікацію між командою розробки та замовником.

Тестування програмного забезпечення за допомогою нейронних мереж:

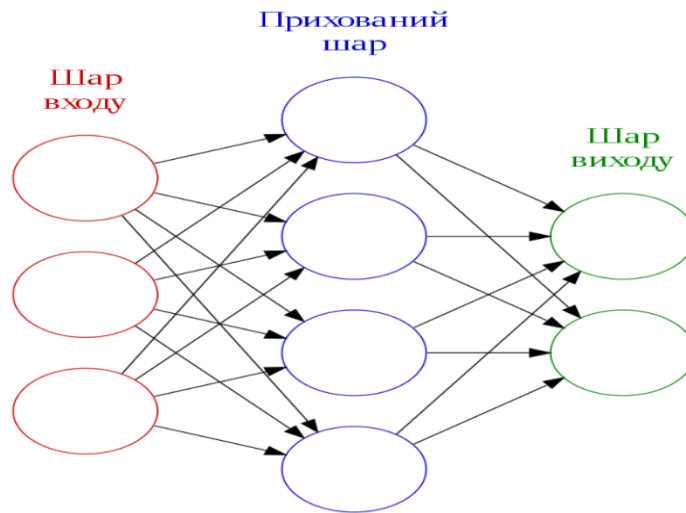


Рис.1.6- Штучна нейронна мережа

Нейромережі використовуються для створення систем автоматизованого тестування, що дає змогу виявити дефекти в коді та запобігти можливим помилкам. Це скорочує час, необхідний для ручного тестування, і покращує загальну стабільність програмного продукту.

Нейромережі проводять аналіз коду, виявляють потенційні помилки і вразливості, що сприяє поліпшенню якості програмного забезпечення. Що, своєю чергою, підвищує надійність коду та зменшує ризик виникнення критичних збоїв у процесі експлуатації.



Рис.1.7 - Тестування нейронними мережам користувачами

Застосування нейронних мереж у тестуванні допомагає виявляти й усувати проблеми, які могли б залишитися непоміченими за традиційних методів тестування. Це сприяє створенню більш надійного програмного забезпечення, покращує досвід користувачів і знижує ризики технічних збоїв.

Порівняння переваг та недоліків використання нейромереж:

Плюси нейронних мереж:

Висока ефективність у розв'язанні складних завдань: нейромережі демонструють унікальну здатність розв'язувати складні завдання, в яких традиційні методи можуть виявитися неефективними. Їхня здатність виявляти складні закономірності в даних робить їх потужним інструментом у галузях, таких як розпізнавання образів і аналіз текстів.

Навчання на основі даних: нейронні мережі можуть навчатися на основі наявних даних, автоматично адаптуються до змін і покращують свою продуктивність з плином часу. Вони здатні ефективно розв'язувати завдання, навіть якщо вони пов'язані з постійними змінами у оточуючому середовищі.

Мінуси нейронних мереж:

Використання великого обсягу даних: нейромережі вимагають великих даних для ефективного навчання. У деяких випадках, особливо в медичних або наукових галузях, забезпечити такий обсяг даних просто неможливо.

Складність у розумінні: архітектура нейронних мереж може бути складною для розуміння, навіть для досвідчених фахівців. Унаслідок цього ускладнюється процес налаштування та інтерпретації результатів, а здатність працювати як "чорний ящик" ускладнює пояснення прийнятих рішень, що може бути проблемою.

Відсутність пояснення рішень: на відміну від класичних методів, нейромережі не завжди надають зрозумілі пояснення своїх рішень. Це може бути критично в задачах, де важливо розуміти логіку прийнятих рішень, наприклад, у медичних діагнозах.

Незважаючи на зазначені недоліки, використання нейронних мереж в ІТ і розробці продовжує зростати, оскільки їхня здатність до вирішення складних завдань та адаптація до динамічних умов надає значні переваги в сучасному світі технологій.

Також це скорочує час для роботи після аналізу задачі, бо нейромережа може виконати ряд дій які пришвидшать процес розробки програмного коду та план розташування об'єктів поля гри.

1.2. Штучний інтелект та його використання в розробці програмного забезпечення



Рис.1.8 – Штучний інтелект

Штучний інтелект (ШІ) — це галузь інформатики, яка займається розробкою інтелектуальних машин, здатних виконувати завдання, які зазвичай потребують людського інтелекту. Системи штучного інтелекту створені для навчання на досвіді, розпізнавання закономірностей і ухвалення рішень на основі вхідних даних. Ці системи можна навчити виконувати певні завдання, наприклад, розпізнавати зображення, розуміти природну мову або грати в ігри. Технологія штучного інтелекту охоплює широкий спектр методів, включаючи машинне навчання, обробку природної мови, робототехніку, експертні системи тощо. Мета досліджень штучного інтелекту полягає в тому, щоб створити машини, які можуть міркувати, розуміти та навчатися, як люди, і використовувати ці можливості для покращення життя людства та вирішення складних проблем.

У більшості випадків алгоритм розв'язання проблем невідомий наперед. Точного визначення цієї науки немає, оскільки у філософії не розв'язано питання про природу і статус людського інтелекту. Немає і точного критерію досягнення комп'ютером «розумності», хоча перед штучним інтелектом було запропоновано низку гіпотез, наприклад, тест Тюрінга або гіпотеза Ньюелла-Саймона. Нині існує багато підходів як до розуміння задач штучного інтелекту, так і до створення інтелектуальних систем.

Це досягається шляхом вивчення закономірностей роботи людського мозку та аналізу когнітивних процесів. Результатом цих досліджень є розробка інтелектуального програмного забезпечення та систем.

ІІІ є важливою складовою багатьох сучасних технологій та має широкий спектр застосувань. Ось деякі ключові аспекти та області використання ІІІ:

Обробка природної мови (NLP):

Використовується у чат-ботах, перекладачах, системах розпізнавання мови та голосових помічниках, таких як Siri або Alexa.



Рис.1.9 – Використання NLP

Комп'ютерний зір:

Використовується для розпізнавання обличч, обробки зображень, в автономних транспортних засобах для виявлення перешкод та визначення маршруту.

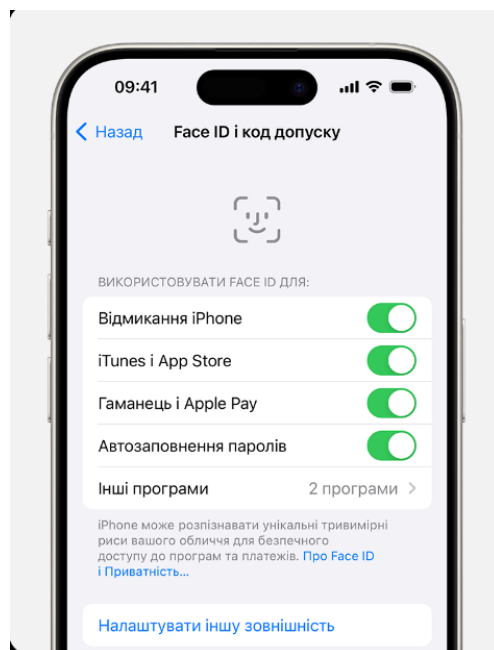


Рис.1.10 - Використання Face ID

Ігрова індустрія:

ШІ створює реалістичних NPC, адаптує складність гри до рівня гравця, допомагає виявляти шахраїв та покращує загальний ігровий досвід.



Рис.1.11 - Приклад створення NPC

Аналіз даних:

ШІ аналізує великі обсяги даних, виявляє закономірності та тренди, допомагає у прийнятті рішень в бізнесі, науці та державному управлінні.



Рис.1.12 - Аналіз даних ШІ

Переваги використання ІІІ:

Автоматизація: ІІІ дозволяє автоматизувати рутинні та складні задачі, знижуючи потребу в ручній праці.

Швидкість і ефективність: ІІІ може обробляти великі обсяги даних набагато швидше, ніж людина.

Точність: У багатьох випадках ІІІ забезпечує вищу точність і мінімізує кількість помилок.

Приватність: Використання ІІІ для збору та аналізу даних викликає питання про конфіденційність особистої інформації.

Безробіття: Автоматизація може призвести до втрати робочих місць у деяких секторах.

Етичність: Прийняття рішень на основі ІІІ повинно бути етичним і прозорим, щоб уникнути дискримінації та несправедливості.

ІІІ продовжує активно розвиватися і інтегруватися в різні сфери нашого життя, роблячи його більш зручним та ефективним.

Роль ІІІ в розробці ПЗ:

Штучний інтелект (ІІІ) відіграє важливу роль у розробці програмного забезпечення (ПЗ), покращуючи ефективність, точність і швидкість процесу розробки. Ось кілька ключових способів, як ІІІ використовується в цій галузі:

1. Автоматизація кодування

Автозаповнення коду: Інструменти на основі ІІІ, такі як GitHub Copilot, можуть пропонувати автозаповнення коду, зменшуючи кількість рутинної роботи та підвищуючи продуктивність розробників.

Генерація коду: ІІІ може автоматично генерувати код на основі опису завдань або специфікацій, що прискорює процес розробки.

2. Тестування ПЗ

Автоматизація тестування: ІІІ може автоматично створювати та виконувати тестові сценарії, забезпечуючи більш широке покриття тестами та швидше виявлення помилок.

Аналіз помилок: ІІІ допомагає ідентифікувати корінні причини помилок і

пропонує способи їх усунення, що спрощує процес відладки.

3. Аналіз вимог і планування

Обробка природної мови (NLP): ШІ може аналізувати вимоги до ПЗ, написані природною мовою, і перетворювати їх у технічні завдання.

Прогнозування: ШІ може аналізувати історичні дані про проєкти, щоб прогнозувати терміни виконання, бюджет і можливі ризики.

4. Підтримка та обслуговування

Чат-боти: ШІ-чат-боти можуть забезпечувати підтримку користувачів, відповідати на питання та допомагати вирішувати проблеми.

Аналіз логів: ШІ аналізує журнали роботи ПЗ, виявляючи аномалії та передбачаючи потенційні збої.

5. Безпека

Виявлення вразливостей: ШІ може аналізувати код і виявляти потенційні вразливості безпеки, допомагаючи запобігати кібератакам.

Моніторинг загроз: ШІ аналізує мережевий трафік і поведінку системи, виявляючи підозрілу активність та реагуючи на потенційні загрози.

6. Оптимізація процесів

Управління проєктами: ШІ може аналізувати дані про проєкти і пропонувати оптимальні шляхи їх виконання, враховуючи ресурси та терміни.

Аналіз продуктивності: ШІ аналізує продуктивність розробників і команд, допомагаючи виявити області для покращення та оптимізації процесів розробки.

Приклади інструментів та платформ:

GitHub Copilot: Інструмент для автозаповнення коду на основі ШІ, розроблений GitHub у співпраці з OpenAI.

Snyk: Платформа для безпеки коду, яка використовує ШІ для виявлення вразливостей у коді та залежностях.

SonarQube: Інструмент для аналізу коду, який використовує ШІ для виявлення помилок, вразливостей та покращення якості коду.

ШІ стає невід'ємною частиною процесу розробки ПЗ, допомагаючи розробникам створювати більш надійні, безпечні та ефективні програми.

1.3. Опис існуючого аналогу програмного продукту

Brick ball

Як одна з найпопулярніших ігор у світі, Bricks Ball Crusher забезпечить вам нескінченне задоволення. Гра містить десятки тисяч добре продуманих рівнів і понад 200 блоків навичок і куль навичок, які ви можете досліджувати, а також різні режими випробувань, такі як "режим порятунку". На що ти чекаєш? Спробуйте використати потужний м'яч, щоб прицілитися і стріляти цеглинами, щоб відчути чарівний світ ліквідації.

Оригінальність гри

Гра відрізняється яскравими кольорами та веселою музикою. Хоча в грі є реклама, її можна пропустити приблизно через п'ять секунд, щоб якнайшвидше повернутися до гри. У грі є кілька рівнів, кожен з яких включає більше цеглинок і більше труднощів. На кожній цеглині є номер, за який нараховуються окуляри. Єдине, що потрібно робити, - це переміщати кульку в нижній частині екрана, направляючи стрілку туди, куди вона повинна потрапити, а потім натискаючи кнопку Enter, щоб запустити кульку.

Найчастіше одна куля вибиває більшу частину цегли. Одна куля перетворюється на декілька, оскільки вона продовжує вдарятися об краї екрану, що збільшує шанси збити більше цегли за один раз. Щоб перейти на наступний рівень, потрібно очистити гру від цегли. Є стрілка, яка може зупинити м'яч, якщо жодна з цегли не потрапляє в ціль.

Спочатку після відкриття гри ми потрапляємо до меню, де бачимо кнопки запуск, мультиплеєр та редагування музики та звуків у грі (рис.1.14)

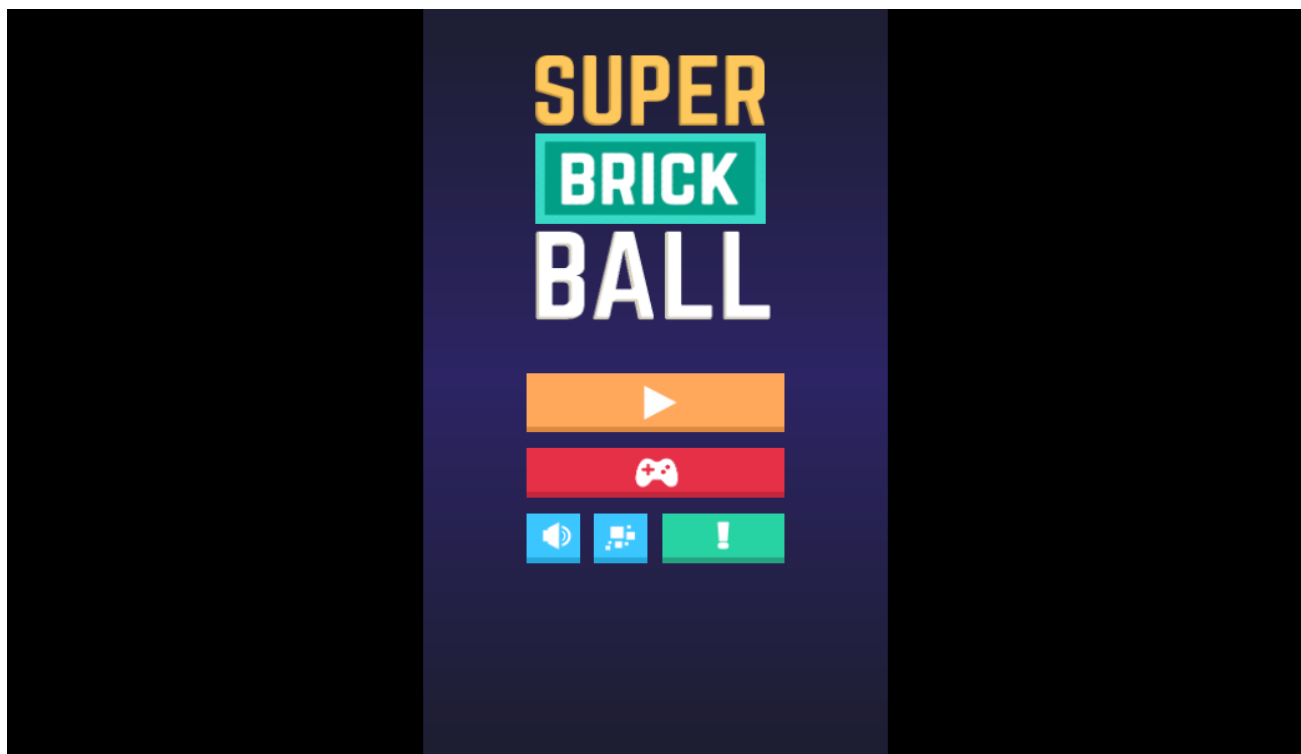


Рис. 1.14 – Головне меню

Гравець який потрапляє до меню вибору рівнів, бачить екран який йому показує скільки рівнів він вже виконав, і з ті, що виконані ви має право пройти ще раз якщо забажає (рис. 1.15), також є кнопка повернення до головного меню.

Рівні які не доступні гравцеві, закриті замочками, знизу від них є три крапки які позначають кількість сторінок рівнів (Рис.1.16).

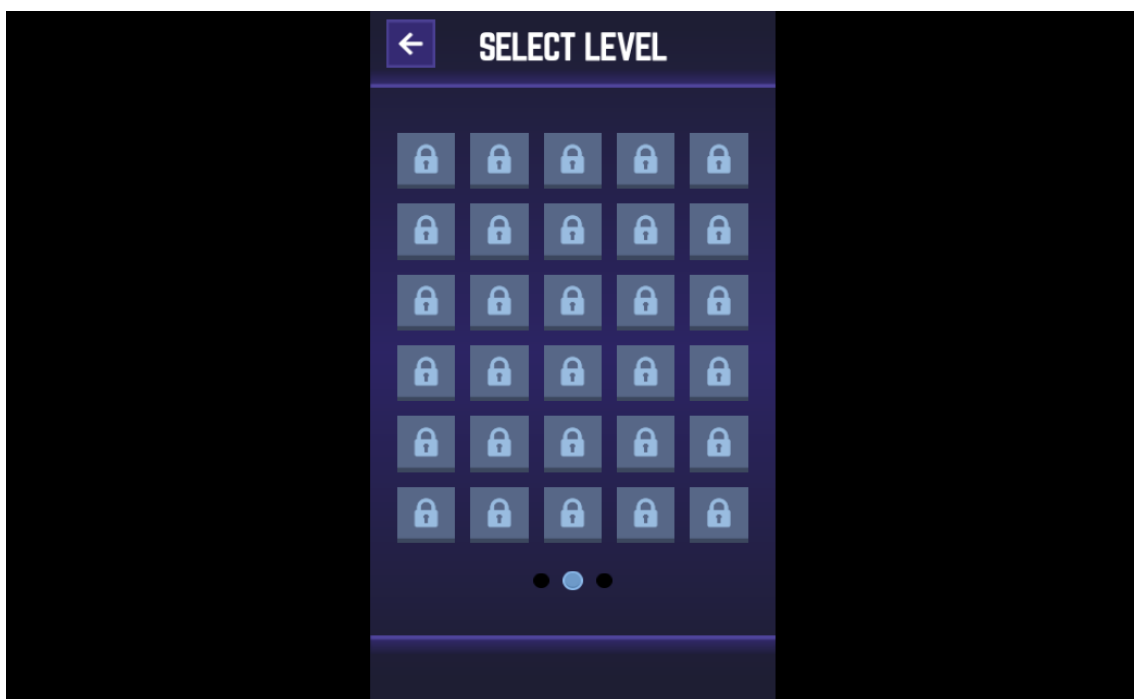


Рис.1.15 – Вибір рівнів за власним бажанням

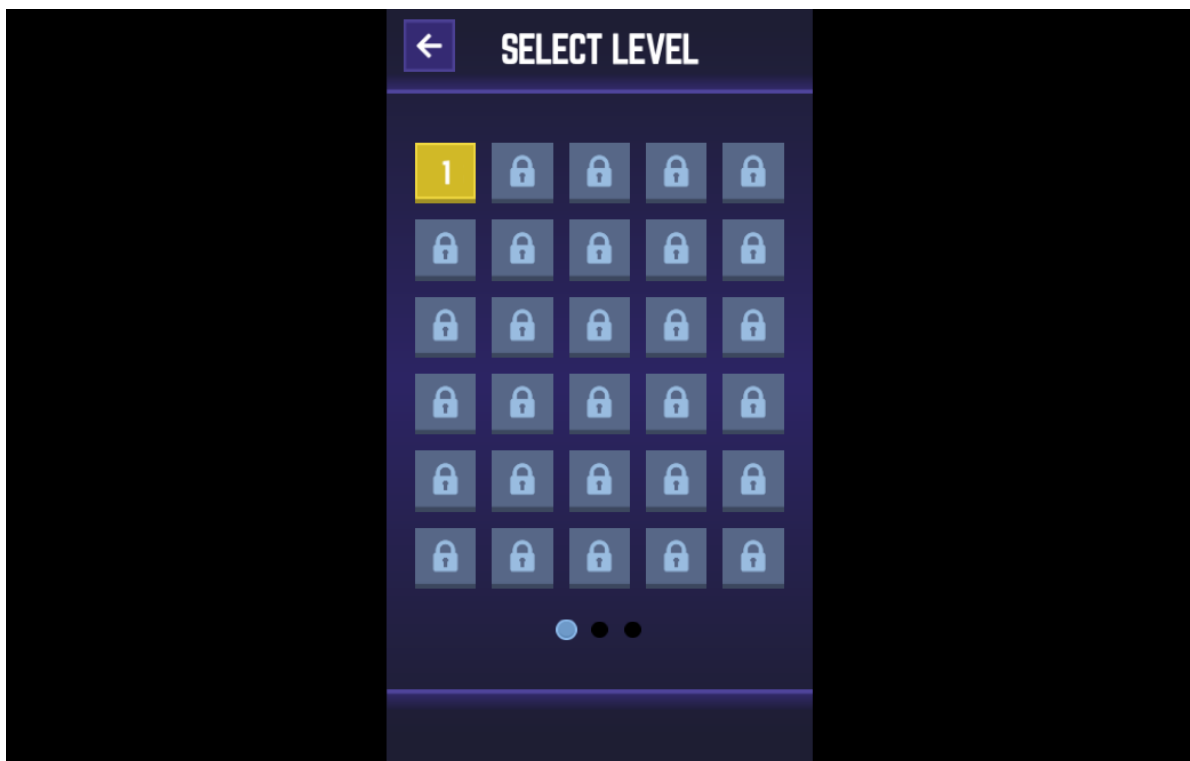


Рис. 1.16 – Меню вибору рівнів

Екран безпосередньо самої гри

Гравець потрапляє до екрану даної гри, і що ж він бачить? Перед ним з'являється екран із зображенням саме даного мяча, який і є головної зброєю нашого гравця, даний мяч стоїть на платформі. Зверху гравець бачить таблицю рекорду яка буде змінюватись після кожного його кращого результату за попередній. По середині табло запису результату саме за нинішню сесію, і з права зліва кнопка “пауза”(Рис. 1.18).

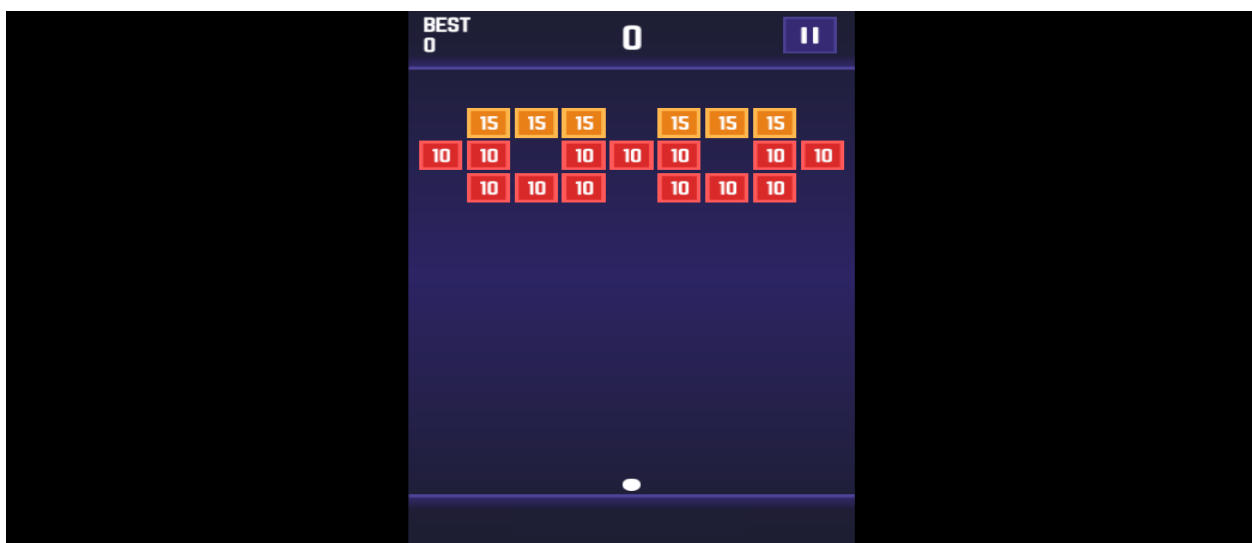


Рис.1.17 -Екран безпосередньо самої гри

Пауза

Після нажаття кнопки паузи гравець побачить ще одне меню в якому можна буде редагувати звук(вкл\викл), кнопку закрученої стрілки, яка буде означати, що можна почати рівень заново, також кнопку повернення до вибору рівнів і провження даної гри.

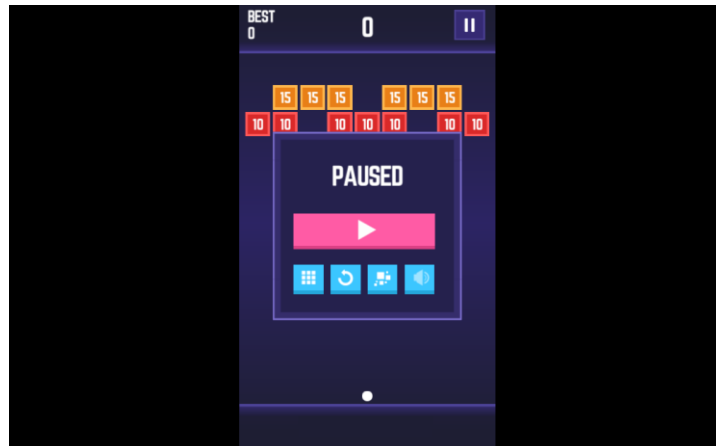


Рис. 1.18- Пауза

Тестуємо продукт як гравець

Після початку гри буде запуску гри буде запущено черга шариків які будуть летіти один за одним до цілі в яку ви їх направите (Рис. 1.19). Один шар при зіткненні з кубом з номерацією буде знімати 1 показник, але відскочити він може також і від стінки полетівши в інші кубики (Рис.1.20.).

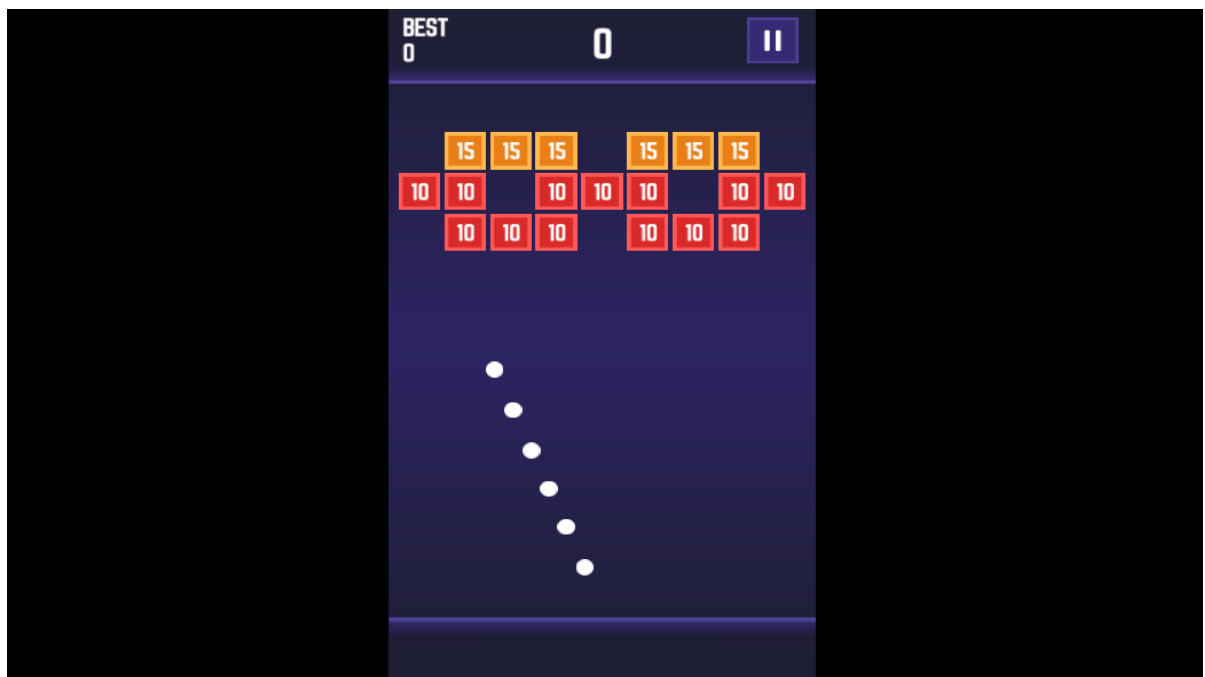


Рис. 1.19 – Виліт кульок в блоки чисел



Рис. 1.20- Відштовхування та зменшення номерації на блоках

Механіка кожного рівня

Після того як шари вдаряються в блоки з числами, вони тягнуться до низу, і коли перед ними закінчуються блоки вони летять до нижньої платформи, що зайнятий стандартне своє положення. Після кожного пострілу шариків, блоки які залишились не зниженими опускаються на одну клітинку до низу. Також в правому нижньому кутку ми маємо стрілочку, за допомогою якої можемо не чекати, а дати нейромережі вирахувати приблизне зниження блоків (рис. 1.12).

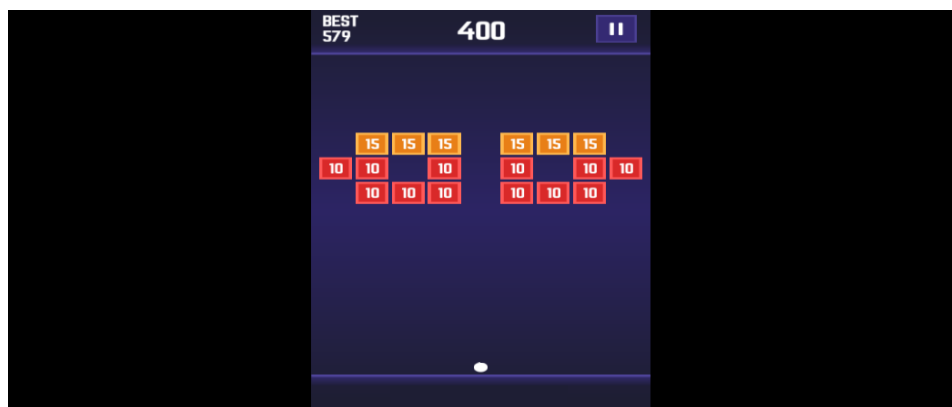


Рис. 1.21 – Зміна положення блоків після декількох пострілів

Проходження рівня до кінця

Тут в нас є 2 варіанти закінчення гри:

1.Рівень не пройдено. Коли всі блоки опустяться до рівня мяча і ми їх не зможемо зруйнувати, це буде означати програш гравця на даному етапі, і він побачить повідомлення про поразку в даному рівні, із запропонованими варіантами подальшого розвитку (Рис.1.22).

2.Рівень пройдено успішно. Коли всі блоки на полі буде зруйновано, гравець побачить повідомлення закінчення рівня з подальшим переходом на наступний рівень.(Рис.1.23)

Також в меню вибору рівнів буде відкрито ще один додатковий рівень, і тоді гравець буде мати право вибору будь якого з доступних йому рівнів.(Рис.1.24).

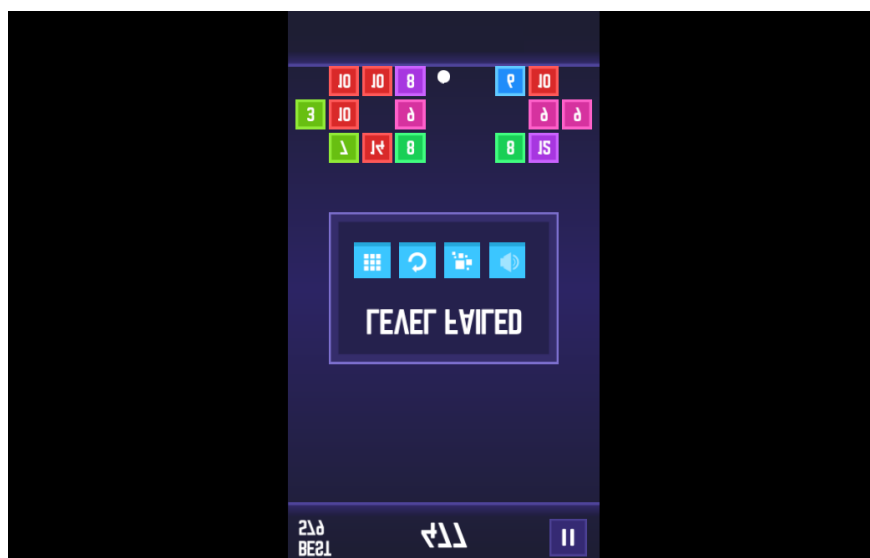


Рис.1.22 – Гравець зазнав поразки на даному рівні

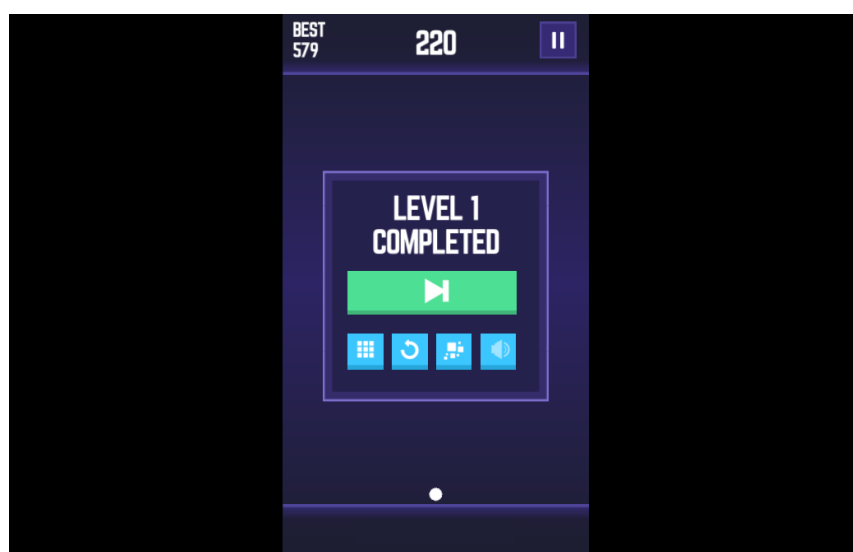


Рис. 1.23 – Гравець досяг успіху і пройшов рівень

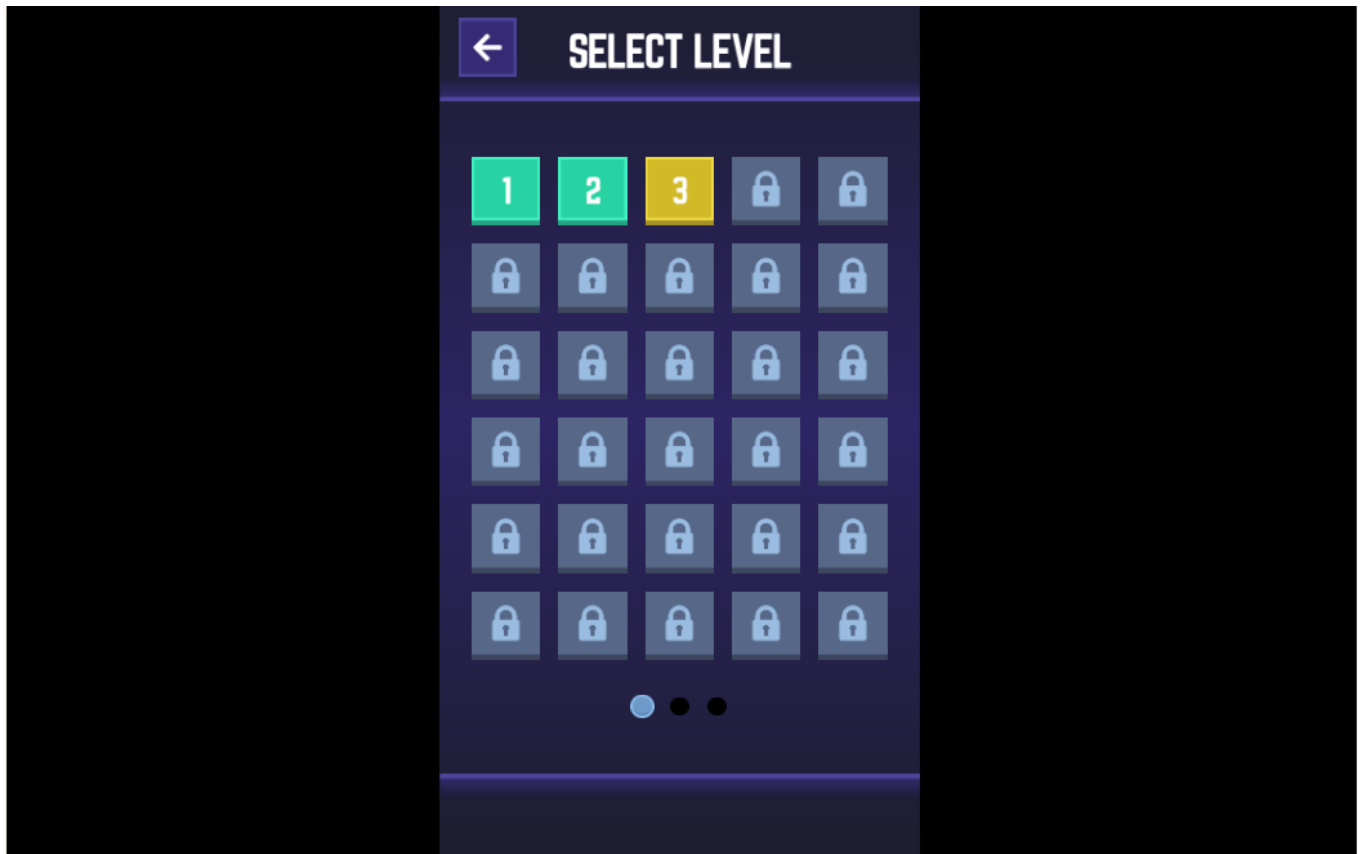


Рис. 1.24 – меню вибору рівнів з новими відкритими

З кожним подальшим рівнем складність збільшується:

1. З'являється захисна стіна, яку не можна пробити, тим самим ризик поразки зростає (Рис.1.25).
2. Розміщення блоків стає все ближче до платформи з мячем (Рис.1.26.)
3. Кількість ударів яку потрібно нанесту в блок теж зростає, і тому гравцю доведеться придумувати різні кути пострілу, для ефективного результату (Рис.1.27.)

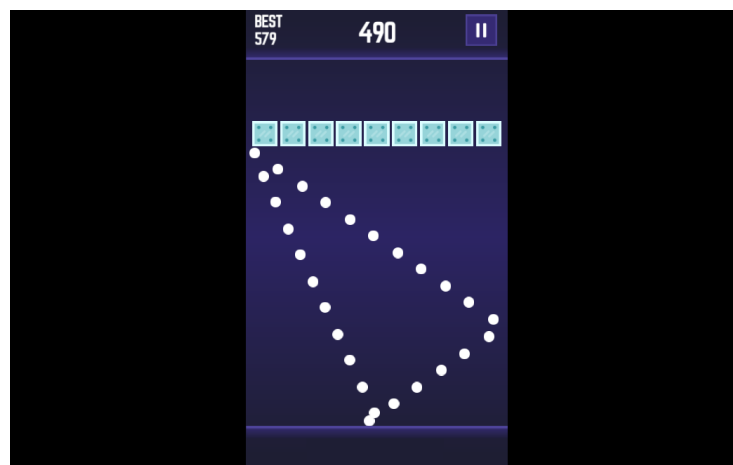


Рис.1.25 – М'ячі вдаряються в захисну стіну яка опускається до низу

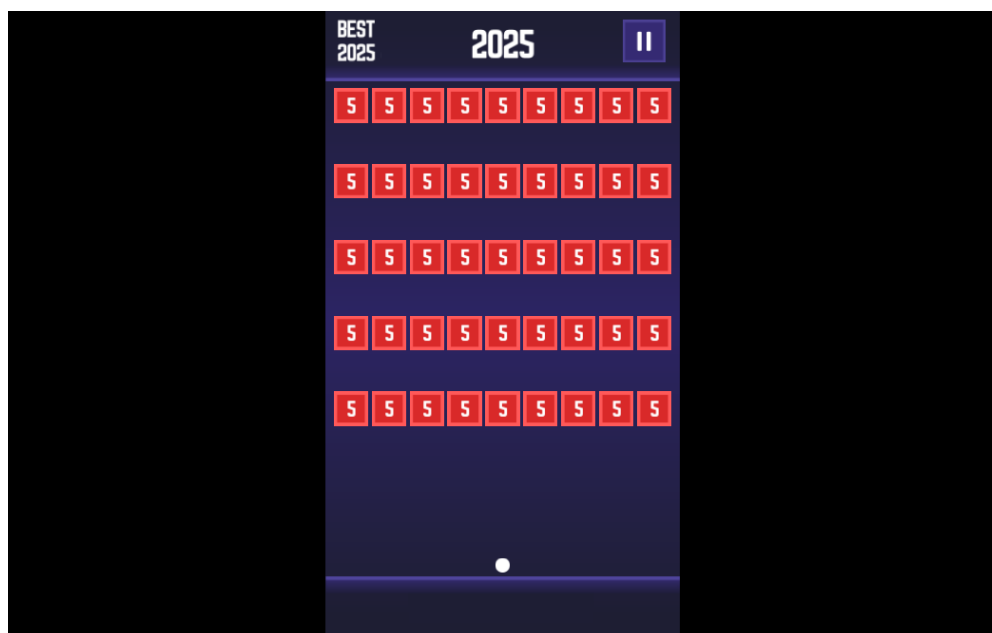


Рис.1.26 – Блоки розміщені набагато ближче до платформи з мячем ніж раніше

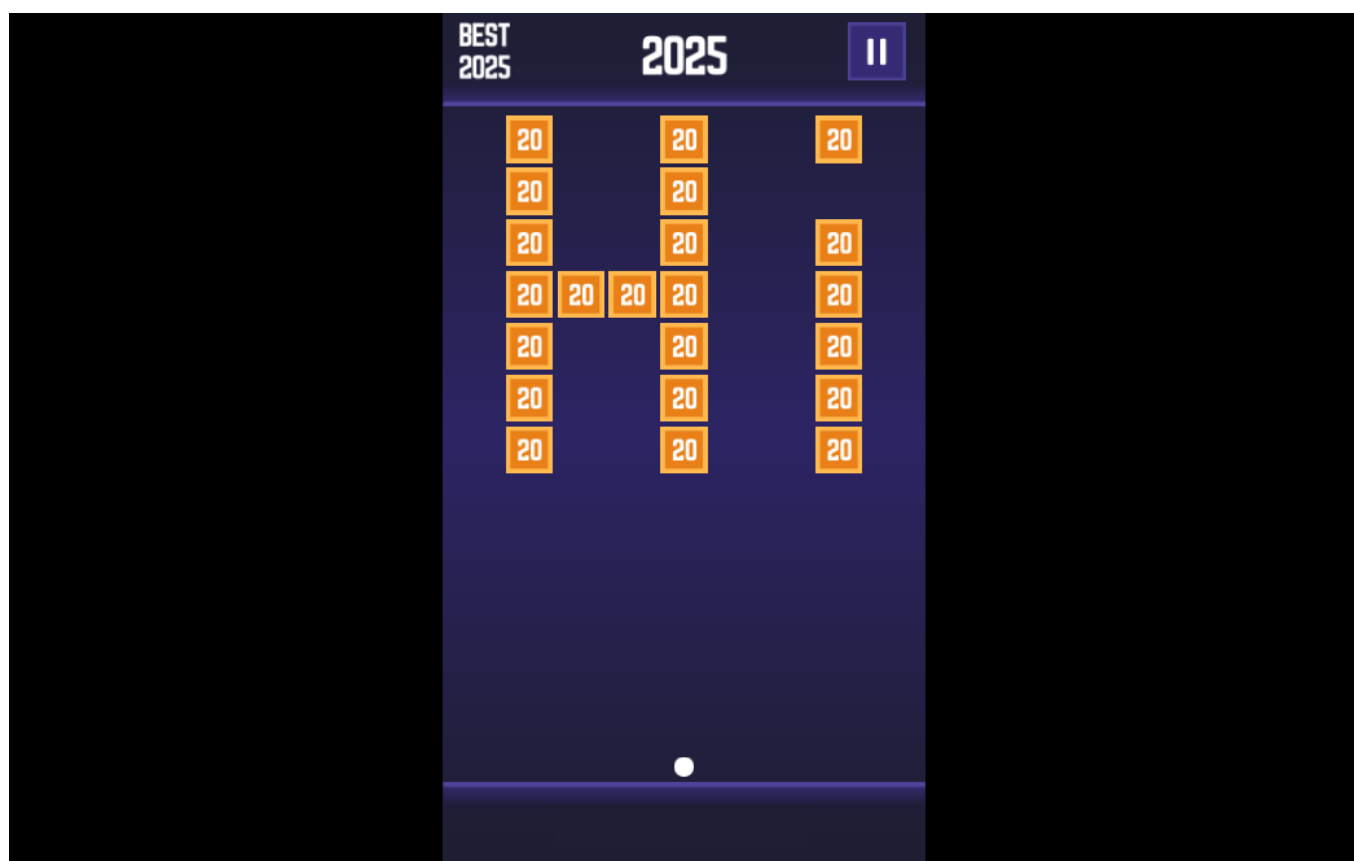


Рис.1.27 – Блоки мають більшу чисельність ні на попередніх рівнях

Також слід побачити те що рекорд встановлений на момент однієї і тієї ж сесії буде зберігатися до моменту повного оновлення гри(Рис.1.28)

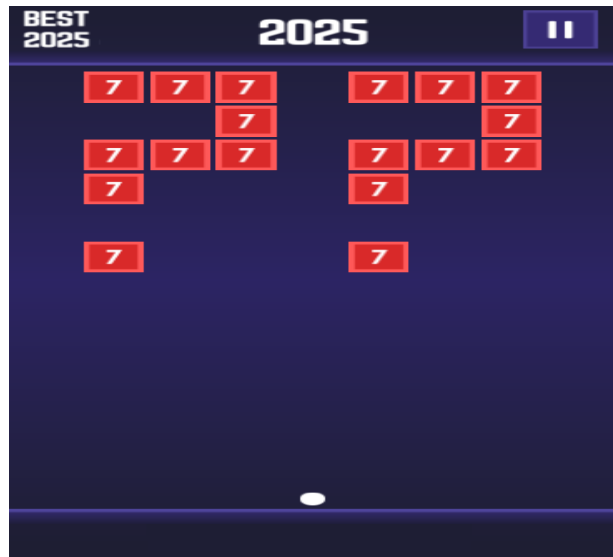


Рис.1.28 – Збереження ігрового рекорду

1.4. Аналіз інструментальних засобів графіки

Перед початком роботи над певним програмним продуктом необхідно провести аналіз, який програмний інструментарій вам може знадобитись для виконання поставлених завдань. Так як практичною частиною моєї кваліфікаційної роботи являється гра. Розглянемо з яких основних частин і компонентів вона складається, для того щоб вибрати програмне забезпечення для подальшої розробки. Відеогра складається з 2 основних частин, це візуальна частина (графіка гри) та програмна (програмний код гри).

Перед вибором інструментів для розробки необхідно визначитись з яким типом графіки буде йти роботи. Розглянемо більш детально які види графіки існують. Також не буде забувати про вибір графіки, на даному етапі у нас графіка 2D на платформі Unity 3D. Розрізняють три види комп'ютерної графіки: растрову графіку, векторну графіку і фрактальну графіку. Растрову графіку застосовують при розробці електронних (мультимедійних) і поліграфічних видань.

На відміну від растрової графіки, в якій зображення формується з сукупності точок, в векторній графіці зображення являє собою сукупність простих елементів: прямих ліній, дуг, кіл, еліпсів, прямокутників. Які називаються графічними примітивами. Положення і форма графічних примітивів задаються в системі координат, пов'язаних з екраном.

Фрактальна графіка, як і векторна, - обчислюється, але відрізняється тим, що ніякі об'єкти в пам'яті комп'ютера не зберігаються. Зображення будується по управлінню, тому нічого, крім формули зберігати не треба. Зміна коефіцієнтів в рівнянні дозволяє отримувати абсолютно іншу картину.

1.5. Задачі для виконання

Наш продукт буде формуватися на основі жанру казуальна гра. Вона також може кваліфікувати як аркада, адже вона розрахована на будь яку вікову категорію людей від дітей до дорослих.

Гра на відміну від аналогій матиме режим мультиплеєра, щоб можна було винести її на більш широку аудиторію.

Створюємо меню гри з якмога більше функціями, для зручності користувача, щоб йому було простіше розібратися в функціоналі майбутнього продукту.

Створити валюту яку гравці будуть заробляти на протязі того як вони будуть грати, щоб в подальшому апгрейдити свої навички і тд.

Додати функцію магазину, щоб користувач міг використовувати переможні очки або валюту ігрову для закупівлі ефектів посилення персонажу щоб мати на початку вже перевагу над опонентом

Технології створення

1. Розробка концепції:

вік користувача.

вид гри: розважальна, розвиваюча, навчальна, комплексна.

2. Розробка адресних вимог до гри:

до сценарію.

до ігрового середовища.

до графіки гри.

3. Написання сценарію:

створення сюжетної канви

В створенні процесу гри ми само собою використаємо ШІ для того, щоб він

там реалізував генерацію сценарію сценарій казуальної гри.

Створення гри завжди починається з ідеї, яка буде покладена в основу всього наступного сюжету. Деталі цієї ідеї можуть змінюватись від графічних технологій до глобального сетингу.

Процес створення гри еволюційний. За час існування проекту можуть кілька разів змінитись основні моменти гри (навіть жанр), а також дрібніші нюанси. Щоб гра була стійка у програмуванні потрібна міцна концепція. Створення концепції може бути поетапним процесом. Насамперед слід створити концепцію, а вже потім переходити до її реалізації.

Перш за все потрібно визначитися з основною ідеєю. Головна ідея ви значає простір для майбутньої роботи. Можна провести аналогію між концепцією проекту та таким математичним поняттям, як базис – набір векторів, які однозначно задають простір. Створення концепції гри – це складніший процес, оскільки він не настільки математично вивірений.

Для створення успішної гри потрібно постійно знаходити і генерувати ідеї, що приваблюють користувачів. Натхнення можна зустріти і в реальному житті, але глобально є три напрями для пошуку ідей:

1. Аналіз популярних трендів на ринку.
2. Проведення арт-конкурсів серед спільноти геймерів.
3. Розробка унікальної концепції для привернення уваги.

Часто розробники ігор практикують проведення арт-конкурсів серед спільнот гравців.

Цей крок розробки є дуже важливим, він відбувається безпосередньо перед початком фактичного виробництва і тут, щоб перевірити основну концепцію та ідею. Маючи хороший прототип, потрібно добре усвідомлювати всі складності на шляху до успішного проекту та бути готовим до їх усунення. Це перша віха, але також єдина, на котрій вирішується подальша доля проекту. Якщо результати тестів, проведених на етапі першого ігрового прототипу, є задовільними, стає зрозуміло, що можна почати повну розробку гри. Однак важливо зупинити розробку та повернутися на етап назад чи переосмислити ідею, якщо цей етап показав якісь недоліки чи

проблеми.

1.6. Обґрунтування вибору платформи для розробки додатку

Для розробки взято Unity 3D - потужне середовище розробки для кросплатформенного двигуна Unity для створення відеоігор та додатків. Створені за допомогою Unity 3D ігри та програми працюють в операційних системах Windows, OS X, Android, iOS, Linux, Blackberry а також на ігрових приставках Wii, PlayStation 3 та Xbox 360. Програми, створені за допомогою Unity, підтримують DirectX та OpenGL.

За допомогою Unity 3D можна створювати відеоігри будь-яких жанрів. Розробник може легко імпортувати текстури, моделі та звуки. Для текстури підтримуються всі популярні формати зображень. Створення скриптів здійснюється, переважно, з допомогою C#.

Ігрові движки – це інструменти, які використовуються дизайнерами ігор для швидкого та легкого програмування та планування гри, не створюючи її з нуля. Незалежно від того, чи це 2D-ігри чи 3D-ігри, вони пропонують інструменти, які допоможуть у створенні та розміщенні елементів.

Таким чином, ці движки – це в основному архітектура, на яку розробники ігор покладаються для запуску відеоігри. Типовий ігровий движок дозволяє розробникам включати в гру такі елементи: введення, рендеринг, сценарії, виявлення зіткнень, штучний інтелект тощо. Іншими словами, ігрові движки — це багаторазові частини, які розробники можуть використовувати для створення основи гри. Це дає їм більше часу, щоб зосередитися на унікальних аспектах, таких як моделі персонажів, текстури, об'єкти, що взаємодіють один з одним тощо. Без цих механізмів розробки ігор довелося б створювати ігри з нуля, і, отже, це займало б більше часу та було б складніше зробити. Таким чином, ці ігрові движки насправді спрощують дизайн ігор. Тому розробники та менеджери ігор повинні прийняти правильне рішення, коли справа доходить до вибору правильної платформи ігрового движка для своїх потреб, незважаючи на велику різноманітність рішень, доступних сьогодні [9].

Варто проаналізувати, що обране рішення справді дозволяє вирішувати кілька проблем одночасно. В середньому ігровий двигун повинен бути в змозі спростити такі важливі завдання, як:

- фізика – занурення / фізика у грі мають бути ідеальним балансом між якістю моделювання та обмеженнями обчислювальної потужності для кінцевого користувача;
- введення – це надзвичайно поширена проблема при кросплатформній розробці;
- обробка візуальних ресурсів – освітлення, тіні, накладання текстур та глибина чіткості вимагають менше зусиль при програмуванні при використанні ігрових двигунів.

Тож обраний ігровий двигун повинен дати можливість виконувати вищезазначені завдання з меншими зусиллями написання коду. Це допомагає значно скоротити час розробки і дозволяє командам зосередитися на розробці своїх ігор, щоб забезпечити унікальний і особливий інтерфейс користувача.

Unreal Engine

Одним з найпопулярніших і широко використовуваних ігрових двигунів є Unreal Engine, що належить Epic Games. По суті, це мультиплатформний двигун для розробки ігор, призначений для підприємств будь-якого розміру, який допомагає використовувати технології реального часу для перетворення ідей у привабливий візуальний контент.

Створений розробниками ігор для розробників ігор цей висококласний ігровий движок більше не призначений тільки для студій з мільйонним доходом. Геймдизайнери будь-якого рівня підготовки тепер мають змогу втілити свої ідеї у власну епічну гру.

Початкова версія була випущена ще в 1998 році, і через 19 років вона продовжує використовуватися для деяких з найбільших ігор. Головна перевага Unreal Engine полягає в тому, що він може бути досить модифікований, щоб у

іграх можна було створити унікальний ігровий досвід. Однак для цього потрібні кваліфіковані розробники з великим досвідом.

Написаний мовою C++, рушій дозволяє створювати ігри для більшості операційних систем і платформ: Microsoft Windows, Linux, Mac OS і Mac OS X, консолей Xbox, Xbox 360, PlayStation 2, PlayStation Portable, PlayStation 3, Wii, Dreamcast і Nintendo GameCube. У грудні 2009 Марк Рейн продемонстрував роботу рушія Unreal Engine 3 на iPod Touch і iPhone 3GS. У березні 2010 робота рушія була продемонстрована на комунікаторі Palm Pre, що базується на мобільній платформі webOS [10].

Для спрощення портування движок використовує модульну систему залежних компонентів: підтримує різні системи рендерингу (Direct3D, OpenGL, Pixomatic; раніше підтримувалися Glide API, S3 Metal, PowerVR SGL), відтворення звуку (EAX, OpenAL, DirectSound3D; раніше підтримувалися A3D), засоби голосового відтворення тексту, розпізнавання мовлення (тільки для Xbox360, PlayStation 3, Nintendo Wii і Microsoft Windows, також планувалося для Linux і Mac)[9], модулі для роботи з мережею й підтримка різних пристроїв вводу.

Для гри у мережі підтримуються технології Windows Live, Xbox Live, і GameSpy, що дає можливість підключити до 64 гравців (клієнтів) одночасно. Попри те, що офіційно засоби розробки не містять у собі підтримки великої кількості клієнтів на одному сервері, рушій використовувався для створення MMORPG-ігор. Один з найвідоміших представників жанру, Lineage II, використовує рушій Unreal Engine.

Крім того, Epic Games придбала Quixel, що має величезну бібліотеку «фотограмметричної» тобто реалістичної графіки, яку можна використовувати для створення анімації та відеоігор. Користувачі Unreal Engine можуть безкоштовно використовувати інструменти, що входять до комплекту Quixel (Bridge, Mixer) та всі ресурси бібліотеки Quixel Megascans.

Unreal Engine – один з найпопулярніших ігрових двигунів. Хоча

розробникам подобається UE4, ранній випуск Unreal Engine 5 (UE5) включає деякі довгоочікувані покращення (плюс деякі приголомшливі нові функції). Unreal Engine 5 – це майбутня версія Unreal Engine. Ранній доступ до Unreal Engine 5 був оголошений у травні 2021 року, а повний випуск UE5 очікується пізніше у 2021 – на початку 2022 року.

Велика перевага Unreal Engine – в універсальності та доступності: його можуть використовувати як досвідчені розробники, так і новачки, які вперше беруться до створення гри. Вся справа в тому, що UE4 за умовчанням підтримує відразу дві мови програмування: текстовий C++, в якому потрібно писати рядки коду, і візуальна мова Blueprints, в якій ігрова логіка вибудовується за допомогою блоків, пов'язаних між собою. Такий підхід допомагає зробити програмування наочнішим і зрозумілішим для тих, у кого немає досвіду.

Blueprints – це візуальна, нодова система програмування, яка використовується в Unreal Engine . За допомогою складання логічних блоків нодів, можна "зібрати" як з конструктора програму будь-якої складності, починаючи від простого клікера, закінчуючи повноцінною RPG-грою. Оскільки в блупринтах не використовується програмний код, то написати програму може кожен, хто розуміє основні принципи ООП.

За допомогою Блупринтів розробники можуть створювати такі речі, як:

- ігрові режими – встановлювати правила гри, змінювати поведінку гри в загальному плані;
- гравці – призначати гравців, надавати їм особливих рис і вигляду;
- камери – створення видів перегляду та змінення властивостей камер у реальному часі;
- управління – призначати кнопки для керування персонажем, автомобілем або зовнішнім рівнем;
- речі – зброя, предмети, що підбираються та інше;
- оточення – створення випадково генерованого оточення.

Unity 3D

Unity – більше, ніж движок, це середовище розробки комп'ютерних ігор, у якій об'єднані різні програмні засоби, використовувані під час створення ПЗ – текстовий редактор, компілятор, відладчик тощо. При цьому, завдяки зручності використання, Unity робить створення ігор максимально простим і комфортним, а мультиплатформеність движка дозволяє розробникам охопити якнайбільшу кількість ігрових платформ та операційних систем.

Насамперед, движок Unity3D дає можливість розробляти ігри, не вимагаючи для цього якихось спеціальних знань. Тут використовується компонентно-орієнтований підхід, у якого розробник створює об'єкти (наприклад, головного героя) і до них додає різні компоненти (наприклад, візуальне відображення персонажа та способи управління ним). Завдяки зручному Drag & Drop інтерфейсу і функціональному графічному редактору движок дозволяє малювати карти і розставляти об'єкти в реальному часі і відразу ж тестувати результат.

Цей движок також відомий як середовище розробки інді та мобільних ігор. Деякі статистичні дані: станом на вересень 2019 року 52% з 1000 найкращих мобільних ігор були створені на базі Unity, а також 60% всього контенту AR/VR, за даними компанії. Гравці в ігри, створені у Unity знаходяться в 195 країнах – це буквально кожна країна на планеті. Платформа для розробки в режимі реального часу дозволяє створювати 2D, 3D, VR і AR-контент і досягла 3 мільярдів пристроїв за останні 12 місяців.

Основна причина популярності Unity, особливо для невеликих проєктів: це безкоштовно. Персональна ліцензія движка дозволяє розробникам створювати комерційні ігри безкоштовно, за умови, що вони не заробили або не отримали більше 100 000 доларів США доходу або фінансування за останні 12 місяців.

Unity дозволяє легко створювати інтерактивний 3D-контент. Цей ігровий движок сьогодні вибирають багато великих організацій через його відмінну функціональність, високоякісний контент і можливість використання для будь-

якого типу гри. Він підтримує як 2D-, так і 3D-контент.

Unity – це більше ніж ігровий движок, це середовище для розробки ігор на ПК та мобільних ігор, в якому об'єднанні різні програмні компоненти, які використовуються при створенні програмного забезпечення.

Перевагою Unity є те, що ти можеш почати розробляти власний програмний продукт не маючи за плечами великого досвіду розробки. Тобі не потрібні фундаментальні знання для того, щоб почати творити свій продукт. В Unity використовується компонентно-орієнтований підхід, в рамках якого ви створюєте якийсь об'єкт, а потім в порядку необхідності можете його кастомізувати. Другим великим плюсом Unity є великий вибір плагінів та асетів за допомогою яких можна сильно пришвидшити розробку.

Третім плюсом Unity є кросплатформеність – це підтримка великої кількості платформ, технологій та інтерфейсів користувача.

Ось представлення головне вікно при відкритті платформи Unity 3D

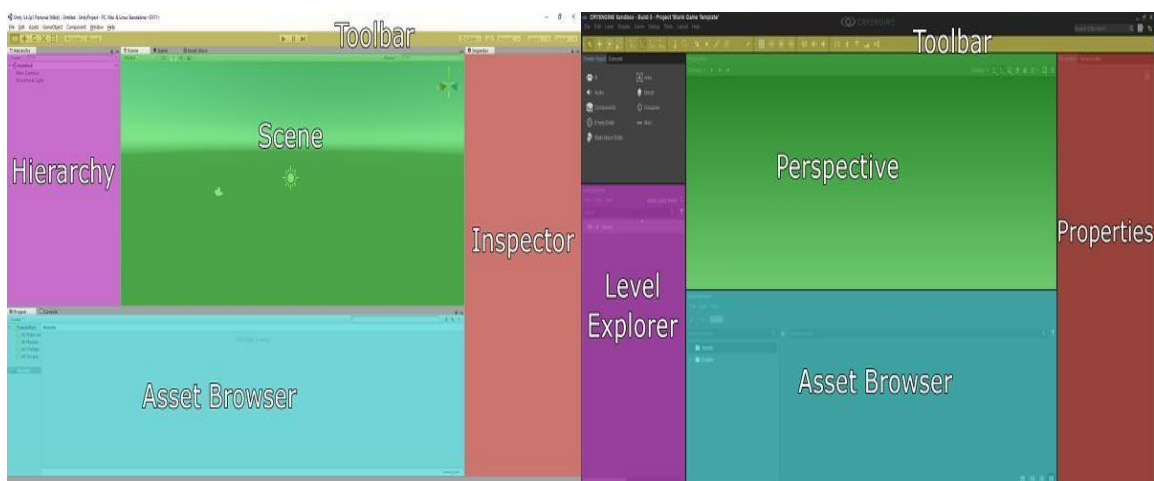


Рис. 1.29 Інтерфейсне вікно Unity3D

Висновки до першого розділу

Проведене дослідження Brick Ball і на основі побаченого було закріплено знання продукту, для подальшого створення гейплею гри з нейромережею та реалізацією функцій які можна додати до продукту щоб реалізувати недоліки даного продукту. Для кращого створення гри, проаналізовано платформи створення та мови написання, і на основі теоретичних знань можна починати

формувати ідею продукту.

Також із інструментів що вдалося дослідити краще за все нам підходить Unity 3D, з урахування написання коду на мову програмування C#.

РОЗДІЛ 2. ПРОЕКТУВАННЯ ТА РОЗРОБКА ВЕБ-ДОДАТКУ

2.1. Аналіз реалізації системи використання продукту та вихідних даних

2.1.1. Діаграма використання продукту користувачем

За даною діаграмою ми можемо спостерігати які можливості користувача нам надані. На ній видно всі дії які може виконати користувач, і до якого результату кожна з них його приведе. На даній діаграмі ми використовуємо одного користувача, щоб зрозуміти його повні можливості.(Рис. 2.1)

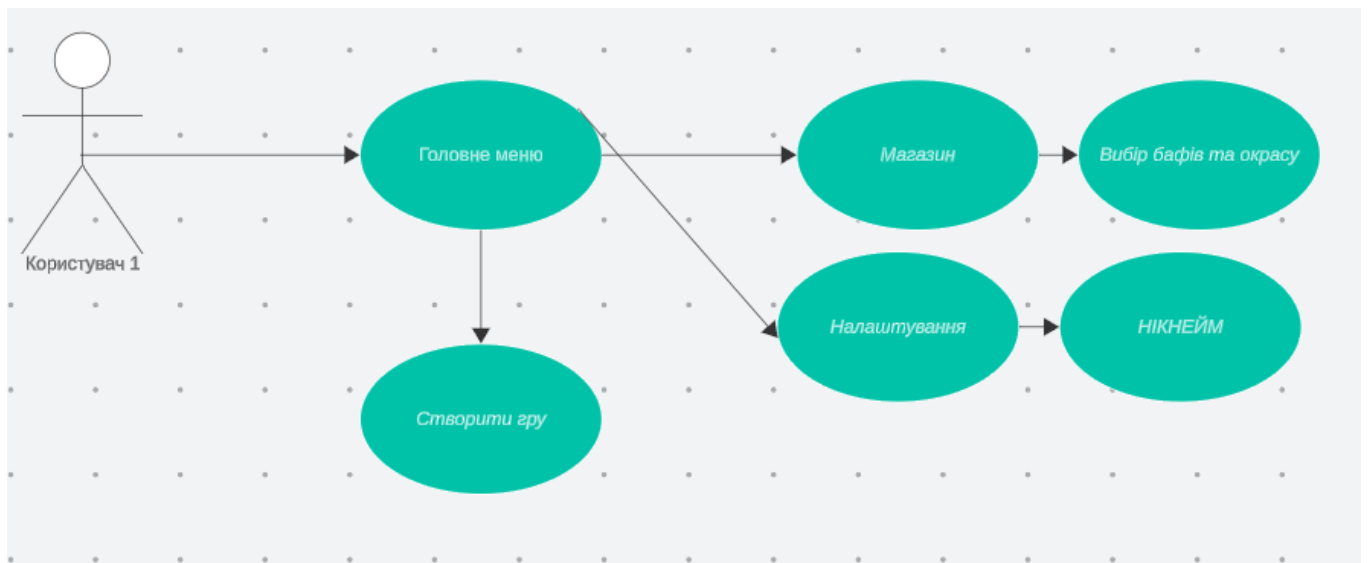


Рис. 2.1 – Діаграма головного меню і його функцій

Якщо брати інформацію, якою вже заволоділи то можна діаграму розподілити на два типи мультиплеєр і ні, Рис.2.1- не мультиплеєрна, адже там один користувач тільки задіяний, а на Рис.2.2- вже задіяно двоє користувачів і гра може запускатись в мультиплеєрі.



Рис. 2.2 – Діаграма запуску мультиплеєрного режиму

На рисунку 2.2 зображена діаграма коли два користувачі запустили додаток, перейшовши, до кнопки за пуску і кнопки приєднатися, користувачі автоматичному перейдуть безпосередньо до гри.

Якщо будь який з користувачів, не виконає умову поставлену так як в діаграмі, то кінцевий результат не зможе буде реалізованим. Також є умова що користувач може брати із собою тільки 3 посилення, і двоє користувачів одночасно не можуть вибрати однаковий окрас.

2.1.2. Діаграми класів

Основними класами є AppInstaller, UIService та IOHelper. AppInstaller – клас, який ініціює основні компоненти гри та реєструє їх в контейнері залежностей DI.

```

1  using Diploma.Application.Providers;
2  using UnityEngine;
3  using Zenject;
4
5  namespace Diploma.Application
6  {
7      public class AppInstaller : MonoInstaller
8      {
9          [SerializeField] MonoBehaviour[] autoBind;
10
11         public override void InstallBindings()
12         {
13             Container.
14                 Bind<AssetProvider>().
15                 FromNew().
16                 AsSingle().
17                 CopyIntoAllSubContainers().
18                 NonLazy();
19
20             for(var x = 0; x < autoBind.Length; x++)
21             {
22                 Container.
23                     Bind(autoBind[x].GetType()).
24                     FromInstance(autoBind[x]).
25                     AsSingle().
26                     //CopyIntoAllSubContainers().
27                     NonLazy();
28             }
29         }
30     }
31 }

```

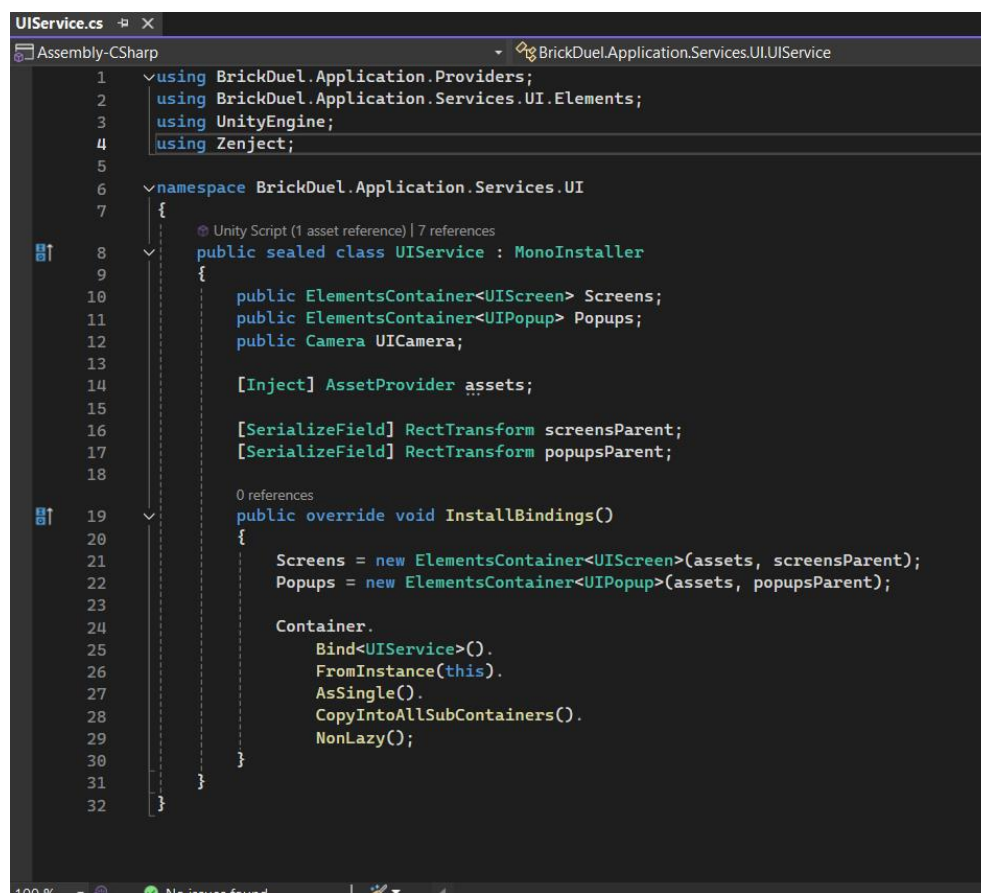
Рис. 2.3 – AppInstaller

Addressables - це система управління ресурсами в Unity, яка дозволяє ефективно керувати завантаженням та розподілом ресурсів у вашому проекті. Основна ідея Addressables полягає в тому, щоб дозволити вам завантажувати, вивантажувати та керувати ресурсами, такими як тексти, зображення, аудіо, префаби тощо, незалежно від того, як вони розміщені на диску чи як вони організовані у вашому проекті Unity.

Основні переваги Addressables включають:

Ефективне управління пам'яттю: Addressables дозволяють завантажувати ресурси в пам'ять лише тоді, коли вони потрібні, та вивантажувати їх при необхідності, що допомагає економити пам'ять вашого додатку. Динамічне завантаження ресурсів: Ви можете завантажувати ресурси асинхронно під час виконання програми, що дозволяє створювати більш гнучкі та динамічні додатки. Гнучке управління версіями та оновленнями ресурсів: Addressables дозволяють легко керувати версіями ресурсів та їх оновленнями, що робить робочий процес оновлення вашого додатку більш простим та ефективним. Можливість роботи з великими обсягами ресурсів: Addressables дозволяють легко керувати великими

обсягами ресурсів у ваших проектах, що дозволяє підтримувати високу продуктивність та швидкодію вашого додатку



```
1 using BrickDuel.Application.Providers;
2 using BrickDuel.Application.Services.UI.Elements;
3 using UnityEngine;
4 using Zenject;
5
6 namespace BrickDuel.Application.Services.UI
7 {
8     public sealed class UIService : MonoInstaller
9     {
10         public ElementsContainer<UIScreen> Screens;
11         public ElementsContainer<UIPopup> Popups;
12         public Camera UICamera;
13
14         [Inject] AssetProvider assets;
15
16         [SerializeField] RectTransform screensParent;
17         [SerializeField] RectTransform popupsParent;
18
19         public override void InstallBindings()
20         {
21             Screens = new ElementsContainer<UIScreen>(assets, screensParent);
22             Popups = new ElementsContainer<UIPopup>(assets, popupsParent);
23
24             Container.
25                 Bind<UIService>().
26                 FromInstance(this).
27                 AsSingle().
28                 CopyIntoAllSubContainers().
29                 NonLazy();
30         }
31     }
32 }
```

Рис. 2.4 – UIService

"UIService" це зазвичай клас або компонент у програмному коді, який використовується для управління елементами інтерфейсу користувача (UI) у вашому додатку чи грі. Його основна функція - забезпечити легкий доступ до різних елементів UI, їх створення, зміну та взаємодію з ними. Ось деякі загальні функції, які може виконувати UIService:

Створення та управління UI елементами: Це може включати створення нових вікон, кнопок, текстових полів та інших елементів, а також розміщення їх на екрані.

Завантаження та управління ресурсами UI: UIService може брати на себе завдання завантаження зображень, шрифтів, анімацій та інших ресурсів, необхідних для побудови UI, та їх управління

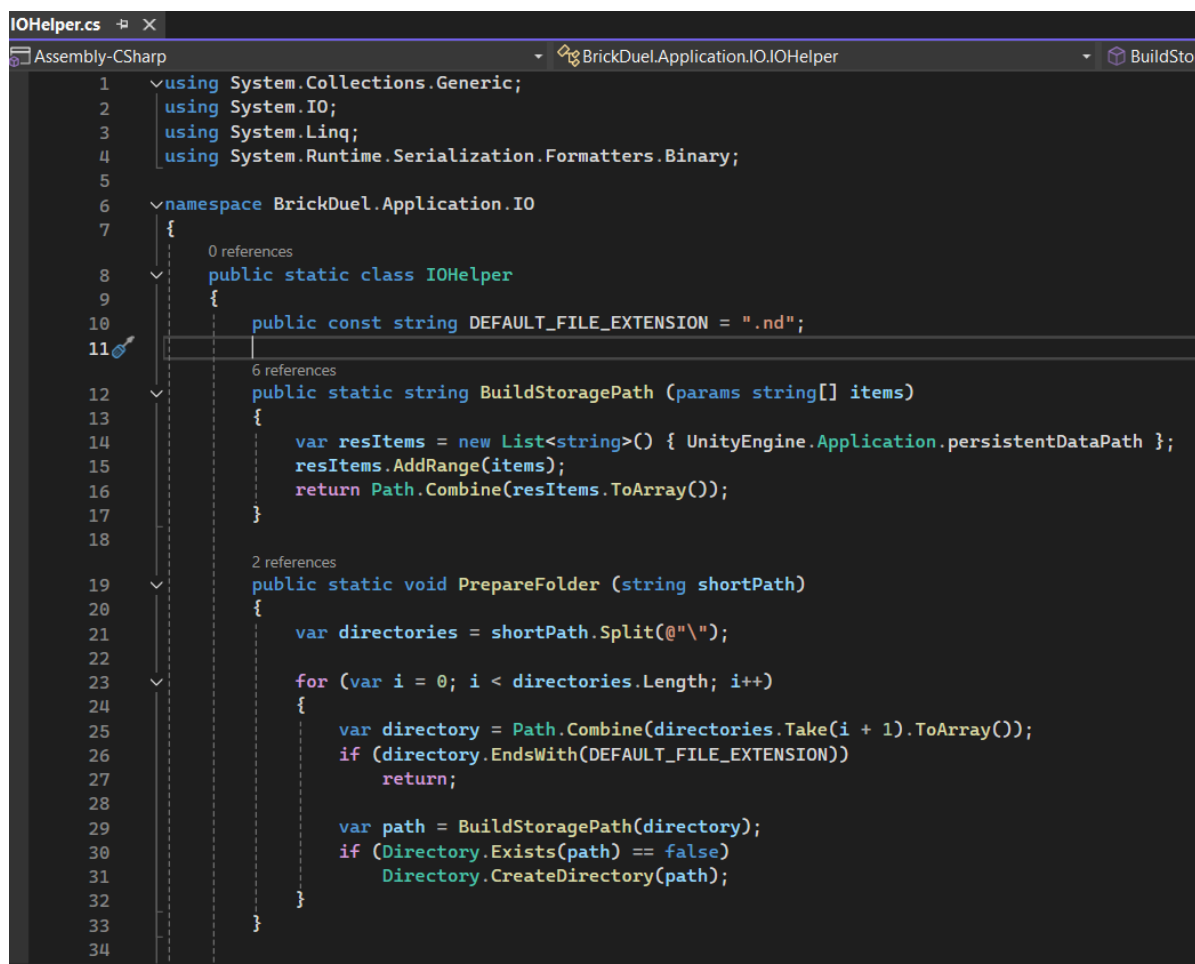


Рис. 2.5 – IOHelper

IOHelper може містити методи для читання та запису даних до файлів на диску. Це може бути корисно для збереження та завантаження конфігураційних файлів, текстових файлів, зображень тощо.

Робота з потоками даних: IOHelper може надавати функції для створення, читання та запису потоків даних. Це може включати роботу з потоками вводу/виводу, мережевими потоками тощо.

Операції з директоріями та файлами: IOHelper може включати методи для створення, видалення, переміщення та перейменування файлів та директорій.

Конвертація та обробка даних: IOHelper може надавати допоміжні функції для конвертації даних з одного формату в інший, обробки даних та перевірки їх правильності.

Робота з ресурсами за межами файлової системи: Окрім роботи з файлами, IOHelper також може включати функції для роботи з іншими джерелами даних,

такими як бази даних, веб-сервери, API тощо.

Загальна мета IOHelper полягає в тому, щоб спростити та уніфікувати операції вводу/виводу в програмному коді, що дозволяє розробникам ефективніше працювати з файловою системою та даними. За поведінку гравця відповідає клас PlayerController. Він передає керування конкретним персонажем на клавіатуру та мишку.

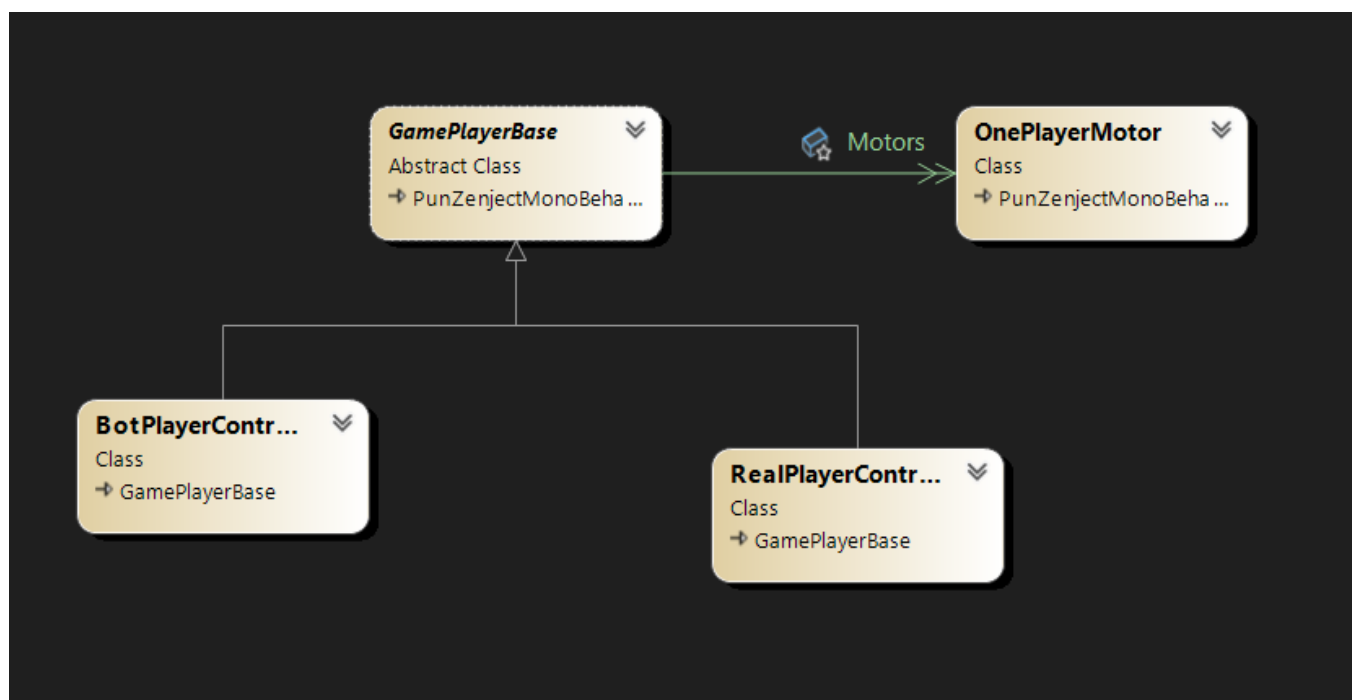


Рис. 2.6 – Діаграма класів Player

Діаграма Player дає змогу побачити управління над персонажем, методи переміщення гравця по ігровому світу, виконання дій, таких як атака, захист збирання предметів відступ по чистому полю.

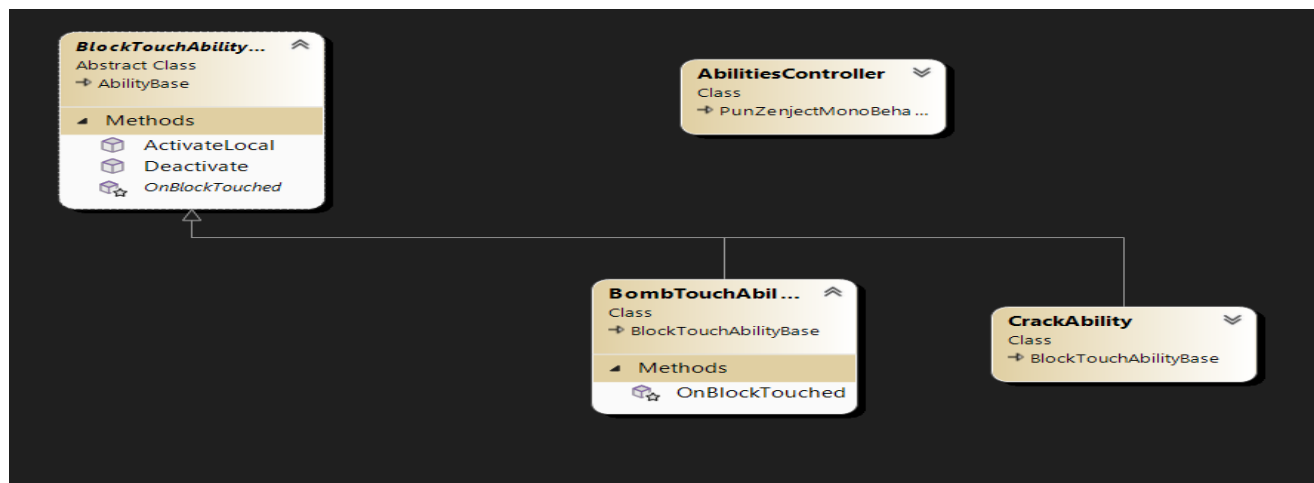


Рис. 2.7 – Діаграма класів Abilities

Діаграма Abilities показує типи здібностей їх зміна та контролер.

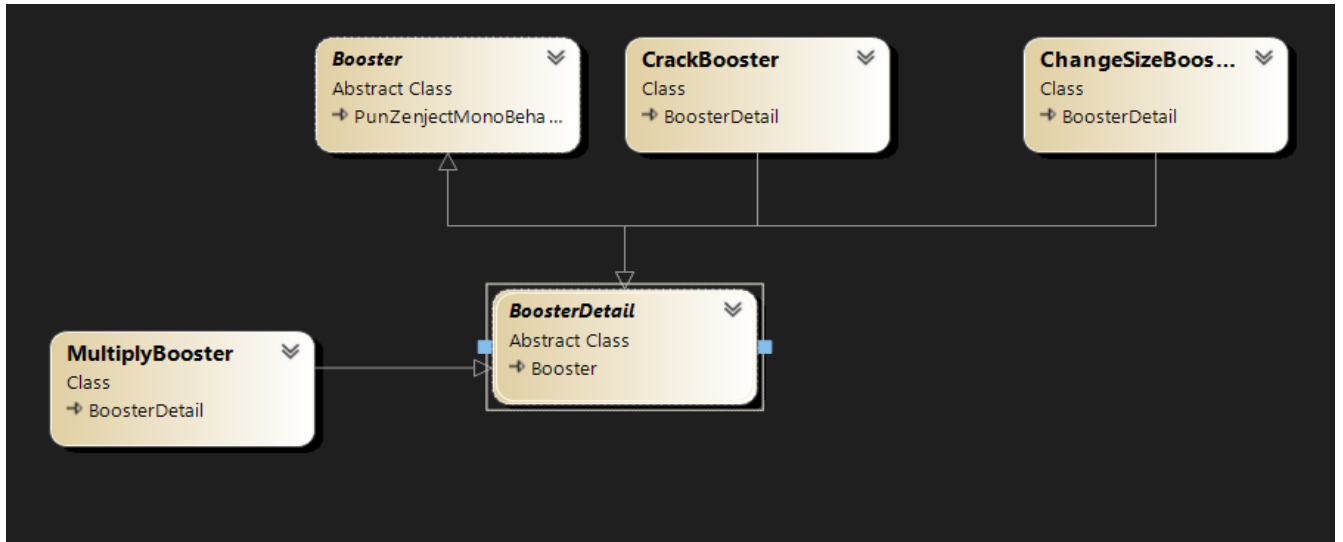


Рис. 2.8 – Діаграма класів Boosters

Діаграма показує **Booster** використовується для детального аналізу та відображення інформації про бустери, які посилюють можливості гравця

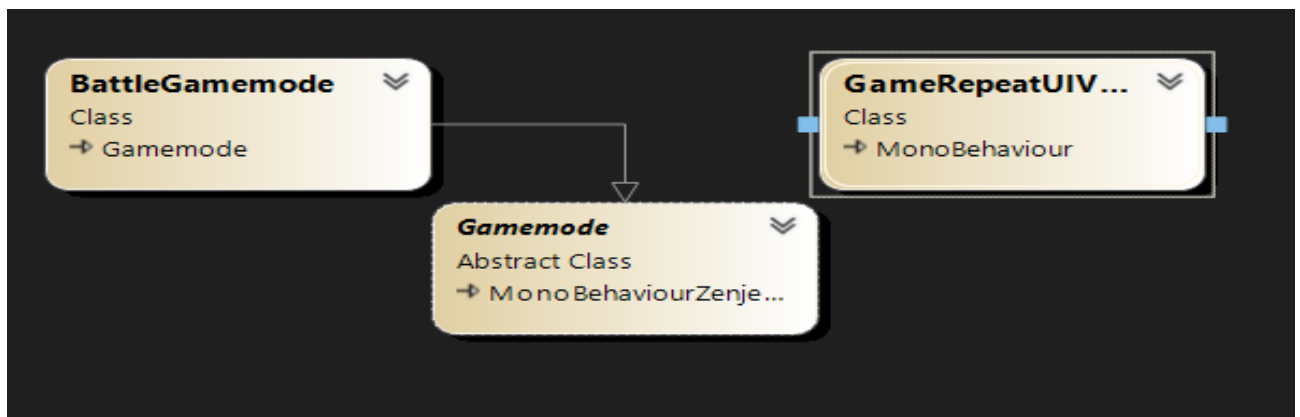


Рис. 2.9 – Діаграма класів Gamemodes

Діаграма Gamemodes показує вивід повідомлень про початок та завершення гри

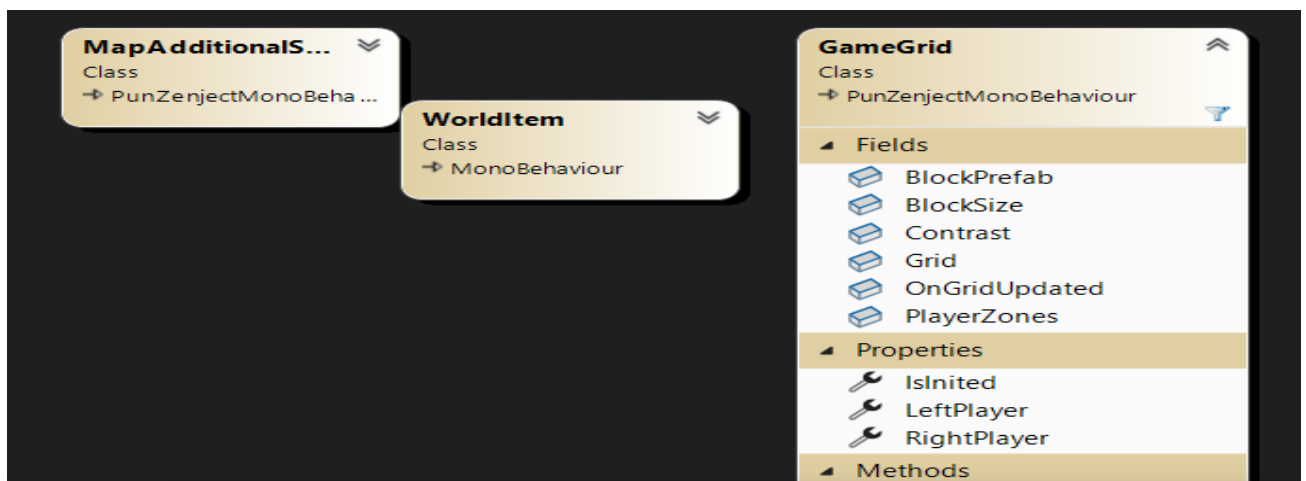


Рис. 2.10 – Діаграма класів Maps

Діаграма Mars показує, що відбувається на мапі, появлення додаткових можливостей, предмети які під час гри будуть з'являтися на полі бою, блоки кожного з персонажів та їх розмір, зони дії кожного з персонажів.

2.2. Представлення дизайн документу

Розробка та геймплей гри

Жанром гри обрано казуальні відеоігри. Казуальні відеоігри — відеоігри, призначені для широкого кола користувачів. Термін «казуальні» походить від латинського слова *casualis*, що означає — випадковий. Таким чином казуальні ігри — це ігри, в які грають від випадку до випадку, мимохідь, найчастіше — щоб якось «вбити» час.

На старті гри граці з'являються на двох рівно розподілених половинах, які можна назвати базою кожного з них, божна база буде покрашена в колір який гравець вибере заздалегіть в магазині головного меню.

В кожного гравця є дві задачі:

1. Захопити всі пікселі території суперника як омога швидше та ефективніше.
2. Не дати супернику знищити свою територію, адже буде зарахована поразка.

Для цього необхідно, думати на декліка кроків в перел свого суперника

Важливо розуміти як правильно атакувати половину ворога, та вмінням швидко адаптуватися до ситуації нападати обдуманим шляхом, та реалізовувати помилку суперника на свою користь.

Ігри цього жанру зазвичай не є складними у реалізації, але, не зважаючи на це можуть затягнути гравця на довгі години гри та зацікавити досить широку аудиторію. Але перевагою даного продукту виступає мультиплеєр, так як гравцям будь якої вікової категорії хочеться позмагатися одне з одним, і ця гра є кращим рішенням для такої дії. Нейромережа генерує так, що один і той же гравець не зможе після серії перемог здобувати великі переваги, не рахуючі грошові поїти.

Коли один з гравців перемагає, все повертається до початку.

2.2.1. Сцени та їх вигляд

Наші події розвивають на одній локації, але локація може виглядати по різному. В залежності від того, який колір виберуть користувач 1 і користувач 2, карта буде окрашена дзеркально від викору кожного користувача.

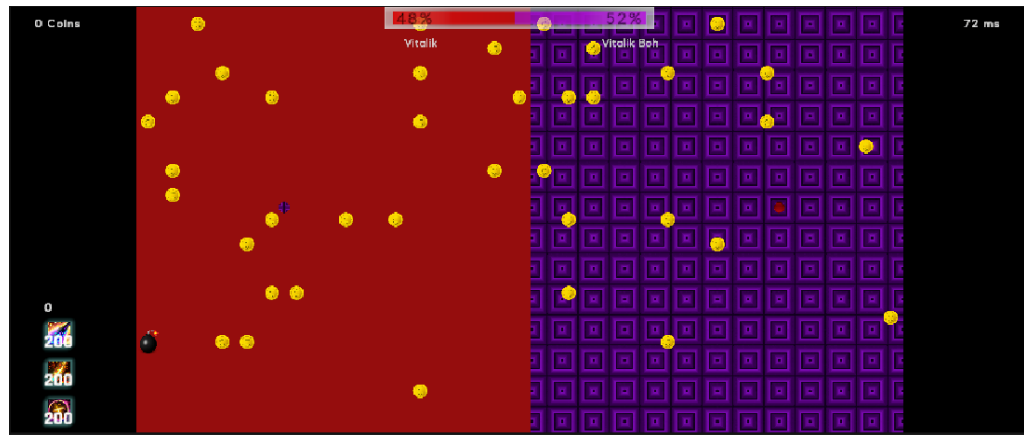


Рис. 2.11 – Вигляд поля аркади, від початку гри



Рис. 2.12- Користувач 1 фіолетовий піксель

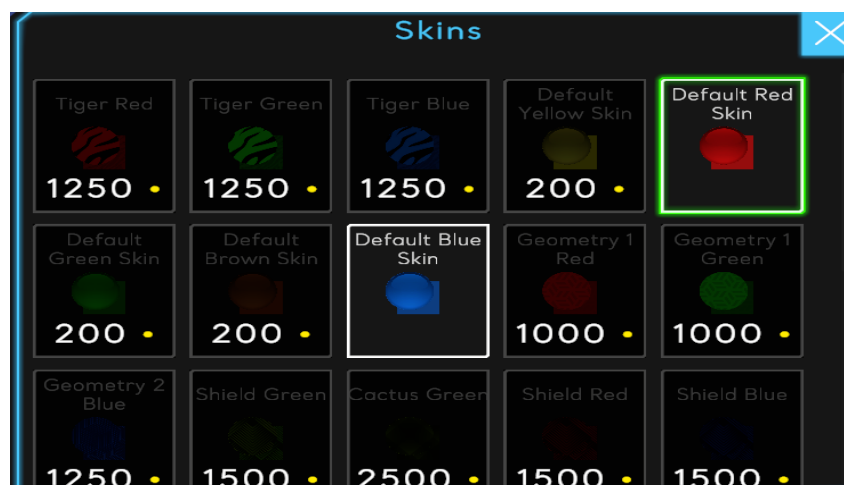


Рис. 2.13 – Користувач 2 вибрав тигрою піксель

Також важливою частиною є **Внутрішня ігрова логіка**:

Для даної гри у якості ігрової платформи обрано ПК тому що це найзручніша платформа для одночасного створення і тестування гри. А також комп'ютерні ігри все ще найпопулярніші серед усіх існуючих платформ.

Майже зображень повинні мати кожну гру можна представити як одну основну функцію, яка містить всю ігрову логіку і яка запускається або тоді, коли користувач щось робить, або після закінчення певного часу. Цей цикл запуску однієї і тієї ж основної функції знову і знову називається ігровим циклом.

2.2.2 Гейплей користувачів, текстури арени, підготовка графічних елементів.

Усі графічні елементи гри представлені у вигляді зображень png формату. PNG (Portable Network Graphics) – растровий формат збереження графічної інформації, що використовує стиснення без витрат. Розміри зображень повинні мати один за таких розмірів: 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048 пікселів. На цьому етапі розробки створюються такі графічні елементи, як фон, елементи платформ, зовнішній вигляд персонажів та перешкод, елементи інтерфейсу та логотип гри. Для подальшої анімації об'єктів, необхідно відмалювати всі можливі їх стани.

В даній таблицю представлення 2D- моделі, що застосовуються у грі



Рис. 2.14- Зображення частини з якої складається фон.

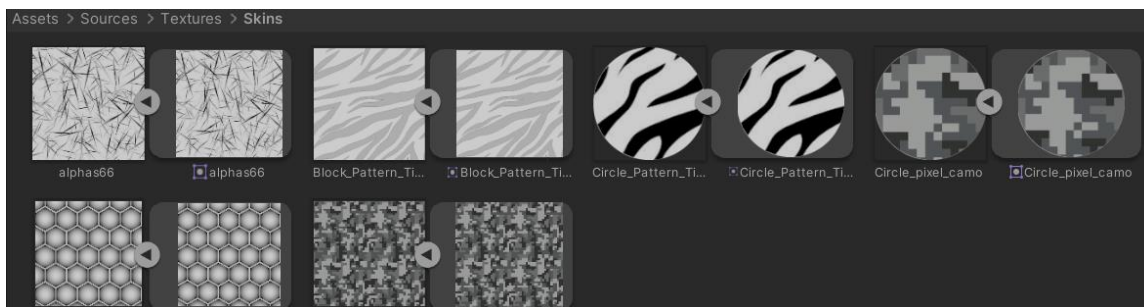


Рис. 2.15 – Зображення скінів мячів, якими Користувач виконує дії.

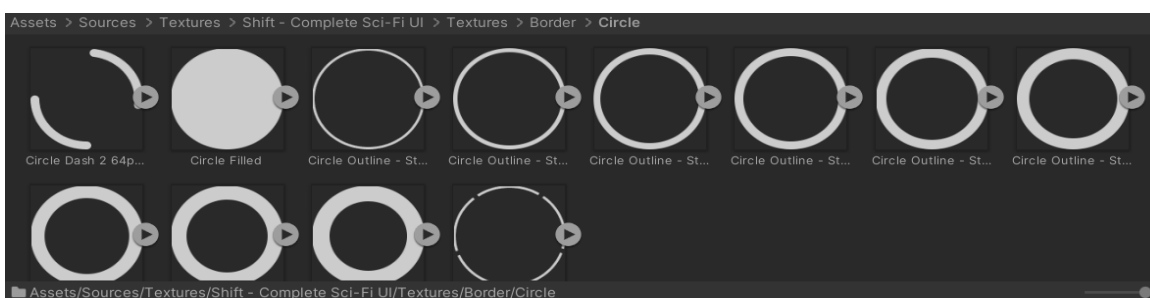


Рис. 2.16 – Зображення текстури мячів

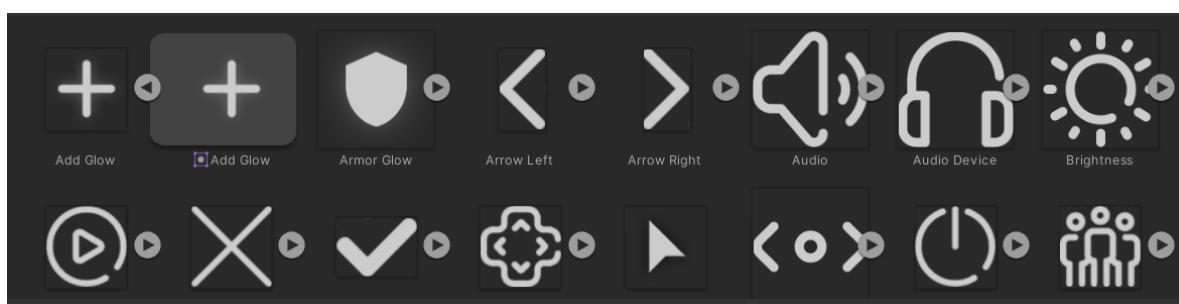


Рис. 2.17 – Зображення іконок ефектів посилення.

Управління персонажем та аналіз його функцій:

Користувач може рухатись як кліком так і протягнути курсор, для направлення м'ячика в тому чи іншому напрямку, також посилені ефектики гравець може використовувати за допомогою кнопок 1, 2, 3 в послідовності зверху до низу.

2.3. Процес реалізація програмного коду

На даному етапі ми звернемося до частини програмного коду яка дасть нам змогу зрозуміти, як був реалізований наш продукт.

Першим у нас показано реалізацію AbilitiesController. Відповідає стабілізацію ігрового процесу, задає певні рамки, в яких буде працювати гра.

Лістинг коду:

```
namespace BrickDuel.Game.Abilities
{
    public class AbilitiesController : PunZenjectMonoBehaviour
```

```

{
    public const int MAX_POINTS_USED = 800;
    public const int MAX_RANK = 3;

    public int[] RankPricing = new int[MAX_RANK] { MAX_POINTS_USED / 3, MAX_POINTS_USED / 2,
MAX_POINTS_USED };
    public Value<int> CurrentRank = new Value<int>();

    public Slot[] slots;

    public UnityAction<string> OnAbilityActivate;

```

Далі ми бачимо наведено скрипт **BoosterDetail**, використовується для детального аналізу та відображення інформації про бустери у даній грі.

Лістинг коду:

```

namespace BrickDuel.Game.Boosters
{
    public abstract class BoosterDetail : Booster
    {
        [Range(0, 15)]
        public float DestroyTimer = 0f;
        public GameObject Graphics;

        public override void Activate(OnePlayerMotor player)
        {
            base.Activate(player);

            Graphics?.SetActive(true);
        }
    }

```

Далі ми бачимо скрипт **MultiplyBooster**, він відповідає за розповсюдження ефектів посилення по карті, для двох користувачів.

Лістинг коду:

```

using BrickDuel.Game.Players;
using UnityEngine;

namespace BrickDuel.Game.Boosters
{
    public class MultiplyBooster : BoosterDetail
    {
        [SerializeField, Range(1, 5)] int amount = 2;
        [SerializeField] float copyTimelife = 5f;
    }

```

```

public override void Activate(OnePlayerMotor player)
{
    base.Activate(player);

    player.Owner.SpawnMotors(amount, transform.position, copyTimelife);
}
}
}

```

Далі ми бачимо скрипт `NetworkPlayerExtensions`, включає в себе функцію керування діями користувачів, керування рухом та взаємодією з об'єктами.

Лістинг коду:

```

namespace BrickDuel.Game.Extensions
{
    public static class NetworkPlayerExtensions
    {
        public static void SetPlayerParameters(Hashtable entries)
        {
            PhotonNetwork.LocalPlayer.SetCustomProperties(entries);
        }
        public static void SetPlayerReady(this Player player, bool isReady)
        {
            player.SetCustomProperties(new Hashtable() { { "IsReady", isReady } });
        }
        public static bool GetPlayerReady(this Player player)
        {
            return (bool)player.CustomProperties["IsReady"];
        }
        public static void SetPlayerColor(this Player player, Color color)
        {
            player.SetCustomProperties(new Hashtable() {
                { "ColorR", color.r },
                { "ColorG", color.g },
                { "ColorB", color.b }
            });
        }
    }
}

```

Тепер черга дійшла до скрипта `GameGrid`, він виконаний для створення ігрової сітки і поля на якій розташовані елементи, об'єкти та персонажі гри.

Лістинг коду:

```

namespace BrickDuel.Game.Map

```

```

{
    public class GameGrid : PunZenjectMonoBehaviour
    {
        [Range(0.15f, 3f)]
        public float BlockSize = 0.5f;
        [Range(0f, 1f)]
        public float Contrast = 0.5f;
        public GameObject BlockPrefab;

        public Vector2Int FieldSize = new Vector2Int(35, 35);

        [TableMatrix]
        public GridCell[,] Grid;
        public Transform[] PlayerZones;

        public bool IsInitied { get; private set; }
        public GamePlayerBase LeftPlayer => gamemode.LeftPlayer;
        public GamePlayerBase RightPlayer => gamemode.RightPlayer;

        public UnityAction<Vector2Int> OnGridUpdated;

        [SerializeField]
        private Gamemode gamemode;
        private MapAdditionalSpawner spawner;
        [SerializeField]
        private int spawnItemID = 0;
        [SerializeField]
        private float errosionRate = 3f;
        [SerializeField]
        private float chanseSpawnOnChangeOwner = 0.15f;
    }
}

```

Далі ми розглянемо скрипт BotPlayerController, він ангоритми штучного інтелекту, такі як алгоритм прийняття рішення, пошук шляху, обчислення території

Лістинг коду:

```

using UnityEngine;

namespace BrickDuel.Game.Players
{
    public class BotPlayerController : GamePlayerBase
    {
        public Vector2 SpeedThrow = new Vector2(0.25f, 5f);
        public float ChangeThrowSide = 0.5f;
        public float UpdateRate = 3f;
    }
}

```

```

private float lastUpdateTime;

public override void GameUpdate()
{
    if (photonView.IsMine == false)
        return;
    if (Time.time - lastUpdateTime > UpdateRate)
    {
        Throw(new Vector2(Random.Range(-1f, 1f), Random.Range(-1f, 1f)), Random.Range(SpeedThrow.x, SpeedThrow.y));
        lastUpdateTime = Time.time;
    }
}

```

Ще один з важливих скриптів є `GamePlayerBase`, він дає змогу управління над персонажем, методи переміщення гравця по ігровому світу, виконання дій, таких як атака, захист збирання предметів відступ по чистому полю і так далі. Лістинг коду:

```

namespace BrickDuel.Game.Players
{
    [RequireComponent(typeof(AbilitiesController))]
    public abstract class GamePlayerBase : PunZenjectMonoBehaviour
    {
        [ShowInInspector]
        public int UniqueID =>
            photonView.OwnerActorNr - 1;
        public int MaxMotors = 5;
        public Color TeamColor
        {
            get
            {
                var player = PhotonNetwork.CurrentRoom.GetPlayer(UniqueID + 1);
                if (player.CustomProperties.ContainsKey("ColorR"))
                {
                    return player.GetPlayerColor();
                }
                return Color.cyan;
            }
        }
        public Value<int> Coins = new Value<int>();
        public Value<int> RankedPoints = new Value<int>();
        public AbilitiesController Abilities { get; private set; }
        public LayerMask MapMask;
    }
}

```

Давайте розглянемо досить цікавий скрипт завершення GameResultScreen, він відповідає за відображення результатів гри чи певного подійного екрану після завершення гри. Мова йде про екран на яку користувачеві показують його досягнення, статистику гри

Лістинг програми:

```
using BrickDuel.Application;
using BrickDuel.Application.Services.UI.Elements;
using BrickDuel.Game.Players;
using BrickDuel.Lobby.UI;
using TMPro;
using UnityEngine;
using UnityEngine.SceneManagement;

namespace BrickDuel.Game.UI
{
    [AssetPath("UI/Screens/Game/Result")]
    public class GameResultScreen : UIScreen
    {
        [SerializeField] TextMeshProUGUI statusText;
        [SerializeField] TextMeshProUGUI coinsCounter;
        [SerializeField] TextMeshProUGUI playerTitle;

        public void SetInfo(GamePlayerBase localPlayer,
            bool localWinner,
            int coinsCollected)
        {
            if (localPlayer != null)
            {
                statusText.text = localWinner ? "Win" : "Defeat";
                coinsCounter.text = $"{{coinsCollected}} coins";
            }
        }

        public async void GoToMenu()
        {
            await UI.Screens.CreateElement<LobbyScreen>();
            SceneManager.LoadScene(0);
        }
    }
}
```

Висновки до другого розділу

В ході написання другого розділу дипломної роботи були проаналізовані можливості і причини вибору програмного забезпечення для розробки дипломного проекту, були розглянуті функціональні можливості вибраного програмного забезпечення та проведено порівняння зі схожими аналогами, також було розглянуто покрокове налаштування кожного з етапів розробки та налаштування проекту.

Також були розглянуті мови програмування які могли використовуватись при написанні програмного коду проекту, відмінність роботи з та графікою та розглянуто порядок етапів і алгоритмів для розробки фінальної цілі дипломного проекту – створення власної 2D гри. Діаграма надає можливість нам побачити всі можливі варіації дій користувачем. Також була згенерована та реалізована таблиця класів. Проаналізували ідею розробки додатку. було активовано сцени безпосередньо гри, для перевірки готвності чи можлий запуск користувачем.

Проаналізована пиксельна частина, текстур та піксельні анімації з яких і створено арена для мільтиплерного кросплатформеного додатку.

РОЗДІЛ 3. ПОРЯДОК РОБОТИ ТА ТЕСТУВАННЯ ПРОГРАМНОГО КОМПЛЕКСУ

3.1. Покрокова робота та інтерфейс додатку.

Для початку треба зрозуміти, що опис роботи додатка – це представлення його графічної частини, а також покрокове представлення роботи, як це буде бачити користувач, коли буде працювати з даним продуктом. Після запуску додатку, користувач потрапляє на екран головного меню, яке оснащено кнопками, що відповідають за певні функції. Далі натискаємо кнопку «Play».

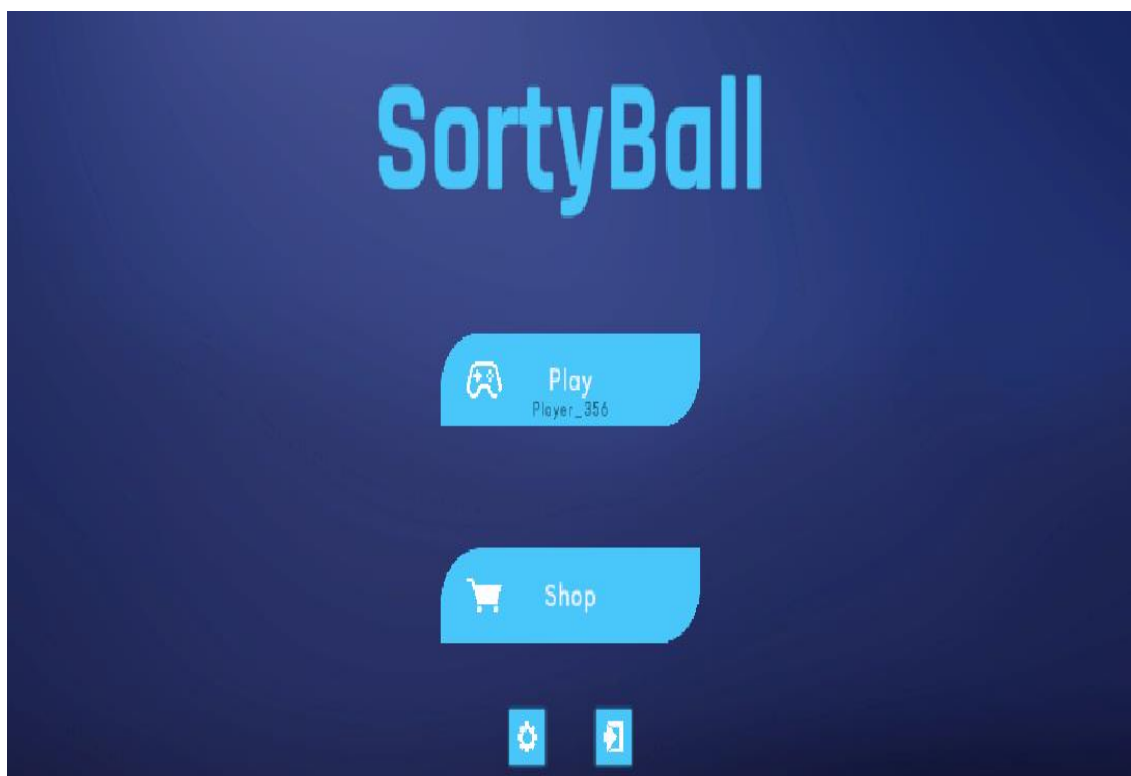


Рис. 3.1 – Головне меню гри

Після натискання кнопки «Play» користувач перейде до меню вибору, передним з'являється п'ять кнопок вибору продовження сценарію гри, «Create» - це буде означати, що користувач створить лоббі, для очікування наступного користувача, щоб гра могла розпочатися, «Join» - це буде означати, що користувач подасть позив на мультиплеєр, для приєднання в лоббі, щоб гра розпочалась. «Back» - натискання цієї кнопки поверне користувача до попереднього меню, якщо він наприклад не був готов розпочати, або поспішно натиснув кнопку мультиплеєра(Рис. 3.2).

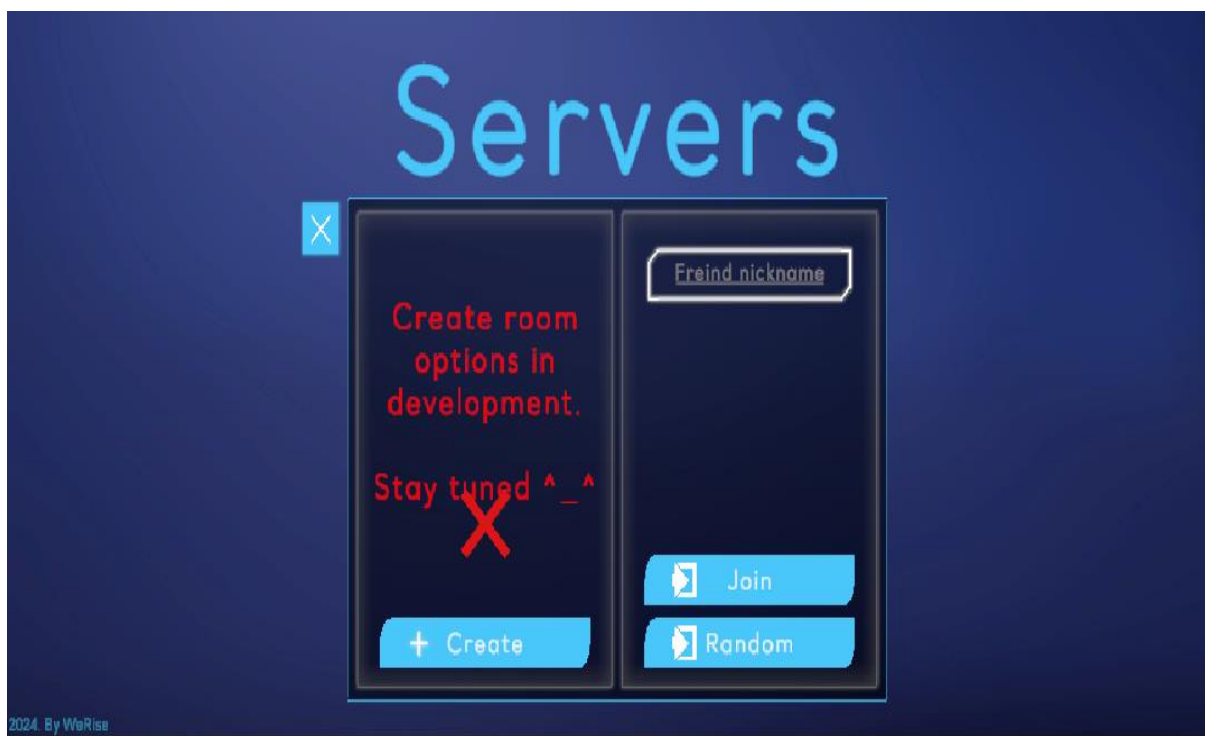


Рис.3.2- Мультиплеєрне меню для створення лоббі

Далі виникає запуск безпосередньо сцени, де гравці і зможуть позмагатися одне з одним . Користувачі побачать кожен свою і ворожу частину території, яку необхідно забрати способом хітріше ніж суперник. (Рис.3.3).

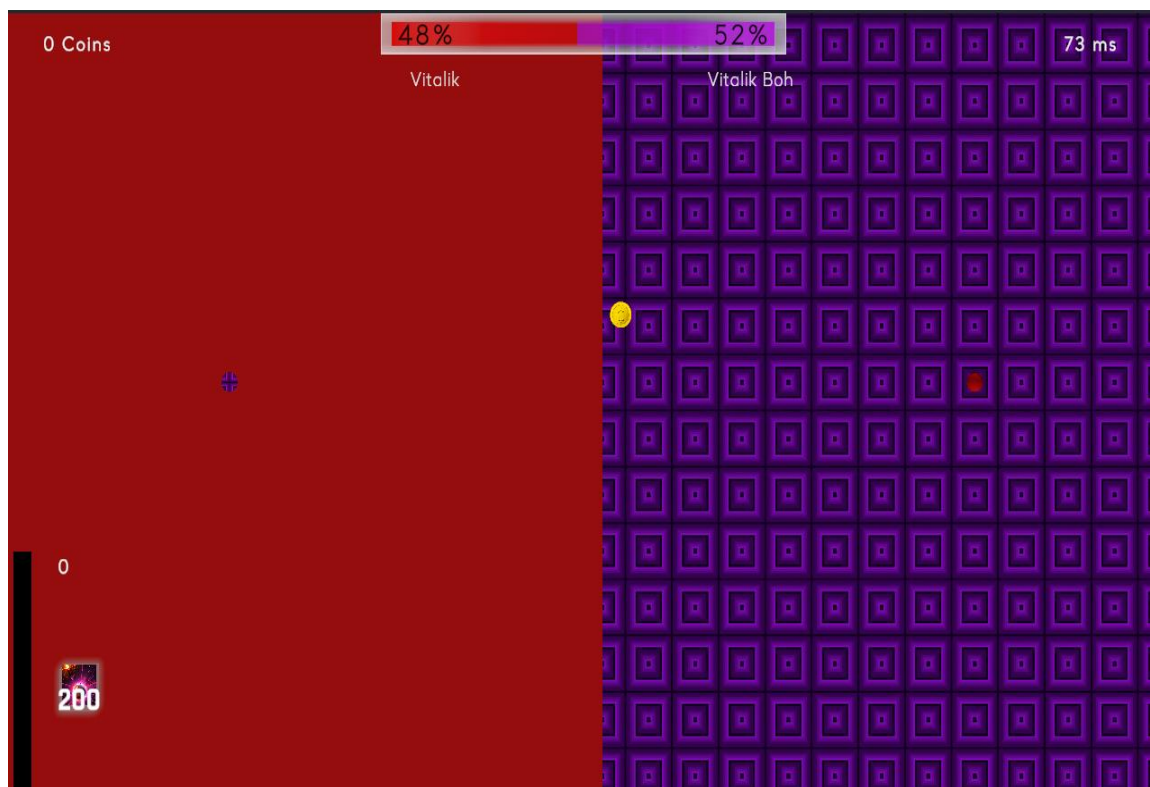


Рис. 3.3 – Початкове розміщення гравців на арені

Тепер давайте повернемося до магазину. Для того щоб туди потрапити з головного меню натискаємо на вкладку «Shop», після її натискання гравець потрапляє до вибору бонусних ефектів та окрасу території та персонажа (див. рис. 3.4).



Рис.3.4 – Вкладка здібностей



Рис. 3.5– Вкладка скінни.

Також є функція, яка працює при певних умовах, це функція активації ефектів бусту героя, коли число монет набирається від 250 і вище, можна скористатись одним з цих ефектів, після використання буду доступні лише два але вже за ціною 400, після другого використання буде доступно 1 за ціною 800, кожне використання подальше посилює наступний скіл, останній буде масимальної сили, але порядок виконання, користувач буде вибирати самостійно(Рис 3.4).

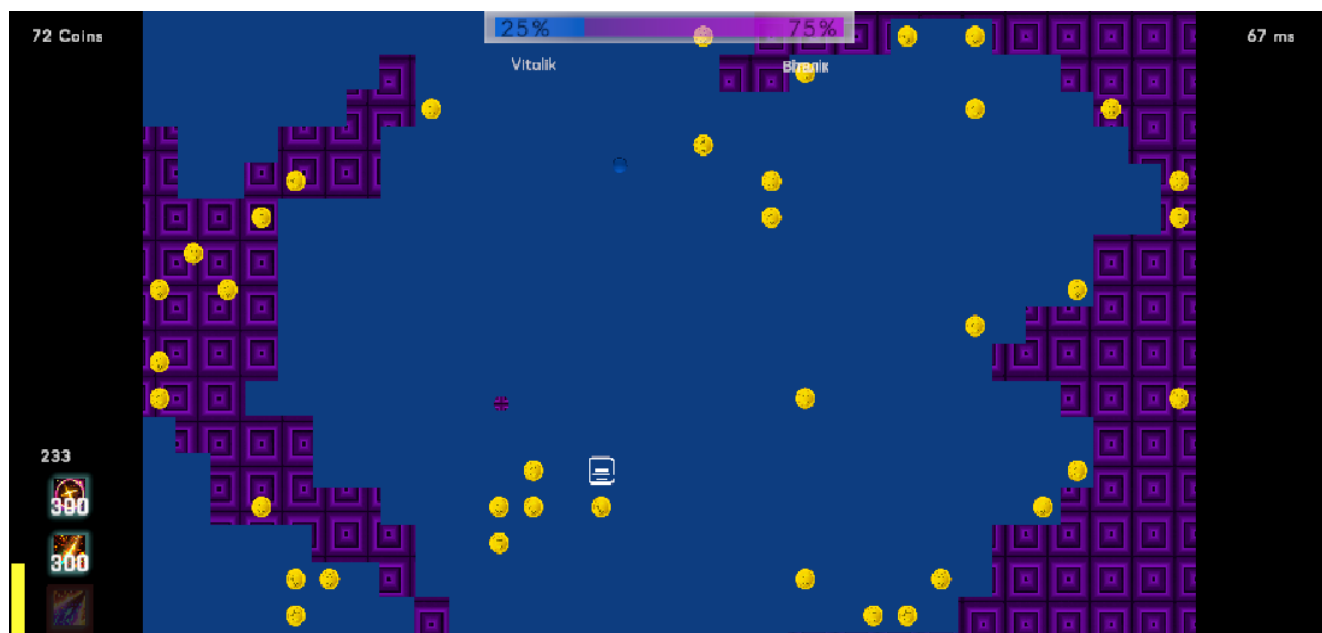


Рис. 3.6- Активоване пробиття подвоєне пробиття вожих пікселів

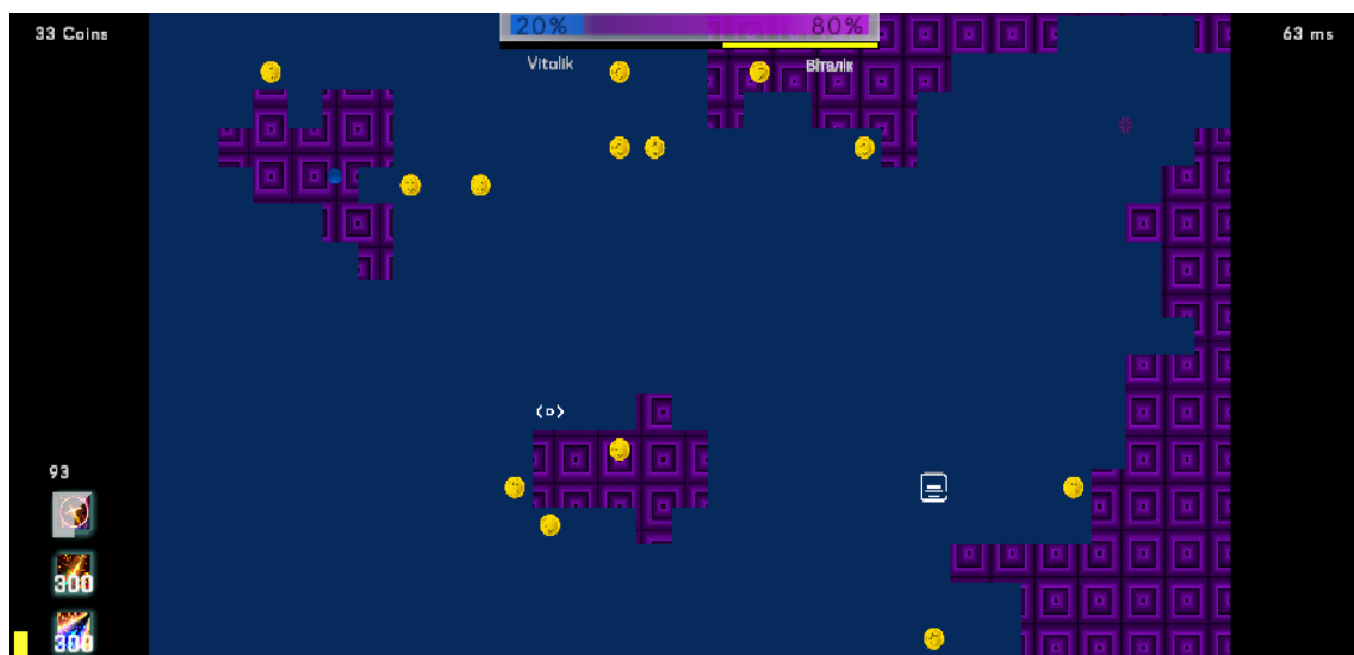


Рис.3.7- Відмерення відділених пікселів

Також є скріп який регулює помилку, виборі однакової анімації. Оскільки при виборі аніції в тей же окрас малюється карти противника, при однаковій анімації не буде видно захвату піселі і одному ні іншому користувачеві, при даній помилці обом користвачам відкриється магазин для перевибору анімації(рис. 3.8).



Рис. 3.8 – Помилка однакових скінів

Анімація при якій кожен з гравців бачив своє повідомлення, про результат гри.



Рис. 3.9. – Екран переможення

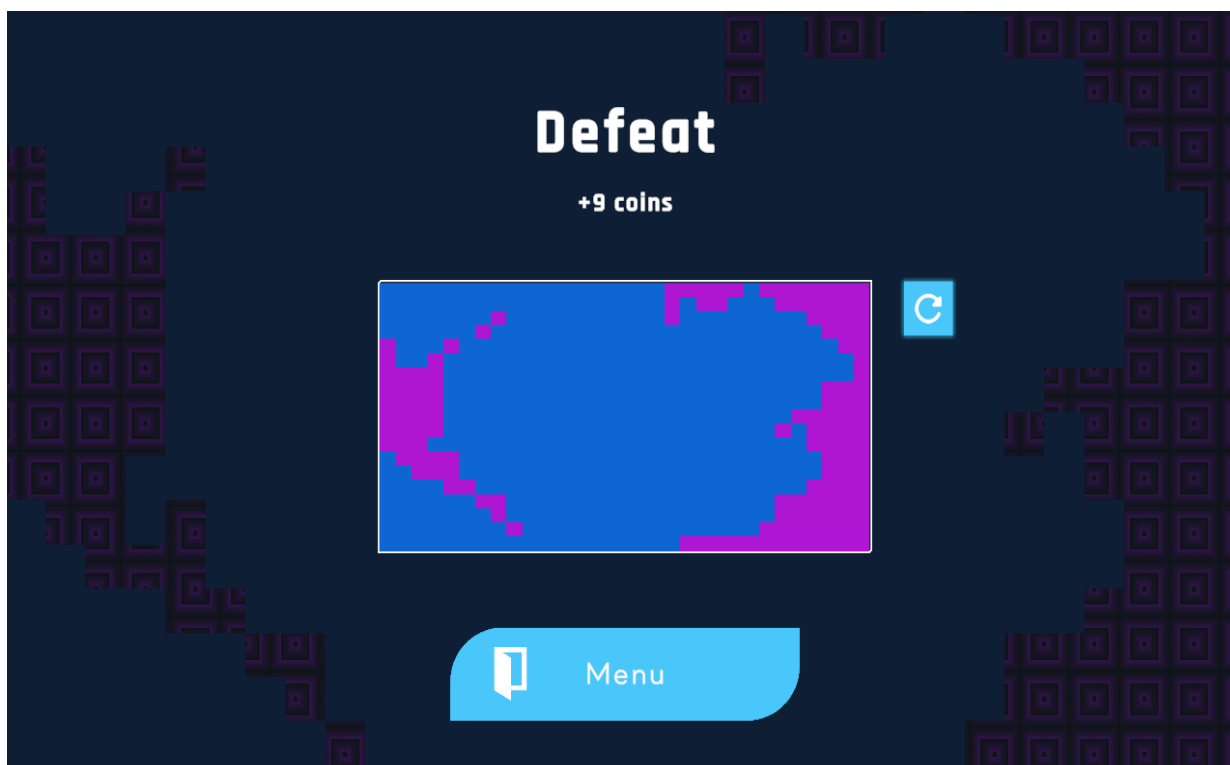


Рис.3.10- Екран програвшого

Після цього обидва користувачі можуть повернутися по доловного меню та спробувати зіграти ще раз.

Була створена система донатів для ЗСУ, за які ти отримуєш скіни для того, щоб додатково мотивувати користувачів.

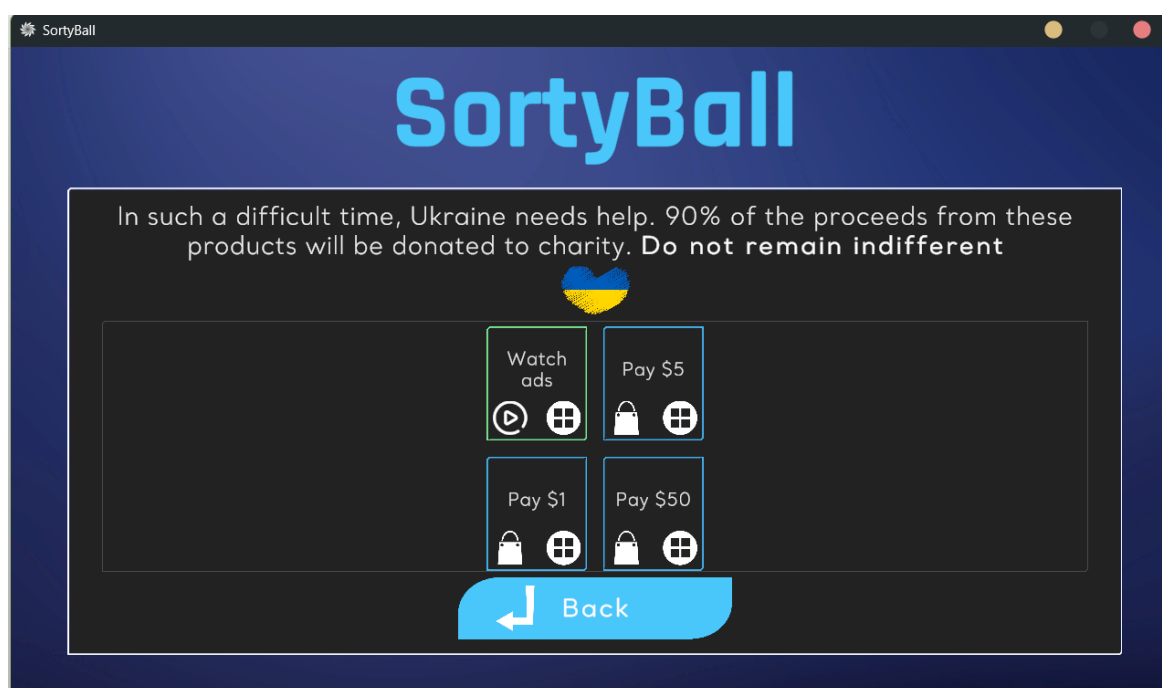


Рис.3.11 – Екран донатної дошки

Також після завершення гри користувачі можуть подивитися повтор того як змінювалась мапа на протязі гри.

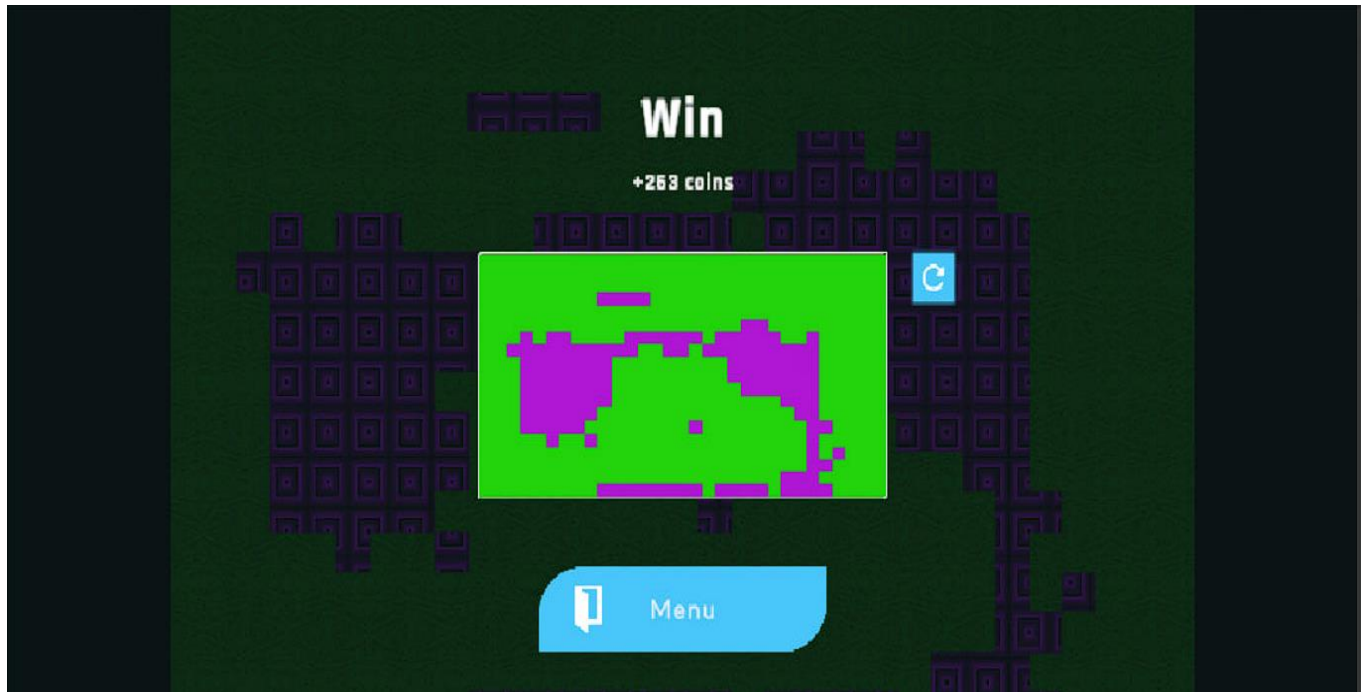


Рис. 3.11 – Екран повтору зміни мапи в грі

3.2. Тестування роботи програми .

Тестування програмного забезпечення є дуже важливою частиною в розробці нашого продукту, адже треба бути орієнтовним на користувача, щоб при його користуванні йому було максимально зручно.

Проведемо тестування та переконаємось в точності безпомилкої роботи даного проекту.

- у цьому релізі всі функції повні та допрацьовані;
- цільовий час проходження гри повністю реалізовано;
- усі письмові матеріали професійно озвучені;
- немає художніх об'єктів-заповнювачів відсутніх елементів;
- на титульному екрані є справжній заголовок та логотип;
- назву остаточно узгоджено;
- усі художні матеріали відповідають головному стилю;

- більшість нових гравців може грати без зовнішнього керівництва;
- багато гравців із цільової аудиторії грають більшу частину основного ігрового контенту за власним бажанням;
- усі звукові ефекти та музика вбудовані та відповідають основній темі;
- усі взаємодії мають звукові ефекти;
- гра працює з цільовою частотою кадрів у більшості тестових сценаріїв. збережено грабельність у всіх протестованих сценаріях;
- усі функції та контент завершені;
- інтерфейс користувача завершено і відповідає головному стилю, за винятком заставки та титрів.

Після завершення тестування можна сказати, що даний продукт реалізований правильно відповідає тим параметрам, які спочатку були для нього задані. Нейромережа працює в хорошому режимі, і не потребує новацій та корекцій. ШІ відіграє значну роль в ігровому світі для чесної гри та покращення користування додатком для гравців.

Висновки до третього розділу

Було успішно запущено всю систему гри для перевірки повної реалізації послідовного переходу користувача від одного пункту до іншого

Написаний проект був протестований на основні норми випробувань, задля кращого розуміння на чим треба працювати і що треба вилучити або додати, за для того аби користувачі мали як можна комфортніші умови для гри.

Всі 2D 3D Анімацію працюють належним чином і відповідають поставленим умовам реалізації проекту.

ВИСНОВКИ

В процесі виконання кваліфікаційної роботи магістра було написано та реалізовано кросплатформену мультиплеєрну 2D гру на Unity 3D з застосуванням URP.

Також було проведено аналіз існуючих технологій програмного продукту і представлено переваги та недоліки. Було обрано інтегроване середовище розробки Visual Studio через його надійність та зручність програмного коду. Рушієм було обрано Unity 3D, тому що це якісний інструмент для розроблення відеоігор і застосунків. Створені за допомогою Unity програми працюють на настільних комп'ютерних системах, мобільних пристроях та гральних консолях у дво- та тривимірній графіці, та на пристроях віртуальної чи доповненої реальності.

В ході аналізу було визначено метод навчання нейромережі та реалізовано можливості ШІ для втілення ідеї продукту, чат GPT було взято через швидкість прийняття рішень та для вирішення проблем інтерфес і ключового геймплею.

Далі була вирішена проблема створення базової ігрової механіки, включаючи рух персонажів, взаємодію об'єктів, систему керування та інші аспекти ігрового процесу. Застосування різних фізичних властивостей для об'єктів сцени. Формалізовано взаємодію та реакцію віртуальних об'єктів, що дає змогу використання користувачам будь якої вікової категорії.

Завдяки кваліфікаційній роботі було описано архітектури проекту, використаних технологій і методів, а також інструкцій для користувачів

Як додатковою привілеگیю було створена система донатів для ЗСУ, за які ти отримуєш скін для того, щоб додатково мотивувати користувачів

Під час написання роботи було детально розглянуто процес створення продукту, та проаналізовано всі його етапи. Системи та підсистеми було об'єднано в один суцільний продукт.

Головне меню, сцени, музика піксельні текстура назви все це було взято на Substance share які запропонувано ШІ. Цей сервіс має широкий спектр послух

для людей які реалізують нові проекти.

У підсумку, частини даного проєкту реалізовані вірно і успішно поєднані в повноцінну гру, а, отже, підтверджує, що Unity 3D є потужним і гнучким інструментом для розробки ігрових і інтерактивних додатків, здатним задовольнити потреби як початківців, так і професійних розробників. Отримані знання та навички дозволяють продовжувати розвиток в області геймдевелопменту та успішно реалізовувати складні проекти в майбутньому.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Akenine-Möller, Tomas, Eric Haines, and Naty Hoffman. Real-Time Rendering. A K Peters/CRC Press, 2018.
2. Goldstone, Will. Unity 3.x Game Development Essentials. Packt Publishing Ltd, 2011.
3. Creighton, Jared Halpern and Edward. Learn 2D Game Development with Unity: A Hands-On Guide to Game Creation. Addison-Wesley Professional, 2014.
4. Sparks, Ashley. Unity 2D Game Development Cookbook. Packt Publishing Ltd, 2014..
5. Stamate, Paris. Mastering Unity 2D Game Development. Packt Publishing Ltd, 2018..
6. Sroelsen, Andrew and Philip Japikse. Pro C# 8.0 and the .NET Core 3.0. Apress, 2019..
7. Albahari, Joseph and Ben Albahari. C# 8.0 in a Nutshell: The Definitive Reference. O'Reilly Media, 2019.
8. Skeet, Jon. C# in Depth. Manning Publications, 2019.
9. Офіційна документація Unity: docs.unity3d.com
10. Gregory Jason. Game Engine Architecture. – New York, CRC Press. – 2009. – No. 5. – s. 15 – 24.
11. Gregory Jason. Game Engine Architecture: Second Edition. – New York, CRC Press. – 2014. – No. 32. – s. 23 – 30.
12. Lengyel Eric. Mathematics for 3D Game Programming and Computer Graphics: Third Edition. – Boston, Course Technology. – 2012. – 215s..
13. Microsoft documentation Справочник по языку C# [Електронний ресурс] / режим доступу: <https://itproger.com/ua/spravka/csharp->
14. Unity Asset Store [Електронний ресурс] / режим доступу: <https://assetstore.unity.com/>

ДОДАТКИ

DoubleHealthAbility.cs:

```

using BrickDuel.Game.Players;
using System.Collections.Generic;
using UnityEngine;
using static BrickDuel.Game.Map.GameGrid;

namespace BrickDuel.Game.Abilities
{
    public class DoubleHealthAbility : AbilityBase
    {
        public const int TARGET_HEALTH_VALUE = 2;

        private int targetPlayerId;

        public override void ActivateLocal(int Rank, GamePlayerBase player)
        {
            base.ActivateLocal(Rank, player);

            targetPlayerId = player.UniqueID;
            var cluster = GetCluster(Rank, player.UniqueID);
            grid.SetClusterHealth(cluster, TARGET_HEALTH_VALUE, true);
        }

        public override void ActivateRemote(int Rank, GamePlayerBase player)
        {
            base.ActivateRemote(Rank, player);

            audio.PlaySfx(activateSoundId);
        }

        public override void Deactivate()
        {
            base.Deactivate();

            var cluster = GetCluster(AbilitiesController.MAX_RANK - 1, targetPlayerId);
            grid.ResetClusterHealth(cluster);

            audio.PlaySfx(deactivateSoundId);
        }

        private ClusterData GetCluster (int rank, int playerUniqueId)
        {
            Debug.Log($"Generate cluster by rank {rank}");
            if (rank == AbilitiesController.MAX_RANK - 1)
                return new ClusterData(playerUniqueId);

            var blocks = new List<Vector2Int>();
            for (var x = 0; x < grid.Grid.GetLength(0); x++)
            {
                for (var y = 0; y < grid.Grid.GetLength(1); y++)
                {
                    var current = grid.Grid[x, y];
                    if (current.Owner != playerUniqueId)
                        continue;

                    switch (rank)
                    {
                        case 0:
                            var up = hasOpponentBlock(Vector2Int.up);
                            var down = hasOpponentBlock(Vector2Int.down);
                            var left = hasOpponentBlock(Vector2Int.left);
                            var right = hasOpponentBlock(Vector2Int.right);

                            if (up || down || right || left)
                                blocks.Add(new Vector2Int(x, y));

                            bool hasOpponentBlock (Vector2Int direction)
                            {

```

```

                var pos = new Vector2Int(x + direction.x, y +
direction.y);
                return grid.IsValidIndex(pos) && grid.Grid[pos.x,
pos.y].Owner != playerUniqueId;
            }
            break;
        case 1:
            blocks.Add(new Vector2Int(x, y));
            break;
        }
    }
}
return new ClusterData(blocks.ToArray());
}
}
}
}

```

BombTouchAbility.cs:

```

using BrickDuel.Game.Players;

namespace BrickDuel.Game.Abilities
{
    public class BombTouchAbility : AbilityBase
    {
        public override void ActivateLocal(int rank, GamePlayerBase player)
        {
            base.ActivateLocal(rank, player);

            UnityEngine.Debug.Log($"Init local ability: {player}");
        }
    }
}

```

AbilitiesPanelUI.cs:

```

using BrickDuel.Data;
using BrickDuel.Game.Players;
using System.Collections.Generic;
using TMPPro;
using UnityEngine;
using UnityEngine.UI;
using Zenject;

namespace BrickDuel.Game.Abilities
{
    public class AbilitiesPanelUI : MonoBehaviour
    {
        [SerializeField] RectTransform parent;
        [SerializeField] AbilityPanelItemUI prefab;
        [SerializeField] Image pointsBar;
        [SerializeField] TextMeshProUGUI pointsText;

        private PlayerStorage playerStorage;
        private AbilitiesController controller;
        private List<AbilityPanelItemUI> slots = new();

        public void LinkWithController (AbilitiesController player, PlayerStorage
storage)
        {
            controller = player;
            playerStorage = storage;

            controller.GetComponent<GamePlayerBase>().RankedPoints.OnValueChanged.AddListener(OnRanked
dPointsUpdate);

            InitSlotsView();
            UpdatePanelView();
            OnRankedPointsUpdate(0, player.CurrentRank.Current);
        }

        private void OnRankedPointsUpdate (int old, int current)
        {
            var percent = (float)current / AbilitiesController.MAX_POINTS_USED;
            pointsBar.fillAmount = percent;
        }
    }
}

```

```

        if (pointsText)
            pointsText.text = current.ToString();
    }

    private void Update()
    {
        if(controller)
        {
            UpdatePanelView();
        }
    }

    private void OnDestroy()
    {
        if(controller)
            controller.GetComponent<GamePlayerBase>().
                RankedPoints.OnValueChanged.RemoveListener(OnRankedPointsUpdate);
    }

    private void InitSlotsView ()
    {
        for (var x = 0; x < controller.slots.Length; x++)
        {
            var slotItemId = x >= playerStorage.LocalSession.UsedAbilities.Length ?
                string.Empty :
                playerStorage.LocalSession.UsedAbilities[x];
            if (string.IsNullOrEmpty(slotItemId))
                continue;

            var instance = Instantiate(prefab, parent);
            instance.controller = controller;
            instance.playerData = playerStorage;
            instance.slotId = x;

            ProjectContext.Instance.Container.Inject(instance);
            slots.Add(instance);
        }
    }

    public void UpdatePanelView ()
    {
        for (var x = 0; x < slots.Count; x++)
            UpdateSlotView(x);
    }

    private void UpdateSlotView (int slotId)
    {
        slots[slotId].UpdateView();
    }
}

```

Booster.cs:

```

using BrickDuel.Game.Map;
using BrickDuel.Game.Players;
using BrickDuel.Lobby.Grid;
using Photon.Pun;
using Sirenix.OdinInspector;
using UnityEngine;

namespace BrickDuel.Game.Boosters
{
    public abstract class Booster : PunZinjectMonoBehaviour
    {
        [ShowInInspector, ReadOnly]
        public Vector2Int Pos { get; private set; }
        public float TimeLife => Time.time - spawnDate;
        public float ActivatedTime => Time.time - activatedDate;
        public bool IsActivated => activatedDate != 0;

        [Range(0, 360)]
        public float MaxTimeLife = 5f;

        protected GameGrid grid { get; private set; }
    }
}

```

```

protected float spawnDate { get; private set; }
protected float activatedDate { get; private set; }

public virtual void Init(GameGrid grid, Vector2Int pos)
{
    Pos = pos;
    this.grid = grid;
    spawnDate = Time.time;

    if (PhotonNetwork.IsMasterClient)
        photonView.RPC(nameof(InitRpc), RpcTarget.Others, pos.x, pos.y);
}

[PunRPC]
public void InitRpc(int x, int y)
{
    var pos = new Vector2Int(x, y);
    Init(FindObjectOfType<GameGrid>(), pos);
}

public virtual void Activate(OnePlayerMotor motor)
{
    activatedDate = Time.time;
    if (photonView.IsMine)
        photonView.RPC(nameof(ActivateRpc), RpcTarget.Others,
motor.photonView.ViewID);
}

protected virtual void ActivateAnswer()
{
}

[PunRPC]
public void ActivateRpc(int playerViewID)
{
    var player = PhotonView.Find(playerViewID).GetComponent<OnePlayerMotor>();
    Activate(player);
    ActivateAnswer();
}

protected virtual void Update()
{
    if (PhotonNetwork.IsMasterClient &&
        MaxTimeLife > 0 &&
        TimeLife > MaxTimeLife)
    {
        PhotonNetwork.Destroy(gameObject);
    }
}
}

```

IOHelper.cs:

```

using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Runtime.Serialization.Formatters.Binary;

namespace BrickDuel.Application.IO
{
    public static class IOHelper
    {
        public const string DEFAULT_FILE_EXTENSION = ".nd";

        public static string BuildStoragePath (params string[] items)
        {
            var resItems = new List<string>() {
                UnityEngine.Application.persistentDataPath };
            resItems.AddRange(items);
            return Path.Combine(resItems.ToArray());
        }

        public static void PrepareFolder (string shortPath)
        {
        }
    }
}

```

```

        {
            var directories = shortPath.Split(@"\");

            for (var i = 0; i < directories.Length; i++)
            {
                var directory = Path.Combine(directories.Take(i + 1).ToArray());
                if (directory.EndsWith(DEFAULT_FILE_EXTENSION))
                    return;

                var path = BuildStoragePath(directory);
                if (Directory.Exists(path) == false)
                    Directory.CreateDirectory(path);
            }
        }

        public static void DeleteFile (string shortPath)
        {
            var fullPath = BuildStoragePath(shortPath);
            if (File.Exists(fullPath))
                File.Delete(fullPath);
        }

        public static void SerializeIntoFile(this object target,
                                            string directories,
                                            string fileName,
                                            string fileExtension =
DEFAULT_FILE_EXTENSION)
        {
            var fullPath = BuildStoragePath(directories, fileName + fileExtension);
            PrepareFolder(directories);
            SerializeIntoFile(target, fullPath);
        }

        public static void SerializeIntoFile (this object target, string shortFilePath)
        {
            var fullPath = BuildStoragePath(shortFilePath);
            PrepareFolder(fullPath);
            using (var fs = File.Open(fullPath, FileMode.OpenOrCreate))
            {
                var formatter = new BinaryFormatter();
                formatter.Serialize(fs, target);
            }
        }

        public static bool ContainsSavedFile (this string shortFilePath)
        {
            var fullPath = BuildStoragePath(shortFilePath);
            return File.Exists(fullPath);
        }

        public static T DeserializeFromFile<T> (this string shortFilePath) where T :
class
        {
            var fullPath = BuildStoragePath(shortFilePath);
            if (File.Exists(fullPath))
            {
                using (var fs = File.Open(fullPath, FileMode.Open))
                {
                    var formatter = new BinaryFormatter();
                    return formatter.Deserialize(fs) as T;
                }
            }

            return null;
        }
    }

```

GlobalMap.cs:

```

using BrickDuel.Game.Players;
using Photon.Pun;
using System;
using System.Collections.Generic;
using UnityEngine;

```

```

using static BrickDuel.Game.Map.GameGrid;

namespace BrickDuel.Game.Boosters
{
    public class CrackBooster : BoosterDetail
    {
        [SerializeField] Vector2Int directionsRange = new Vector2Int(1, 3);
        [SerializeField] bool useDiagonal = true;
        [SerializeField] int lineLength = 8;
        [SerializeField, Range(0, 1)] float lineCurve = 0.1f;

        public override void Activate(OnePlayerMotor player)
        {
            base.Activate(player);

            if (PhotonNetwork.IsMasterClient &&
                photonView.IsMine)
            {
                var data = GetCluster();
                grid.CaptureByCluster(data, player.UniqueID, true);
            }

            private ClusterData GetCluster()
            {
                var result = new List<Vector2Int>() { Pos };
                var directions = UnityEngine.Random.Range(directionsRange.x,
directionsRange.y);

                for (var x = 0; x < directions; x++)
                {
                    var directionVector = grid.GetRandomDirection(useDiagonal);

                    for (var i = 1; i <= lineLength; i++)
                    {
                        var crackPosition = Pos + i * directionVector;
                        if (grid.IsValidIndex(crackPosition) == false)
                            continue;

                        result.Add(crackPosition);

                        if (lineCurve > UnityEngine.Random.value)
                            directionVector = grid.GetRandomDirection(useDiagonal);
                    }
                }

                return new ClusterData(result.ToArray());
            }
        }
    }
}

```

NetworkPlayerExtensions.cs:

```

using ExitGames.Client.Photon;
using Photon.Pun;
using Photon.Realtime;
using UnityEngine;

namespace BrickDuel.Game.Extensions
{
    public static class NetworkPlayerExtensions
    {
        public static void SetPlayerParameters(Hashtable entries)
        {
            PhotonNetwork.LocalPlayer.SetCustomProperties(entries);
        }

        public static void SetPlayerReady(this Player player, bool isReady)
        {
            player.SetCustomProperties(new Hashtable() { { "IsReady", isReady } });
        }

        public static bool GetPlayerReady(this Player player)
        {

```

```

    {
        return (bool)player.CustomProperties["IsReady"];
    }

    public static void SetPlayerColor(this Player player, Color color)
    {
        player.SetCustomProperties(new Hashtable() {
            { "ColorR", color.r },
            { "ColorG", color.g },
            { "ColorB", color.b }
        });
    }

    public static Color GetPlayerColor(this Player player)
    {
        var r = (float)player.CustomProperties["ColorR"];
        var g = (float)player.CustomProperties["ColorG"];
        var b = (float)player.CustomProperties["ColorB"];
        return new Color(r, g, b);
    }

    public static void SetPlayerSkin (this Player player, string skinId)
    {
        player.SetCustomProperties(new Hashtable()
        {
            { "EquippedSkin", skinId }
        });
    }

    public static string GetPlayerSkin (this Player player)
    {
        return (string)player.CustomProperties["EquippedSkin"];
    }
}

```

BattleGamemode.cs:

```

using BrickDuel.Game.Players;
using BrickDuel.Game.UI;
using Photon.Pun;
using Sirenix.OdinInspector;
using System.Collections.Generic;
using System.Linq;
using UnityEngine;

namespace BrickDuel.Game.Gamemodes
{
    public class BattleGamemode : Gamemode
    {
        [Range(0.1f, 1f)]
        public float WinThreshold = 0.85f;
        public float NeedHoldTime = 15f;

        public bool SameSkinsWarning { get; private set; }

        [ShowInInspector]
        internal WinInfo currentInfo;
        internal GamePlayerBase winner;

        protected override async void Start()
        {
            base.Start();

            gameScreen = await ui.Screens.CreateElement<GameplayScreen>();
        }

        protected override void Update()
        {
            base.Update();

            if(gameScreen)
                gameScreen.SkinSelectionMenuVisibled = IsGameBegan == false &&
                SameSkinsWarning;
        }
    }
}

```

```

protected override void GameUpdate()
{
    base.GameUpdate();

    if (PhotonNetwork.IsMasterClient == false)
        return;

    if (currentInfo == null)
    {
        foreach (var player in Players)
        {
            if (player != null &&
                grid.PercentBlocks(player.UniqueID) >= WinThershold)
            {
                currentInfo = new WinInfo(player.UniqueID);
                photonView.RPC(nameof(UpdateNetWinnerInfoRpc), RpcTarget.Others,
player.UniqueID);
                break;
            }
        }
    }
    else if (winner == null)
    {
        if (grid.PercentBlocks(currentInfo.TargetPlayerID) < WinThershold)
        {
            currentInfo = null;
            photonView.RPC(nameof(UpdateNetWinnerInfoRpc), RpcTarget.Others, -1);
        }
        else
        {
            if (currentInfo.TimeHold >= NeedHoldTime)
            {
                winner = Players[currentInfo.TargetPlayerID];
            }
        }
    }
}

protected override void OnGameNotBeStarted()
{
    base.OnGameNotBeStarted();

    SameSkinsWarning = true;
}

protected override bool CheckGameBeginCondition()
{
    if (HasSameSkins())
    {
        SameSkinsWarning = true;
        return false;
    }

    SameSkinsWarning = false;
    return base.CheckGameBeginCondition();
}

private bool HasSameSkins ()
{
    var usedSkins = new List<string>();
    if(Players != null && Players.Length > 0)
    {
        foreach (var player in Players)
        {
            if (usedSkins.Contains(player.GetOwnerEquipSkin))
                return true;

            usedSkins.Add(player.GetOwnerEquipSkin);
        }
    }
}

```

```

        return false;
    }

    [PunRPC]
    public void UpdateNetWinnerInfoRpc(int playerUniqueID)
    {
        UpdateWinnerInfo(playerUniqueID);
    }

    private void UpdateWinnerInfo(int playerUniqueID)
    {
        if (playerUniqueID == -1)
        {
            currentInfo = null;
        }
        else
        {
            currentInfo = new WinInfo(playerUniqueID);
        }
    }

    public override bool GameEndCondition(out GamePlayerBase winner)
    {
        winner = this.winner;
        return winner != null;
    }

    [System.Serializable]
    public class WinInfo
    {
        public int TargetPlayerID;
        public float TimeBegin;

        public WinInfo(int targetPlayerID)
        {
            TargetPlayerID = targetPlayerID;
            TimeBegin = Time.time;
        }

        [ShowInInspector]
        public float TimeHold =>
            Time.time - TimeBegin;
    }
}

```

GameGrid.cs:

```

using BrickDuel.Data;
using BrickDuel.Game.Abilities;
using BrickDuel.Game.Extensions;
using BrickDuel.Game.Gamemodes;
using BrickDuel.Game.Players;
using Photon.Pun;
using Sirenix.OdinInspector;
using System;
using System.Collections.Generic;
using System.Linq;
using UnityEngine;
using UnityEngine.Events;
using Zenject;

namespace BrickDuel.Game.Map
{
    public class GameGrid : PunZenjectMonoBehaviour
    {
        [Range(0.15f, 3f)]
        public float BlockSize = 0.5f;
        [Range(0f, 1f)]

```

```

public float Contrast = 0.5f;
public GameObject BlockPrefab;

public Vector2Int FieldSize = new Vector2Int(35, 35);

[TableMatrix]
public GridCell[,] Grid;
public Transform[] PlayerZones;

public bool IsInitiated { get; private set; }
public GamePlayerBase LeftPlayer => gamemode.LeftPlayer;
public GamePlayerBase RightPlayer => gamemode.RightPlayer;

public UnityAction<Vector2Int> OnGridUpdated;

[SerializeField]
private Gamemode gamemode;
private MapAdditionalSpawner spawner;
[SerializeField]
private int spawnItemID = 0;
[SerializeField]
private float erosionRate = 3f;
[SerializeField]
private float chanceSpawnOnChangeOwner = 0.15f;
//[SerializeField]
//private GameAudioManager audio;
[Inject] SkinsStorage skins;

protected override void Start()
{
    base.Start();

    spawner = GetComponent<MapAdditionalSpawner>();
}

public void InitField()
{
    if (IsInitiated)
        return;

    Grid = new GridCell[FieldSize.x, FieldSize.y];

    var halfWidth = FieldSize.x * BlockSize * 0.5f;
    var halfHeight = FieldSize.y * BlockSize * 0.5f;
    for (var x = 0; x < FieldSize.x; x++)
    {
        for (var y = 0; y < FieldSize.y; y++)
        {
            Grid[x, y] = new GridCell()
            {
                X = x,
                Y = y,
                Block = Instantiate(BlockPrefab, new Vector3(x * BlockSize - halfWidth + transform.position.x, y
* BlockSize - halfHeight + transform.position.y, 0), Quaternion.identity)
            };

            Grid[x, y].Block.transform.localScale = BlockPrefab.transform.localScale * BlockSize;
            UpdateOwner(new Vector2Int(x, y), x <= FieldSize.x * 0.5f ? 0 : 1, isInit: true);
        }
    }
}

```

```

        IsInit = true;
    }

    public void ContactWithPlayer(GamePlayerBase player,
                                Transform block,
                                bool withRpc = true)
    {
        //TODO: without finding block by cycle
        var pos = GetBlockIndex(block);
        UpdateOwner(pos, player.UniqueID);
        if (withRpc)
            photonView.RPC(nameof(ContactWithPlayerRpc), RpcTarget.Others, player.photonView.ViewID,
pos.x, pos.y);
    }

    public Vector2Int GetBlockIndex (Transform block)
    {
        for (var x = 0; x < FieldSize.x; x++)
        {
            for (var y = 0; y < FieldSize.y; y++)
            {
                if (Grid[x, y].Block == block.gameObject)
                {
                    return new Vector2Int(x, y);
                }
            }
        }
        return -Vector2Int.one;
    }

    [PunRPC]
    public void ContactWithPlayerRpc(int playerViewID,
                                    int blockX,
                                    int blockY)
    {
        var block = Grid[blockX, blockY].Block.transform;
        var player = PhotonView.Find(playerViewID).GetComponent<GamePlayerBase>();
        ContactWithPlayer(player, block, false);
    }

    public void UpdateOwner(Vector2Int index,
                            int ownerID,
                            int health = 1,
                            bool isInit = false,
                            bool withRpc = false)
    {
        if (Grid[index.x, index.y].Owner != ownerID)
        {
            if (isInit == false)
            {
                if (Grid[index.x, index.y].Health <= 1)
                {
                    Grid[index.x, index.y].Owner = ownerID;
                    var newHealth =
gamemode.GetPlayerByUniqueID(ownerID).Abilities.TryGetAbility<DoubleHealthAbility>(true, out var
dhAbility) ?
                        DoubleHealthAbility.TARGET_HEALTH_VALUE :
                        health;
                    //Debug.Log($"{ownerID} new health: {newHealth}");
                    Grid[index.x, index.y].SetControll(PlayerZones[ownerID], newHealth);
                    Grid[index.x, index.y].ApplyBlockSkin(LoadBlockSkin(ownerID));
                }
            }
        }
    }

```

```

        if (gamemode.GetPlayerByUniqueID(ownerID).RankedPoints.Current <
AbilitiesController.MAX_POINTS_USED)
            gamemode.GetPlayerByUniqueID(ownerID).RankedPoints.Current++;

        Grid[index.x, index.y].PlayInteractionAnim();

        if (PhotonNetwork.IsMasterClient && UnityEngine.Random.value <
chanceSpawnOnChangeOwner)
            spawner.Spawn(spawnItemID, index);
        }
        else if (Grid[index.x, index.y].Health > 1)
        {
            Grid[index.x, index.y].Health--;
            Grid[index.x, index.y].ApplyBlockSkin(LoadBlockSkin(Grid[index.x, index.y].Owner));
        }

        OnGridUpdated?.Invoke(index);
    }
    else
    {
        Grid[index.x, index.y].Owner = ownerID;
        Grid[index.x, index.y].ApplyBlockSkin(LoadBlockSkin(ownerID));
        Grid[index.x, index.y].SetControll(PlayerZones[ownerID]);
    }

    if (withRpc)
        photonView.RPC(nameof(UpdateOwnerRpc), RpcTarget.Others, index.x, index.y, ownerID, isInit,
health);
    }
}

private SkinModel.DetailInfo LoadBlockSkin(int playerUniqueId)
{
    var player = gamemode.GetPlayerByUniqueID(InvertID(playerUniqueId));
    var skinId = PhotonNetwork.CurrentRoom.Players[player.photonView.OwnerActorNr].GetPlayerSkin();
    var skin = skins[skinId];
    return skin.BlockSkin;
}

[PunRPC]
public void UpdateOwnerRpc(int px,
    int py,
    int owner,
    bool isInit,
    int health)
{
    UpdateOwner(new Vector2Int(px, py),
        owner,
        health: health,
        isInit: isInit,
        withRpc: false);
}

public static int InvertID(int input)
{
    return input == 1 ? 0 : 1;
}

#region Grid Properties and Methods
//Can be optimized into cycles (dont need load skin everytime)

```

```

public void SetClusterHealth(ClusterData cluster, int health, bool withSyncRpc = false)
{
    if (cluster.CaptureByOwnerId)
    {
        for (var x = 0; x < Grid.GetLength(0); x++)
        {
            for (var y = 0; y < Grid.GetLength(1); y++)
            {
                if (Grid[x, y].Owner == cluster.TargetOwnerId)
                {
                    Grid[x, y].Health = health;
                    Grid[x, y].ApplyBlockSkin(LoadBlockSkin(cluster.TargetOwnerId));
                    OnGridUpdated?.Invoke(new Vector2Int(x, y));
                }
            }
        }
    }
    else
    {
        for (var x = 0; x < cluster.Points.Length; x++)
        {
            Grid[cluster.Points[x].x, cluster.Points[x].y].Health = health;
            Grid[cluster.Points[x].x,
cluster.Points[x].y].ApplyBlockSkin(LoadBlockSkin(cluster.TargetOwnerId));
            OnGridUpdated?.Invoke(cluster.Points[x]);
        }
    }

    if (withSyncRpc)
        photonView.RPC(nameof(SyncClusterHealthRpc), RpcTarget.Others, JsonUtility.ToJson(cluster),
health);
}

public void ResetClusterHealth (ClusterData data, bool withSyncRpc = false)
{
    SetClusterHealth(data, 1, withSyncRpc);
}

[PunRPC]
public void SyncClusterHealthRpc (string clusterJson, int health)
{
    var cluster = JsonUtility.FromJson<ClusterData>(clusterJson);
    SetClusterHealth(cluster, health);
}

public Vector2Int GetDirectionVector(Direction direction)
{
    switch (direction)
    {
        case Direction.North:
            return new Vector2Int(0, -1);
        case Direction.South:
            return new Vector2Int(0, 1);
        case Direction.East:
            return new Vector2Int(1, 0);
        case Direction.West:
            return new Vector2Int(-1, 0);
        default:
            return Vector2Int.zero;
    }
}

```

```

public Vector2Int GetRandomDirection(bool withDiagonal = false)
{
    var values = (Direction[])Enum.GetValues(typeof(Direction));
    var tmp = new List<Vector2Int>();
    foreach (var item in values)
        tmp.Add(GetDirectionVector(item));

    if (withDiagonal)
    {
        tmp.Add(new Vector2Int(-1, -1));
        tmp.Add(new Vector2Int(-1, 1));
        tmp.Add(new Vector2Int(1, -1));
        tmp.Add(new Vector2Int(1, 1));
    }

    return tmp[UnityEngine.Random.Range(0, tmp.Count)];
}

public bool IsValidIndex (Vector2Int index)
{
    return (index.x >= 0 && index.x < Grid.GetLength(0) &&
        index.y >= 0 && index.y < Grid.GetLength(1));
}

public void CaptureByCluster (ClusterData cluster, int newOwnerId, bool withSyncRpc = false)
{
    if (cluster.CaptureByOwnerId)
    {
        for (var x = 0; x < Grid.GetLength(0); x++)
        {
            for (var y = 0; y < Grid.GetLength(1); y++)
            {
                if (Grid[x, y].Owner == cluster.TargetOwnerId)
                {
                    UpdateOwner(cluster.Points[x], newOwnerId);
                }
            }
        }
    }
    else
    {
        for (var x = 0; x < cluster.Points.Length; x++)
        {
            UpdateOwner(cluster.Points[x], newOwnerId);
        }
    }

    if (withSyncRpc)
    {
        var json = JsonUtility.ToJson(cluster);
        //Debug.Log($"Send cluster (new owner - {newOwnerId}) json: {json}");
        photonView.RPC(nameof(CaptureByClusterRpc), RpcTarget.Others, json, newOwnerId);
    }
}

[PunRPC]
public void CaptureByClusterRpc (string clusterJson, int newOwnerId)
{
    var cluster = JsonUtility.FromJson<ClusterData>(clusterJson);
    CaptureByCluster(cluster, newOwnerId);
}

```

```

    }

    public void CaptureCircle(Vector2Int center, float radius, int targetPlayerID, bool withSyncRpc = false)
    {
        var blocks = new List<Vector2Int>();
        for (var x = 0; x < Grid.GetLength(0); x++)
        {
            for (var y = 0; y < Grid.GetLength(1); y++)
            {
                if (Vector2Int.Distance(Grid[x, y].Pos, center) <= radius)
                {
                    blocks.Add(new Vector2Int(x, y));
                }
            }
        }

        for (var x = 0; x < blocks.Count; x++)
        {
            UpdateOwner(blocks[x], targetPlayerID);
        }

        if (withSyncRpc)
            photonView.RPC(nameof(CaptureCircleRpc), RpcTarget.Others, center.x, center.y, radius,
targetPlayerID);
    }

    [PunRPC]
    public void CaptureCircleRpc(int x, int y, float radius, int playerId)
    {
        var pos = new Vector2Int(x, y);
        CaptureCircle(pos, radius, playerId);
    }

    public int BlocksCount(int ownerID)
    {
        var result = 0;
        if (Grid == null || Grid.Length == 0)
            return result;

        return PlayerZones[ownerID].childCount;
    }

    public int AllBlocksCount()
    {
        return FieldSize.x * FieldSize.y;
    }

    public float PercentBlocks(int ownerID)
    {
        return (float)BlocksCount(ownerID) / AllBlocksCount();
    }

    public List<ClusterData> FindClusters(int value)
    {
        var clusters = new List<ClusterData>();
        var visited = new HashSet<(int, int)>();

        for (int x = 0; x < Grid.GetLength(0); x++)
        {
            for (int y = 0; y < Grid.GetLength(1); y++)
            {

```

```

        if (Grid[x, y].Owner == value && !visited.Contains((x, y)))
        {
            var cluster = FindCluster(x, y, value, visited);
            clusters.Add(new ClusterData(cluster.ToArray()));
        }
    }
}

return clusters;
}

private List<Vector2Int> FindCluster(int x, int y, int value, HashSet<(int, int)> visited)
{
    var cluster = new List<Vector2Int>();
    visited.Add((x, y));
    cluster.Add(new Vector2Int(x, y));

    for (int dx = -1; dx <= 1; dx++)
    {
        for (int dy = -1; dy <= 1; dy++)
        {
            int nx = x + dx;
            int ny = y + dy;

            if (nx >= 0 &&
                nx < Grid.GetLength(0) &&
                ny >= 0 &&
                ny < Grid.GetLength(1))
            {
                if (Grid[nx, ny].Owner == value && !visited.Contains((nx, ny)))
                {
                    cluster.AddRange(FindCluster(nx, ny, value, visited));
                }
            }
        }
    }

    return cluster;
}

public void MakeForAllBlocks(ClusterData cluster, Func<Vector2Int, GameObject> handleMethod)
{
    for (var x = 0; x < cluster.Points.Length; x++)
    {
        handleMethod?.Invoke(cluster.Points[x]);
    }
}
#endregion

#region Errosion
private float corrosionRate = 3f;

public void HandleErrosion()
{
    if (PhotonNetwork.IsMasterClient == false)
        return;

    if (corrosionRate > 0)
    {
        corrosionRate -= Time.deltaTime;
    }
}

```

```

else
{
    corrosionRate = errosionRate;
    var tmp = new Dictionary<int, List<ClusterData>>>();
    for (var x = 0; x < gamemode.Players.Length; x++)
    {
        var clusters = FindClusters(gamemode.Players[x].UniqueID);
        for (var y = 0; y < clusters.Count; y++)
        {
            if (clusters[y].ConainsPlayer(this, gamemode.Players[x].transform, BlockSize) == false)
            {
                if (tmp.ContainsKey(x))
                    tmp[x].Add(clusters[y]);
                else
                    tmp[x] = new List<ClusterData>() { clusters[y] };
            }
        }
    }

    if (tmp.Count > 0)
    {
        var apply = tmp.ElementAt(UnityEngine.Random.Range(0, tmp.Count));
        var resultCluster = apply.Value[UnityEngine.Random.Range(0, apply.Value.Count)];
        resultCluster.CalculatePointsByZone(this);
        var resultBlock = resultCluster.GetRandomBoundaryPoint();
        UpdateOwner(resultBlock, InvertID(apply.Key), withRpc: true);
    }
}
}
#endregion

```

```

[Serializable]
public class GridCell
{
    [NonSerialized]
    public GameObject Block;
    public int X;
    public int Y;
    public int Owner = -1;
    public int Health = 1;

    public Vector2Int Pos =>
        new Vector2Int(X, Y);

    public void PlayInteractionAnim()
    {
        Block.GetComponent<Animation>().Play();
    }

    public void SetBlockColor(Color color)
    {
        Block.GetComponentInChildren<SpriteRenderer>().color = color;
    }

    public void ApplyBlockSkin (SkinModel.DetailInfo blockStyle)
    {
        var spriteRend = Block.GetComponentInChildren<SpriteRenderer>();
        blockStyle.ApplyTo(spriteRend, Health);
    }

    public void SetControll(Transform parentOfGroup, int health = 1)

```

```

    {
        Block.transform.parent = parentOfGroup;
        Block.layer = parentOfGroup.gameObject.layer;
        Health = health;
    }
}

```

[Serializable]

```

public class ClusterData
{
    public int TargetOwnerId;
    public Vector2Int[] Points;

    public ClusterData (int targetOwnerId)
    {
        TargetOwnerId = targetOwnerId;
    }

    public ClusterData(Vector2Int[] points)
    {
        Points = points;
        TargetOwnerId = int.MinValue;
    }

    public bool CaptureByOwnerId => TargetOwnerId != int.MinValue;

    public bool ContainsPlayer(GameGrid grid, Transform targetToCheck, float blockSize)
    {
        for (int x = 0; x < Points.Length; x++)
        {
            if (Vector2.Distance(targetToCheck.position, grid.Grid[Points[x].x,
Points[x].y].Block.transform.position) < blockSize)
            {
                return true;
            }
        }

        return false;
    }

    public Vector2Int GetRandomBoundaryPoint()
    {
        return Points[UnityEngine.Random.Range(0, Points.Length)];
    }

    public void CalculatePointsByZone (GameGrid grid)
    {
        var tmp = new List<Vector2Int>();

        for(var i = 0; i < Points.Length; i++)
        {
            var x = Points[i].x;
            var y = Points[i].y;

            var up = hasOpponentBlock(Vector2Int.up);
            var down = hasOpponentBlock(Vector2Int.down);
            var left = hasOpponentBlock(Vector2Int.left);
            var right = hasOpponentBlock(Vector2Int.right);

            if (up || down || right || left)
                tmp.Add(new Vector2Int(x, y));
        }
    }
}

```

```
        bool hasOpponentBlock(Vector2Int direction)
        {
            var pos = new Vector2Int(x + direction.x, y + direction.y);
            return grid.IsValidIndex(pos) && grid.Grid[pos.x, pos.y].Owner != grid.Grid[x, y].Owner;
        }
    }

    Points = tmp.ToArray();
}

public enum Direction
{
    North,
    South,
    East,
    West
}
}
```

GamePlayerBase.cs:

```
using BrickDuel.Data;
using BrickDuel.Game.Abilities;
using BrickDuel.Game.Extensions;
using BrickDuel.Game.Gamemodes;
using BrickDuel.Game.Map;
using BrickDuel.Lobby.Grid;
using Photon.Pun;
using Sirenix.OdinInspector;
using System.Collections.Generic;
using UnityEngine;
using Zenject;

namespace BrickDuel.Game.Players
{
    [RequireComponent(typeof(AbilitiesController))]
    public abstract class GamePlayerBase : PunZenjectMonoBehaviour
    {
        [ShowInInspector]
        public int UniqueID =>
            photonView.OwnerActorNr - 1;
        public int MaxMotors = 5;
        public Color TeamColor
        {
            get
            {
                var player = PhotonNetwork.CurrentRoom.GetPlayer(UniqueID + 1);
                if (player.CustomProperties.ContainsKey("ColorR"))
                {
                    return player.GetPlayerColor();
                }
                return Color.cyan;
            }
        }

        public Value<int> Coins = new Value<int>();
        public Value<int> RankedPoints = new Value<int>();
        public AbilitiesController Abilities { get; private set; }
        public LayerMask MapMask;

        public Vector3 StartPos;

        #region Audio
        public string DefaultTouchSound = "bg_block_break";
        public string OverrideTouchSound = string.Empty;

        public string GetTouchSound => string.IsNullOrEmpty(OverrideTouchSound) ?
DefaultTouchSound : OverrideTouchSound;
        #endregion

        public abstract void GameUpdate();

        protected Gamemode gamemode;
        protected GameGrid grid;
        protected PlayerStorage playerData =>
            ProjectContext.Instance.Container.Resolve<PlayerStorage>();

        protected List<OnePlayerMotor> Motors =
            new List<OnePlayerMotor>();

        protected override void Start()
        {
            base.Start();

            StartPos = transform.position;
            grid = FindObjectOfType<GameGrid>();
            Abilities = GetComponent<AbilitiesController>();
            gamemode = FindObjectOfType<Gamemode>();
            gamemode.RegisterPlayer(this);

            PlayerStorage.OnDataChanged += ChangedData;
        }
    }
}
```

```

        MapMask = gamemode.MasksOnSpawnPoints[UniqueID];
        SpawnMotor(StartPos);
    }

protected override void OnDestroy()
{
    PlayerStorage.OnDataChanged -= ChangedData;

    base.OnDestroy();
}

private void ChangedData ()
{
    ChangeSkin(playerData.LocalSession.Equipped);
}

public int MotorsCount =>
    Motors.Count;

public void SpawnMotors (int count, Vector3 position, float timelife = 0)
{
    for (var x = 0; x < count; x++)
        SpawnMotor(position, timelife);
}

public void SpawnMotor (Vector3 position, float timelife = 0)
{
    if (photonView.IsMine &&
        Motors.Count < MaxMotors)
    {
        var instance = PhotonNetwork.
            Instantiate("PlayerMotor", position, Quaternion.identity).
            GetComponent<OnePlayerMotor>();
        Motors.Add(instance);
        instance.Owner = this;
        photonView.RPC(nameof(SyncMotor),
            RpcTarget.Others,
            instance.photonView.ViewID,
            true);

        if (timelife > 0)
            instance.SetTimelifeDuration(timelife);

        UpdateMotorsView();
    }
}

[PunRPC]
public void SyncMotor (int objectViewID, bool addOrRemove)
{
    var instance = PhotonView.Find(objectViewID).GetComponent<OnePlayerMotor>();
    if (addOrRemove)
    {
        Motors.Add(instance);
        instance.Owner = this;
    }
    else
        Motors.Remove(instance);

    UpdateMotorsView();
}

public OnePlayerMotor GetMotor (int index)
{
    if (index >= 0 && index < Motors.Count)
        return Motors[index];

    return null;
}

```

```

public void KillMotors (int count)
{
    if(photonView.IsMine)
    {
        for (var x = Motors.Count - 1;
            x >= 0 && count > 0;
            x--, count--)
        {
            photonView.RPC(nameof(SyncMotor), RpcTarget.Others,
Motors[x].photonView.ViewID, false);
            PhotonNetwork.Destroy(Motors[x].gameObject);
            Motors.RemoveAt(x);
        }
    }
}

public void KillMotor (OnePlayerMotor instance)
{
    if(photonView.IsMine)
    {
        if(Motors.Contains(instance))
        {
            Motors.Remove(instance);
            photonView.RPC(nameof(SyncMotor), RpcTarget.Others,
instance.photonView.ViewID, false);
            PhotonNetwork.Destroy(instance.gameObject);
        }
    }
}

public void Throw (Vector2 dir, float force)
{
    for (var x = 0; x < Motors.Count; x++)
        if (Motors[x] != null)
            Motors[x].Throw(dir, force);
}

public void ThrowInto (Vector2 pos, float force)
{
    for (var x = 0; x < Motors.Count; x++)
        if (Motors[x] != null)
            Motors[x].Throw(pos - (Vector2)Motors[x].transform.position, force);
}

public void UpdateMotorsView()
{
    for (var x = 0; x < Motors.Count; x++)
        Motors[x]?.UpdateView();
}

#region Network Utils
public string GetOwnerEquipSkin =>
    PhotonNetwork.CurrentRoom.Players[photonView.OwnerActorNr].GetPlayerSkin();

public void ChangeSkin (string skinId)
{
    if(photonView.IsMine)
    {
        PhotonNetwork.LocalPlayer.SetPlayerSkin(skinId);
    }
}
#endregion

[System.Serializable]
public class AbilityUsage
{
    public string[] Slots;

    public string this[AbilityRank index]
    {
        get

```

```

        {
            return Slots[(int)index];
        }
        set
        {
            var targetIdx = (int)index;
            if (Slots.Length < targetIdx)
                Slots = new string[3];
            Slots[targetIdx] = value;
        }
    }

[System.Serializable]
public class AbilitySlot
{
}
}
}

```