Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет центр післядипломної освіти
(повна назва)

Кафедра програмної інженерії
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти перший (бакалаврський)

Ігровий програмний застосунок в жанрі аркадного файтінгу на рушії Unity
(тема)

Виконав:
студент 4 курсу, групи ПЗПІп-22-1

Блох Д.С.
(прізвище, ініціали)

Спеціальність 121 – Інженерія програмного забезпечення
(код і повна назва спеціальності)
Тип програми освітньо-професійна
Освітня програма Програмна інженерія
(повна назва освітньої програми)

Керівник доц. кафедри ПІ Кириченко І.В.
(посада, прізвище, ініціали)

Допускається до захисту
Зав. кафедри _____ З.В.Дудар
(підпис) (прізвище, ініціали)

2024 р.

Харківський національний університет радіоелектроніки

Факультет центр післядипломної освіти
Кафедра програмної інженерії
Рівень вищої освіти перший (бакалаврський)
Спеціальність 121 – Інженерія програмного забезпечення
Тип програми Освітньо-професійна
Освітня програма Програмна Інженерія
(шифр і назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«___» _____ 2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові Блоху Денису Сергійовичу
(прізвище, ім'я, по батькові)

1. Тема роботи Ігровий програмний застосунок в жанрі аркадного файтінгу на рушії Unity

Затверджена наказом по університету від 17.06. 2024р. № 588 Ст

2. Термін подання студентом роботи до екзаменаційної комісії 28.06.2024

3. Вихідні дані до роботи Розробити ігровий застосунок в жанрі аркадного файтінгу, який реалізує арену для битви двох гравців, яких можна обрати через меню гри, мовою програмування C#, використовуючи ігровий рушій Unity2D.

4. Перелік питань, що потрібно опрацювати в роботі

Вступ, аналіз предметної галузі, формування вимог до програмної системи, архітектура та проектування програмного забезпечення, опис прийнятих програмних рішень, тестування розробленого програмного забезпечення, висновки, додатки.

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Аналіз предметної галузі	05.05.2024	*виконано*
2	Створення специфікації ПЗ	10.05.2024	*виконано*
3	Проектування ПЗ	14.05.2024	*виконано*
4	Розробка ПЗ	20.05.2024	*виконано*
5	Тестування ПЗ	08.06.2024	*виконано*
6	Оформлення пояснювальної записки	12.06.2024	*виконано*
7	Підготовка презентації та доповіді	14.06.2024	*виконано*
8	Попередній захист	16.06.2024	*виконано*
9	Нормоконтроль, рецензування	18.06.2024	*виконано*
10	Здача роботи у електронний архів	20.06.2024	*виконано*
11	Допуск до захисту у зав. кафедри	22.06.2024	*виконано*

Дата видачі завдання 16 квітня 2024р.

Студент (ка)  Блох Д.С.
(підпис)

Керівник роботи _____ доц. кафедри ПІ Кириченко І.В.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ / ABSTRACT

Пояснювальна записка до кваліфікаційної роботи бакалавра містить 96 стор., 46 рис., 9 табл., 10 джерел.

ІГРОВИЙ ПРОГРАМНИЙ ЗАСТОСУНОК, ЖАНР АРКАДНОГО ФАЙТІНГУ, UNITY 2D, C#

Об’єкт розробки – ігровий програмний застосунок в жанрі аркадного файтінгу.

Мета розробки – створення ігрового програмного застосунку, який реалізовує процес бою між двома гравцями.

Метод рішення – середовище розробки Unity2D, мова програмування C#.

У результаті розробки створено ігровий програмний застосунок, який реалізує бій між двома гравцями. Гравці можуть обирати одного з чотирьох унікальних персонажів, кожен з яких має власні анімації та стилізований зовнішній вигляд. Ігрокам доступен вибір ігрової арени, на якій буде проводитися гра.

GAME SOFTWARE, ARCADE FIGHTING GENRE, UNITY 2D, C#

The object of development is a game software application in in the arcade fighting genre.

The purpose of the work is to create a game application that implements the process of combat between two players.

Solution method – Unity2D development environment, C# programming language.

As a result of the development, a game application that implements combat between two players. Players can choose one of four unique characters, each with their own animations and stylized appearance. Players have the option to choose the game arena where the game will take place.

Я, Блох Денис Сергійович, студент гр. ПЗПІп-22-1, здобувач вищої освіти на першому (бакалаврському) рівні кафедри «Програмна інженерія», заявляю: моя кваліфікаційна робота на тему «Ігровий програмний застосунок в жанрі аркадного файтінгу на рушії Unity», що буде представлена до екзаменаційної комісії для публічного захисту, виконана самостійно, в ній не містяться елементи плагіату і вона може бути опублікована в електронному архіві відкритого доступу ElAr KhNURE. Усі запозичення з друкованих та електронних джерел мають відповідні посилання.

Я ознайомлений з діючим положенням «Про протидію академічному плагіату в ХНУРЕ», згідно з яким виявлення плагіату є підставою для відмови до допуску кваліфікаційної роботи до захисту та застосування дисциплінарних заходів.

ЗМІСТ

ВСТУП

Роль традиційної гри в житті дитини надзвичайно важлива, адже саме через гру відбувається процес соціалізації, що допомагає дитині адаптуватися до життя в суспільстві. Гра забезпечує розвиток, сприяє взаємодії та допомагає вирішувати проблеми, з якими часто стикається дитина. Людина завжди грає, змінюючи лише види та форми ігор в різні вікові періоди [1].

Темою кваліфікаційної роботи є ігровий програмний застосунок в жанрі аркадного файтінгу на рушії Unity. Основне завдання ігрового програмного застосунку в жанрі аркадного файтінгу – надання гравцям можливості понуритись у світ ігрових аркадних автоматів та отримати яскраві враження від процесу бою між двома гравцями.

Метою роботи є розробка програмного продукту, який реалізовує 2D гру в жанрі аркадного файтінгу.

Жанр аркадних ігор був надзвичайно популярний. Коли підлітки збиралися в ігрових клубах, щоб пограти в аркадні ігрові автомати, створювалася надзвичайна атмосфера та настрій. Важливо підкреслити, що грають на аркадному ігровому автоматі можна було відразу удвох, не онлайн, не по черзі, а використовуючи один ігровий автомат, грати друг проти друга. Саме ця ідея реалізована в моєму ігровому застосунку, коли два ігрока пліч опліч керують грою з однієї клавіатури. Такі культові аркадні ігри як PackMan, Mortal Kombat, Super Mario залишили свій слід в індустрії комп’ютерного геймінгу.

У межах даного проекту будуть використовуватися Adobe Premiere Pro, Photoshop, Unity Asset Store. Adobe Premiere Pro буде використовуватися для редагування відео та обробки анімаційних ефектів. Unity Asset Store стане джерелом готових ассетів, таких як 2D моделі, текстури та інші ресурси для використання в ігровому проекті.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ

1.1 Аналіз галузі ігрової індустрії

Перед тим, як розпочати створення ігрового проекту, важливим етапом є оцінка ринку та проведення аналізу конкурентів. Цей етап допомагає зрозуміти динаміку галузі, визначити ключових учасників та ідентифікувати можливості для вдосконалення та виокремлення продукту.

Після аналізу ігрових сервісів, таких як Steam, Epic Games, Playstation Store та Google Play, стало очевидним, що жанр аркадних ігор володіє величезною популярністю серед геймерської аудиторії. На цих ігрових сервісах представлена велика кількість ігор, які належать до цього жанру, при цьому вони мають велику кількість завантажень. Аналітика даних про попит на комп’ютерні ігри в стилі файтінгу допомогла сформувати просту та складну гіпотезу.

Проста гіпотеза висловлює ідею про те, що середній вік гравців у жанрі аркадного файтінгу становить приблизно 15 років.

Складна гіпотеза формую ідею про те, що гравці виявляють вищий рівень зацікавленості в грі, яка пропонує нові інтересні ігрові механіки, і цей вплив є більш суттєвим для ігор в жанрі аркадного файтінгу, де геймплейна механіка грає ключову роль у задоволенні гравців.

На сьогоднішній день індустрія відеоігор є провідним сегментом у сфері розваг по всьому світу. Кількість геймерів невпинно зростає (3,1 мільярда гравців у 2020 році), а сам геймінг перетворюється на високооплачувану професію, формуючи потужну екосистему.

Світовий ринок комп'ютерних ігор, маючи на увазі продаж ігрового контенту користувачам, стає все глибшим та масштабнішим. За даними аналітичної компанії SuperData, обсяг ринку відеоігор у 2020 році досяг $126,6 мільярда, збільшившись за рік на 12%. Це зростання зумовлене тим, що багато користувачів залишалися вдома через пандемію COVID-19 і розважалися за допомогою цифрових ігор. У 2020 році ринок мобільних ігор зріс на 10% і склав 58% загального обсягу ігрової галузі.

Проаналізуємо ігровий ринок України. Спільнота гейм-девелоперів включає як фрілансерів-одинаків, так і великі компанії, відомі у всьому світі. Вітчизняний геймдев стрімко розвивається: за останні кілька років у сфері розробки ігор з'явилося понад 30 нових компаній. Щорічний дохід від продажу відеоігор в нашій країні досягає 200 мільйонів доларів і другий рік поспіль перевищує загальносвітове зростання індустрії [2].

Серед українських розробників комп'ютерних ігор найбільш популярна платформа Unity (69%), за нею йдуть Unreal Engine та HTML5 (по 31%), крім того, 27% використовують власні платформи. Найпопулярнішими жанрами є Action (63%), Головоломки (59%), Стратегії (52%) та Симулятори (44%). Аудиторія ігор, створених українськими компаніями, налічує 770 мільйонів користувачів. Проаналізуємо вітчизняних розробників ігор.

GSC Game World - найпотужніший український розробник ігор, відомий завдяки таким проектам, як «S.T.A.L.K.E.R.» і «Козаки». Ігри серії S.T.A.L.K.E.R. отримали позитивні відгуки від популярних ігрових сайтів і були добре прийняті критиками. Компанія заявляє, що кількість проданих копій гри S.T.A.L.K.E.R.: Shadow of Chernobyl у світі перевищила 2 мільйони примірників.

4A Games була заснована у 2005 році. Студія відома головним чином завдяки постапокаліптичній серії «Metro», перша частина якої, «Metro 2033», була випущена у 2010 році. Оригінальна Metro 2033 здобула високі оцінки та багато схвальних рецензій, отримавши на агрегаторі Metacritic середню оцінку 81/100 для Windows-версії та 77/100 для Xbox 360.

Frogwares – незалежна українська компанія, яка займається розробкою відеоігор переважно жанру квест. Найбільш відомим проектом студії є серія відеоігор «Пригоди Шерлока Холмса». Загалом Frogwares розробила 8 частин серії, продавши при цьому 7 мільйонів копій. Останнім релізом є відеогра The Sinking City, перша відеогра компанії з повністю відкритим світом, випущена 27 червня 2019 року [6].

Серед розробників мобільних ігор варто відзначити компанію iLogos Game Studios. На її рахунку понад 400 проектів, а найуспішніші з них займають високі

позиції в магазинах мобільних додатків [7]. Серед створених iLogos ігор – Shadow Fight і Shadow Fight 2 (36,5 млн гравців), аркада Vector (11 млн гравців), сітібілдер «Мегаполіс» (20 млн гравців).

Таким чином, можна констатувати, що галузь комп'ютерної індустрії стрімко розвивається, і вагомий внесок у цей розвиток вносять українські розробники ігор. Індустрія рухається в напрямку підвищення реалістичності ігор, їх продуктивності та реалізації штучного інтелекту.

Продукт діяльності великих ігрових компаній – відеоігри, які завоювали популярність не лише серед молоді, а й зацікавили також дорослих [3]. Про це свід-чить щорічне дорослішання аудиторії. Що й не дивно, адже відеоігри можуть зацікавити своєю різноманіт-ністю (рис. 1.1).

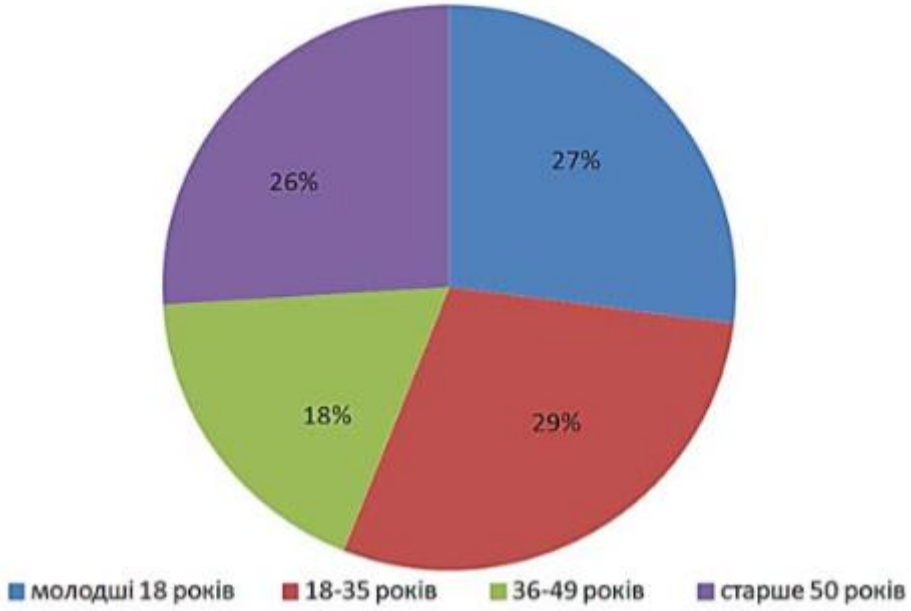


Рисунок 1.1 – Розподіл гравців по віковому показнику

Існує безліч жанрів ігор, наприклад, доросла аудиторія більше зацікавлена в таких жанрах, як стратегії та симулятори, тоді як молодь віддає перевагу пригодницьким та рольовим іграм. Ці жанри є одними з основних у сфері виробництва відеоігор.

На початковому етапі розвитку відеоігри були досить прибутковими, адже собівартість їх розробки була незначною. На той час ігри могли розроблятися невеликою командою з декількох людей, і всього за кілька місяців гра виходила

на ринок. Саме тоді було створено багато відомих сьогодні компаній, таких як Origin Systems, Sierra Entertainment, Capcom, Activision та Electronic Arts. В Україні розробкою відеоігор займаються такі компанії, як Plarium, Playtika, Gameloft, Eforb та Wargaming.net.

Дохід від відеоігор стрімко зростає (див. рис. 1.2). Це пов’язано зі збільшенням популярності відеоігор у світі. Ігрова індустрія постійно розвивається, кожного року виникають нові проєкти що захоплюють людей своїми новинками. Розробники відеоігор вкладають у свої проєкти мільйони доларів.

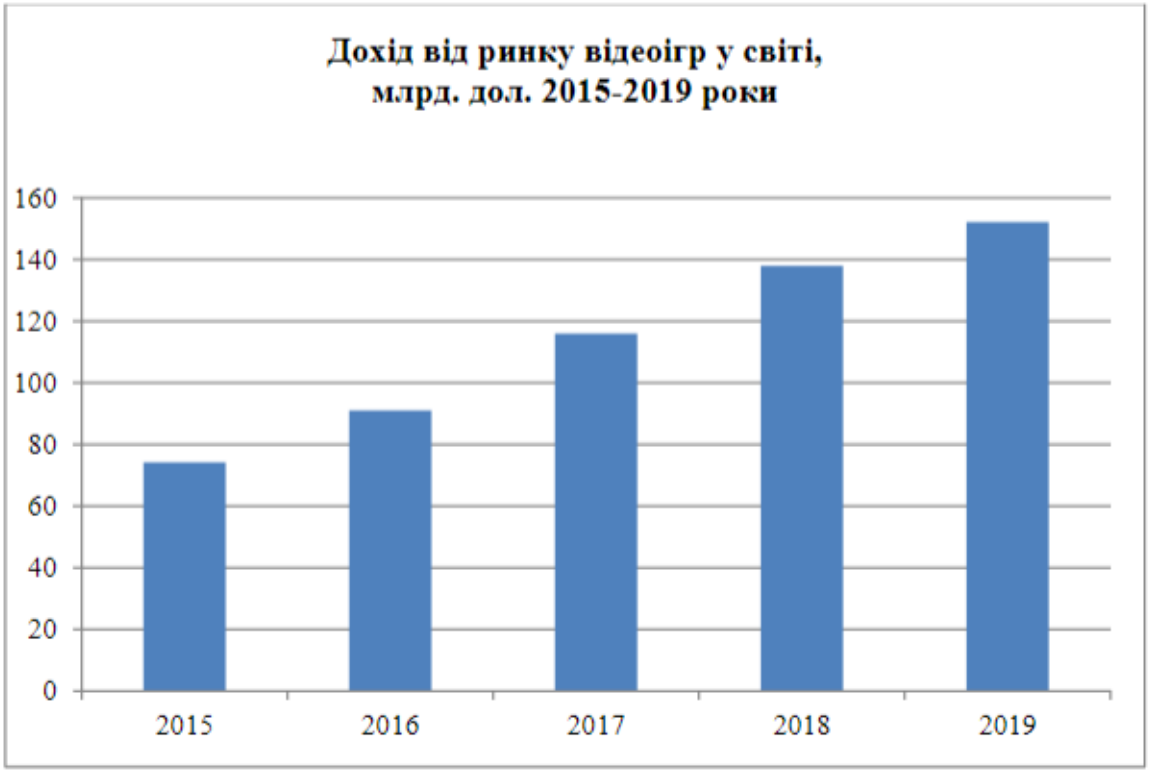


Рисунок 1.2 - Дохід від ринку відеоігор у світі, млрд дол., 2015–2019 рр.

З подальшим розвитком технологій зросли й витрати на виробництво, адже технології вимагали залучення більшої кількості робітників та більше часу на випуск гри. Ланцюжок цінностей ігрової індустрії можна розглядати як шість взаємопов'язаних рівнів, кожен з яких відіграє важливу роль у створенні та просуванні відеоігор.

Найбільшим споживачем відеоігор є Китай, де кількість користувачів Інтернету становить 900,6 млн осіб, а на кінець 2019 р. валовий обсяг продаж становив 36 540 млн дол. Друге місце займають США, а третє – Японія (табл. 1.1).

Таблиця 1.1 - Валовий обіг ігрових індустрій у різних країнах

Країна	Населення, млн.	Кількість інтернет-користувачів, млн.	Валовий обсяг продаж за рік, млн. дол.
Китай	1420,1	900,6	36540
США	329,1	273,7	35510
Японія	126,9	121,2	18683

На сьогодні у світі налічується понад 2,5 мільярда геймерів, які витратили 152,1 мільярда доларів на ігри у 2019 році. Це свідчить про зростання прибутків ігрових компаній на 9,6% порівняно з попереднім роком (див. рис. 1.3).

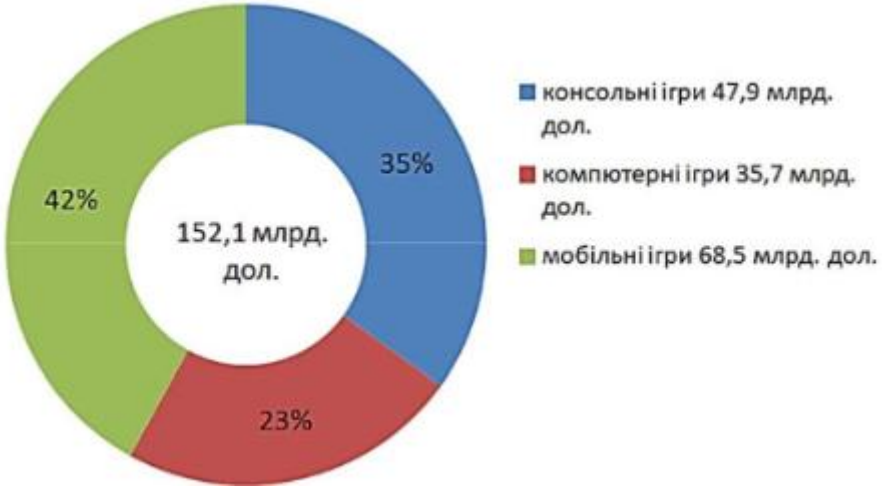


Рисунок 1.3 - Дані розподілу доходу світового ігрового ринку

Ігрові компанії також активно працюють над розширенням та диверсифікацією своїх продуктів, створюючи нові жанри та концепції. Водночас зростає роль соціальних і мультиплеєрних аспектів у відеоіграх, що дозволяє гравцям з усього світу взаємодіяти і змагатися між собою. Такий підхід сприяє формуванню стійких геймерських спільнот, що в свою чергу позитивно впливає на довгостроковий успіх та стабільне зростання ігрової індустрії.

1.2 Аналіз конкурентів

На початковому етапі розробки я провів аналіз конкурентів на ринку. Після аналізу було відібрано 5 ігор від конкурентів, серед яких Street Fighter 6, Drago Ball FighterZ, Mortal Kombat 1, Street Fighter V, Punch Club. Всі вони мають схожий функціонал. Гра, яка розробляється в рамках проекту буде мати назву Fatal Ring.

Порівняльний аналіз дозволяє виявити ключові фактори, які впливають на успіх файтингів на платформі Steam. Основні критерії включають графічне виконання, механіки бою, баланс персонажів, наявність багатокористувацького режиму, частоту оновлень та підтримку спільноти. Ці критерії допомагають оцінити конкурентоспроможність нашого проекту та визначити напрями для покращення (див. таблиця 1.2).

Таблиця 1.2 – Порівняння конкурентів

Аналізований показник	Fatal Ring	Street Fighter 6	DRAGON BALL Fighterz	Mortal Kombat 1	Punch Club	Street Fighters 5
Можливість збереження прогресу у грі	+	+	+	+	+	+
Доступність мультиплеєру	+	+	-	-	+	+
Графічні анімовані ефекти	+	+	-	+	-	+
Кросплатфор-менність	-	-	+	+-	+	-

Всі вищеперераховані програми використовуються на комп’ютерах та доступні на платформі Steam. Ігрові проекти, які розглянуті в якості конкурентів, були обрані за рейтингом популярності та кількості відгуків на платформі. Файтинги, як правило, формують активні спільноти гравців, які беруть участь у турнірах, обговорюють стратегії та діляться своїм досвідом. Важливим елементом успіху є підтримка цієї спільноти через соціальні мережі, форуми та інші платформи. Наявність міцної спільноти може значно підвищити популярність гри та забезпечити її довготривалий успіх.

1.2.1 Аналіз гри Street Fighter 6

Street Fighter 6 - новий виток в легендарній серії файтингів від Capcom, випущений у 2023 році для Windows, PlayStation 4, PlayStation 5 та Xbox.

Ця гра пропонує гравцям свіжий погляд на класичний жанр, зберігаючи при цьому фундаментальні принципи, які зробили серію популярною.

Ядро геймплейу ґрунтується на системі Drive Gauge, що поощрює творчий підхід гравців. Манометр дозволяє використовувати п'ять різних прийомів, вимагаючи від гравців стратегічних виборів. Це дозволяє кожному поєдинку бути унікальним і насиченим тактичними маневрами. У грі реалізована можливість використання декількох суперкомбо, що додає стратегічні аспекти в битви (див. рис. 1.4).

Рисунок 1.4 – Сцена з гри Street Fighter 6

До переваг гри можна віднести збереження традиційних елементів разом із нововведеннями в системі, великий вибір режимів гри, насичена графіка та реалістичний дизайном персонажів і арен.

До недоліків можна віднести обмеження кількості супер ударів, це може призвести до відсутності варіативності в грі для деяких гравців, незручне меню вибора персонажів, велика вартість гри, забагата кількість ефектів.

1.2.2 Аналіз гри Punch Club

Punch Club є спортивним симулятором розробників Lazy Bear Games та видавця tinyBuild. Вийшла 9 січня 2016 року для персональних комп'ютерів та згодом на інших платформах. Гравець управляє боксером, який тренується, розвиває фізичні навички та бере участь в боксерських змаганнях, як частина сюжету гри (див. рис. 1.5).

Рисунок 1.5 – Сцена з гри Punch Club

До переваг цієї гри можна віднести стилізацію, лінійну логіку прокачування героїв, сюжетність гри. Гра здобула визнання за піксельне графічне оформлення та стильні відсилання до ретрофільмів і мультфільмів. Розробники вдало внесли елементи 80-х і 90-х, роблячи гру цікавою для фанатів цього періоду.

До недоліків цієї гри можна віднести монотонний геймплей, неоднозначність бойової системи.

1.2.3 Аналіз гри Dragon Ball Fighterz

Dragon Ball Fighterz – це гра в стилі файтингу, яка дозволяє гравцям вивести до бою до шести героїв зі світу Dragon Ball Гра акцентує на командному геймплеї, де гравець може обирати і перемикатися між своїми героями, створюючи тактичні комбінації та стратегії. Геймплей використовує універсальні рухи, які дозволяють гравцям виконувати різноманітні комбо, розширюючи можливості стратегій у бою (див. рис. 1.6).

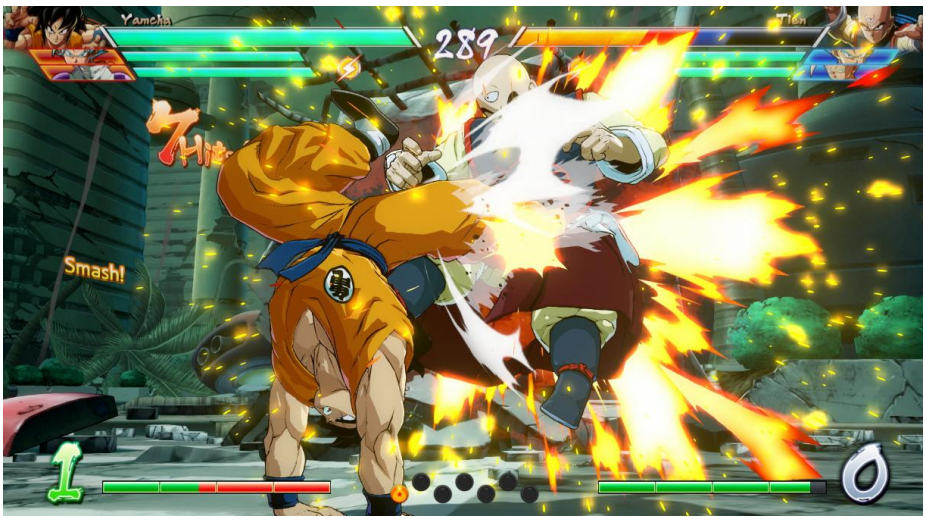


Рисунок 1.6 – Сцена з гри DRAGON BALL Fighterz

До переваг ігри можна віднести якісну графіку та анімації, цікавий підхід до реалізації геймплею, можливість гри онлайн. Комбінація командної механіки та універсальних рухів робить геймплей експресивним та захоплюючим.

До недоліків можна віднести складність керування та специфічні механіки командного геймплею. Для новачків може бути важким освоєння складних комбо та тактик командного геймплею.

1.2.4 Аналіз гри Mortal Kombat 1

Mortal Kombat 1 — дванадцята частина культової серії файтингів, розроблена студією NetherRealm Studios та опублікована Warner Bros. Games.

Mortal Kombat 1 відзначається за свою відому кровопролитну боротьбу та інтенсивний геймплей, що став підписним стилем серії.

Завдяки новим технологіям графічного движка, гра вражає захоплюючою графікою та реалістичним дизайном персонажів. Крім того, додані нові комбінації ударів та добивань, що роблять гру ще більш захопливою для шанувальників (див. рис. 1.7).

Рисунок 1.7 – Сцена з гри Mortal Kombat 1

Гравці можуть насолоджуватися глибокою сюжетною лінією та різноманітними режимами гри, включаючи мережеві турніри та тренувальні вправи. Сюжетна лінія Mortal Kombat 1 розгортається у захоплюючому світі, де кожен персонаж має свою унікальну історію, мотивації та цілі. Це дозволяє гравцям поринути у всесвіт гри, розкриваючи нові деталі та повороти сюжету з кожним пройденим рівнем. Такий підхід робить гру не лише змаганням, але й захоплюючим кінематографічним досвідом. Розробники також постійно оновлюють гру, додаючи нових персонажів та вміст, що забезпечує довготривалу цікавість і свіжість ігрового процесу.

1.2.5 Аналіз гри Street Fighters 5

Street Fighter 5 — аркадний файтинг з видом збоку в популярній серії Street Fighter. Гра пропонує просту та зрозумілу бойову систему з шістьма базовими ударами та кількома загальними спецприйомами, що полегшує освоєння основ.

Впровадження нових прийомів значно збагачує бойову механіку, додаючи нові можливості для унікальних атак, посилення стандартних ударів та контратак.

У простій та зрозумілій формі гра ознайомить вас з основами бойової системи: шість базових ударів, кілька загальних для більшості персонажів спецприйомів. При цьому складні та незрозумілі для новачка дії залишаються наче за кадром, щоб поступово вивчити їх під час подальшого проходження.

Управління багатьма персонажами було змінено, що оновлює ігровий процес, зокрема для тих, хто звик до класичних бійців. Нові комбінації клавіш і поліпшення в управлінні роблять бої більш плавними та інтуїтивно зрозумілими, хоча це може вимагати певного часу для адаптації навіть від досвідчених гравців (див. рис. 1.8).

Рисунок 1.8 – Сцена за гри Street Fighters 5

Серед переваг можна виділити доступність для новачків, розширену бойову систему, еволюцію класичних персонажів, винагороди за перемоги, цікаві концепції. Доступність для новачків забезпечується за рахунок інтуїтивно зрозумілого управління і навчальних режимів, які дозволяють швидко освоїти основи гри.

Серед недоліків можна виділити проблеми з управлінням, рівень опонентів зростає не поступово, а стрибками, що може викликати дискомфорт у гравців. Сюжетна лінія гри дуже слабка та передбачена.

1.3 Розрахунок ергономічності та якості програмного забезпечення

Для оцінки ергономічності та якості програмного забезпечення з точки зору користувача я сформулюю ряд характеристик програмного продукту:

k_1 - цікавий сюжет;

k_2 - зручний інтерфейс;

k_3 - якісна графіка;

k_4 - система бонусів і нагород у грі;

k_5 - надійність програмного забезпечення.

Для проведення оцінки було залучено кілька експертів, які оцінили кожну з вказаних характеристик для різних ігрових проектів. Оцінки проводилися на основі ваг, присвоєних кожній характеристиці, що дозволяє відобразити їх відносну важливість. Експерти обиралися довільно, кожен експерт має свій ігровий досвід та вміння. Для отримання більш точної інтегральної оцінки ігрові за стосунки були оцінені за вищезазначеними характеристиками.

Таблиця 1.3 - Експериментальні оцінки вимог користувача

Характе-ристика	Вага	Варіант проекту						Інтегральна оцінка					
		Fatal Ring	Street Fighter6	Dragon Ball	Mortal Kombat1	Street Fighter5	Punch Club	Fatal Ring	Street Fighter6	Dragon Ball	Mortal Kombat1	Street Fighter6	Punch Club
Експерт 1													
k_1	0.2	30	30	40	50	30	40	6	6	8	10	6	8
k_2	0.2	50	40	50	40	50	40	10	8	10	8	10	8
k_3	0.25	40	40	40	40	36	20	10	10	10	10	9	5
k_4	0.1	50	40	30	50	50	30	5	4	3	5	5	3
k_5	0.25	40	40	20	36	40	48	10	10	5	9	10	12
Загалом	1							41	38	36	42	40	36

Таблиця 1.4 - Експериментальні оцінки вимог користувача

Характеристика	Вага	Варіант проекту						Інтегральна оцінка					
		Fatal Ring	Street Fighter6	Dragon Ball	Mortal Kombat1	Street Fighter5	Punch Club	Fatal Ring	Street Fighter6	Dragon Ball	Mortal Kombat1	Street Fighter6	Punch Club
Експерт 2													
k_1	0.1	30	30	20	40	40	30	3	3	2	4	4	3
k_2	0.3	50	50	60	30	60	50	15	15	18	9	18	15
k_3	0.1	30	50	30	40	40	60	3	5	3	4	4	6
k_4	0.15	60	60	40	40	60	40	9	9	6	6	9	6
k_5	0.35	60	40	40	40	40	60	21	14	14	14	14	21
Загалом	1							51	46	43	37	49	51

Таблиця 1.5 - Експериментальні оцінки вимог користувача

Характеристика	Вага	Варіант проекту						Інтегральна оцінка					
		Fatal Ring	Street Fighter6	Dragon Ball	Mortal Kombat1	Street Fighter5	Punch Club	Fatal Ring	Street Fighter6	Dragon Ball	Mortal Kombat1	Street Fighter6	Punch Club
Експерт 3													
k_1	0.2	60	20	30	50	60	40	12	4	6	10	12	8
k_2	0.2	60	30	20	30	50	40	12	6	4	6	10	8
k_3	0.2	60	60	50	50	50	50	12	12	10	10	10	10
k_4	0.2	30	50	30	30	50	50	6	10	6	6	10	10
k_5	0.2	30	40	40	40	30	30	6	8	8	8	6	6
Загалом	1							48	40	34	40	48	42

Таблиця 1.6 - Експериментальні оцінки вимог користувача

Характе-ристика	Вага	Варіант проекту						Інтегральна оцінка					
		Fatal Ring	Street Fighter6	Dragon Ball	Mortal Kombat1	Street Fighter5	Punch Club	Fatal Ring	Street Fighter6	Dragon Ball	Mortal Kombat1	Street Fighter6	Punch Club
Експерт 4													
k_1	0.4	50	30	40	50	50	50	20	12	16	20	20	20
k_2	0.1	50	50	50	50	50	50	5	5	5	5	5	5
k_3	0.2	40	50	40	50	50	50	8	10	8	10	10	10
k_4	0.1	40	50	40	30	30	30	4	5	4	3	3	3
k_5	0.2	30	30	30	30	30	30	6	6	6	6	6	6
Загалом	1							43	38	39	44	44	44

Таблиця 1.7 – Визначення інтегральної оцінки

	Вага	Fatal Ring	Street Fighter 6	Dragon Ball	Mortal Kombat 1	Street Fighter 5	Punch lub	Fatal Ring	Street Fighter 6	Dragon Ball	Mortal Kombat 1	Street Fighter 5	Punch lub
Експерт 1	0.3	41	38	36	42	40	36	10.6	11.4	10.8	12.6	12	10,8
Експерт 2	0.2	51	46	43	37	49	51	10.2	9.2	8.6	7.4	9.8	10.2
Експерт 3	0.2	48	40	34	40	48	42	9.6	8	6.8	8	9.6	8.4
Експерт 4	0.3	43	38	39	44	44	44	12.9	11.4	11.7	13,2	13,2	13,2
Середня	1							10,8	10	9,5	10.3	11.1	10.6

Проект, який розробляється входить в трійку лідерів по оцінюванню, а саме займає друге місце. Це свідчить про конкурентоспроможність ігрового продукту та його актуальність.

2 ФОРМУВАННЯ ВИМОГ ДО ПРОГРАМНОЇ СИСТЕМИ

2.1 Цільова аудиторія

Цільова аудиторія: хлопці і дівчата, чоловіки і жінки 10-35 років. Основна привабливість – різноманітні ігрові механіки, просте і зрозуміле керування, бої у реальному часі. популярність гри та забезпечити її довготривалий успіх.

2.2 Unique selling proposition

- Унікальні ігрові персонажі;
- вибір ігрових арен;
- унікальні анімовані ефекти атак;
- динамічні бої;
- зручне керування .

2.3 Час ігрової сесії

Гра розрахована на швидкі ігрові сесії, що є адаптованим під вподобання дітей та підлітків, які хочуть швидко отримати яскраві емоції. Одна ігрова сесія розрахована на 3 – 5 хвилин ігрового часу. Після закінчення бою гравцям пропонується зіграти реванш чи перейти до головного меню гри.

2.4 Технологічні характеристики

Гра доступна для операційної системи Windows 10 та новіших версій на ПК. Програмний продукт являє собою 2D ігровий застосунок, розроблений на рушії Unity. Unity є потужною платформою для створення інтерактивних 2D ігор, що дозволяє реалізувати складні ігрові механіки та забезпечити високу якість графіки. Гра доступна для операційної системи Windows 10 та новіших версій на ПК. Програмний продукт являє собою 2D ігровий застосунок, розроблений на рушії Unity, що дозволяє реалізувати складні ігрові механіки та забезпечити високу якість графіки.

2.5 Ігровий процес

Гра відбувається у реальному часі у двовимірному просторі. Гравець контролює персонажа, якому доступні біг, стрибок, регенерація здоров’я, декілька видів атак. Кожна атака наносить різну шкоду супротивнику та супроводжується унікальними анімованими ефектами. На ігровій арені опиняються два персонажа, які були вибрані ігроками і починають бій.

Гравець має два головних ресурси. Перший ресурс гравця – це здоров’я Гравець може відновлювати цей ресурс один раз за всю гру. Другий ресурс гравця - та сила, яка накопичується для супер удару, який наносить максимальну шкоду супротивнику. Також гравець може заплющити очі, знищивши своє коло світла, або навпаки розширити його.

Ігровий процес являє собою битву між двома ігроками. Коли два ігроки обрали свого персонажа, вони переходять до етапу вибору арени. Вибір арени впливає на стратегію гри, оскільки різні оточення можуть мати специфічні перешкоди або особливості, які гравці можуть використовувати на свою користь. Кожна арена має свій унікальний дизайн та ефекти. Бій складається із трьох раундів. Кожен раунд триває 90 секунд. В раунді перемагає той гравець, який наніс 100% шкоди супротивнику або той гравець, у якого на момент закінчення часу відведеного на один раунд залишилося більше здоров’я. Шкала здоров’я для кожного гравця та час тривалості раунду відображаються в верхній частині екрана. Кожна перемога в раунді візуалізується зірочкою, яка відображається під шкалою здоров’я гравця. В бою перемагає той гравець, який перемогає в двох раундах.

На початку кожного раунду відображається анімація ініціалізації раунду. Після закінчення бою відображається інформація про переможця та можливість почати реванш чи перейти до головного меню гри.

Такий підхід забезпечує плавний перехід між ігровими сесіями та дозволяє гравцям швидко повернутися до гри або налаштувань, зберігаючи інтерес і динаміку ігрового процесу.

2.6 Ігрові механіки

2.6.1 Загальні ігрові механіка

В ігровому застосунку реалізован розрахунок відстані від персонажу до полу, який використовується для перевірки доступності стрибка, щоб уникнути подвійних стрибків та створення ефекту неприродної поведінки персонажа.

В ігровому застосунку реалізован розрахунок відстані між персонажами, який використовується для перевірки можливості нанесення шкоди під час атаки, цей показник розраховується кожні 0,01 секунду.

2.6.2 Ігрові механіки гравця

- Біг;
- стрибок;
- атака мечем – базова атака гравця, яка наносить шкоду супротивнику 3-5% в залежності від відстані від персонажами;
- відновлення сил після базової атаки – гравець не може безперервно наносити атаки, для відновлення сили. Щоб знову нанести атаку йому необхідний час, який складає 1 секунду;
- атака вогняним дощем – додаткова атака гравця, яка наносить шкоду супротивнику у розмірі 5%, при умові, що супротивник знаходиться в зоні ураження;
- атака електро-заморозка – додаткова атака гравця, яка наносить шкоду супротивнику у розмірі 25%, у разі якщо він знаходиться у зоні ураження, під час цієї атаки супротивник отримує шкоду, перестає мати можливість рухатись, атакувати на протязі двох секунд;
- супер атака – додаткова атака гравця, яка наносить шкоду супротивнику від 25 до 40% в залежності від відстані між ігроками, для вдалої атаки гравцю треба впритул підійти до супротивника, атака супроводжується потужним вибухом, доступна один раз за один раунд;

- здоров’я – головний ресурс гравця, зменшується при пошкодженнях, герой програє у раунді, коли здоров’я досягає 0;
- регенерація – якщо гравець отримав пошкодження, він має можливість один раз за бій використати функцію відновлення здоров’я, при її використанні здоров’я гравця відновлюється до 100%, регенерація супроводжується анімованим ефектом.

Усі ці ігрові механіки спільно створюють глибокий і відчутний ігровий досвід, що дозволяє гравцям імплементувати різні стратегії та тактики. Взаємодія з різними видами атак, відновлення здоров'я та використання унікальних можливостей персонажів підкреслює індивідуальність кожного героя і стимулює до вдосконалення геймплею. Розмаїття можливостей і стратегій відкриває безліч можливостей для гравців у досягненні перемоги та насолоди від ігрового процесу.

2.7 Опис ігрових елементів

2.7.1 Персонажі

Hero - герой гри, яким керує перший гравець. Він може бігати, стрибати, атакувати, використовувати спеціальні атаки та здібності.

Hero2 – другий персонаж, з яким взаємодіє перший герой. Він може бігати, стрибати, атакувати, використовувати спеціальні атаки та здібності.

2.7.2 Ігровий світ

Ігрові арени – це ігрові рівні, на яких розгортаються події гри. Містять різноманітні об’єкти та елементи оточення, такі як підлога, елменти декору.

2.7.3 Об’єкти ігрового світу

- explosionPrefab - префаб вибуху, який створюється під час спеціальної атаки;

- healingPrefab - префаб, що відповідає за анімацію відновлення здоров'я героя;
- getDammagePrefab - префаб, що відповідає за анімацію отримання ушкоджень;
- electroHitPrefab - префаб для спецільної атаки;
- IceFlorPrefab - префаб для спеціальної атаки.

2.7.4 Інтерфейс користувача (UI):

- Індикатори здоров'я - відображення поточного рівня здоров'я гравця;
- індекатор часу – відображення поточного часу до закінчення раунду.

2.7.5 Анімації героя

- Анімації для різних станів героя, таких як біг, стрибок, атака, спеціальні атаки та інше;
- візуальні ефекти - ефекти вибухів, електроударів, крижаних атак та отримання ушкоджень.

2.7.6 Тригери та події

- Атаки та спеціальні атаки - події, що запускаються при натисканні певних клавіш для різних видів атак;
- взаємодія з об’єктами - події, що відбуваються при зіткненні з сундуками.
- поводження NPC - логіка поведінки іншого героя (Hero2), яка може включати отримання ушкоджень, заморожування та інші реакції на дії головного героя.

2.7.7 Фізичні елементи

- Фізика руху - реалізація бігу, стрибків, падіння та інших дій героя з використанням фізики Unity;

– перевірка на землю - логіка для перевірки, чи знаходиться герой на землі (GroundCheck).

Фізика руху в грі реалізована за допомогою можливостей рушія Unity.

Перевірка на землю (GroundCheck) - це логіка, яка визначає, чи знаходиться герой на землі. Вона забезпечує коректну роботу стрибків, падінь і інших дій, що залежать від контакту з поверхнею.

Загальний опис ігрових елементів надає повний образ та структуру ігрового середовища, яке забезпечує живе та динамічне ігрове досвід. Від детально розроблених персонажів до різноманітних об’єктів і інтерфейсу користувача, кожен елемент спільно додає до глибини та іммерсивності ігрового світу. Префаби вибухів та інші візуальні ефекти роблять ігрові події захоплюючими і ефектними, підкреслюючи важливі моменти в бою та взаємодії між персонажами.

Наявність фізики руху та перевірки на землю дозволяє реалістично відтворювати рухи героїв, що зроблює геймплей більш динамічним та непередбачуваним. Тригери та події, що викликаються під час атак та взаємодії з об’єктами, розширюють можливості стратегічного планування та реалізації тактичних прийомів у бою. Інтеграція цих елементів у єдине ціле створює ігровий досвід, який заохочує до вдосконалення та вивчення нових можливостей кожного персонажа та ігрового рівня. Завдяки ігровим аренам і різноманітним об’єктам ігрового світу гравці отримують можливість досліджувати різні території та взаємодіяти з навколишнім середовищем, що сприяє глибокому зануренню у фантастичний світ гри.

В бою перемагає той гравець, героя якого переміг в двох раундах. Після закінчення бою гравці переходять на сцену, на якій відображається інформація про переможця та дві кнопки, за допомогою яких гравці можуть почати реванш або закінчити бій і перейти до головного меню.

В ігрі представлені чотири персонажі. Вони намальовані власноруч. Створені вони в єдиному дизайнерському стилі. В ігрі представлені чотири персонажі. Вони намальовані власноруч. Створені вони в єдиному дизайнерському стилі.

3 АРХІТЕКТУРА ТА ПРОЕКТУВАННЯ

3.1 Проектування архітектури программного забезпечення

Перед початком розробки ігрового застосунку в жанрі аркадного файтінгу були визначені основні функції зі сторони гравця. Після детального аналізу була створена Use-case діаграма (див. рис. 3.1).

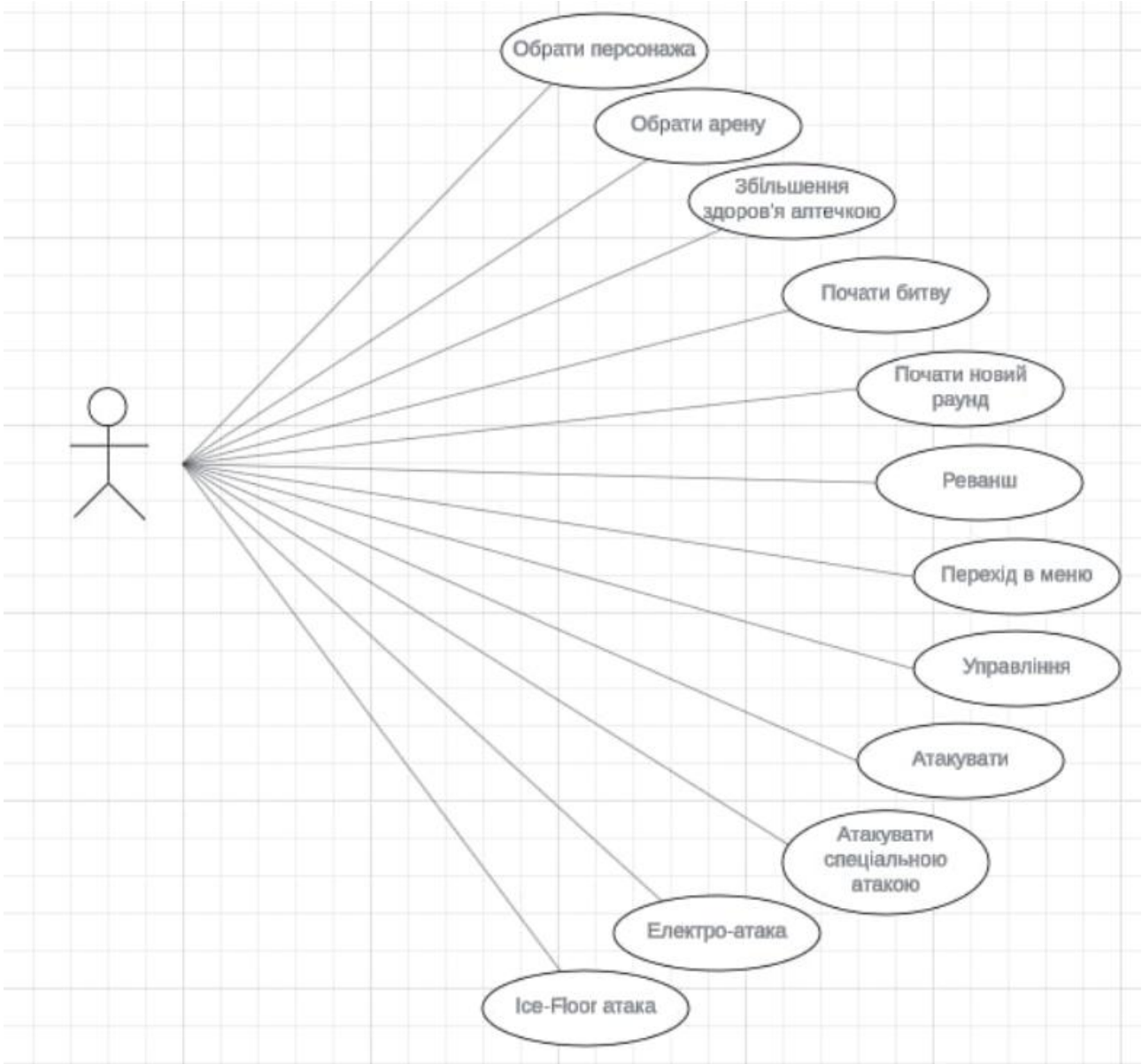


Рисунок 3.1 – Use-case діаграма ігрового застосунку

Користувачами ігрового застосунку є два гравця. При завантаженні гри гравці по черзі обирають собі персонажів, обирають арену для битви та починають бій. Усі ці дії та взаємодії гравців з інтерфейсом і ігровими механіками детально описані на діаграмі класів (див. рис. 3.2).

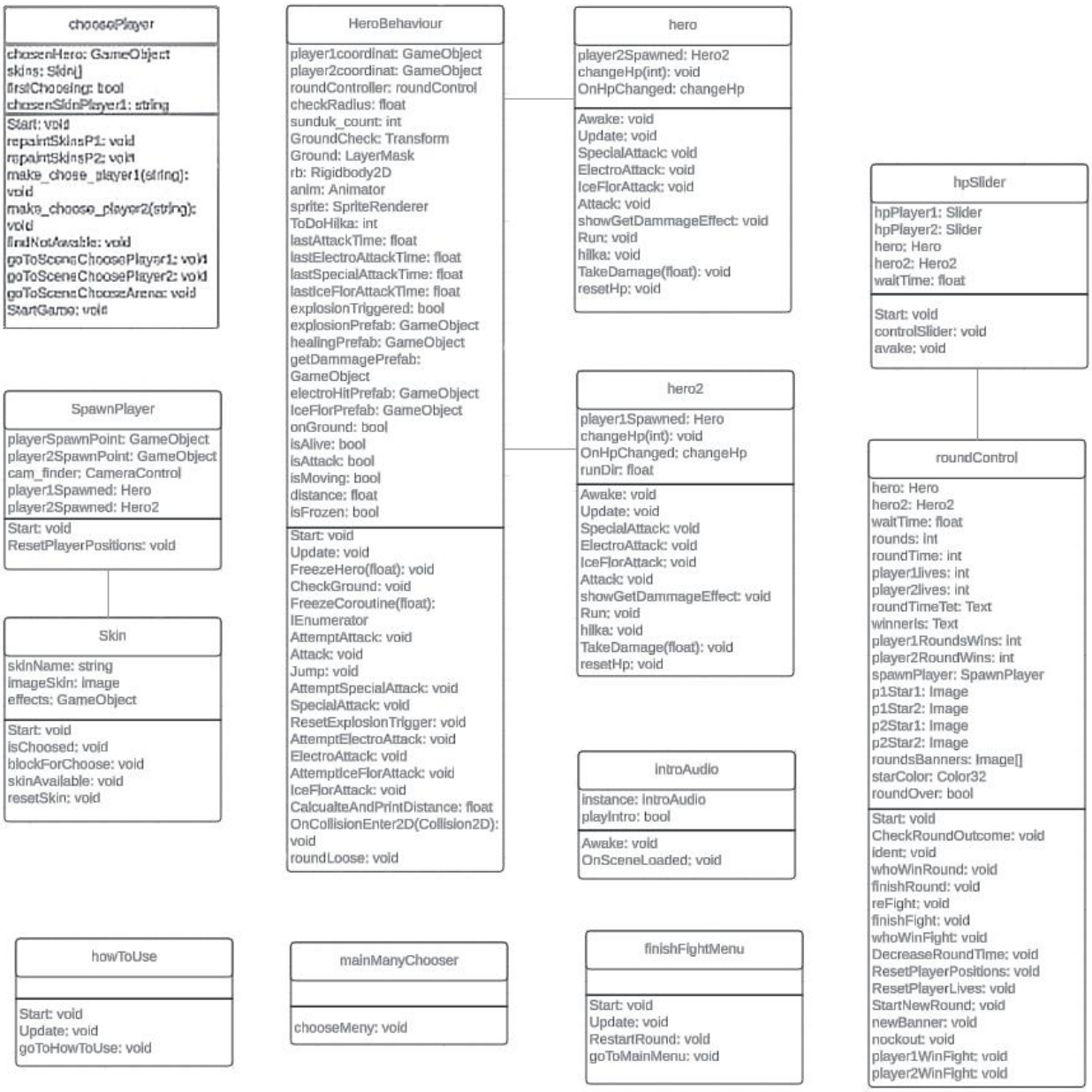


Рисунок 3.2 – Діаграма класів

Клас HeroBehaviour служить базовим класом для класів Hero і Hero2. Він містить загальні методи і властивості, які використовуються обома типами персонажів. Основні властивості, такі як lives, speed, jumpForce, rb (Rigidbody2D), anim, та sprite (SpriteRenderer), оголошені в цьому класі та використовуються похідними класами.

Метод CheckGround використовується для перевірки, чи знаходиться персонаж на землі. Ця перевірка необхідна для реалізації багатьох аспектів ігрової механіки, таких як можливість виконання стрибка, анімації руху та інші фізичні взаємодії. Наведемо програмну реалізацію функції CheckGround:

```
public void CheckGround()
{
    onGround = Physics2D.OverlapCircle(GroundCheck.position, checkRadius, Ground);
}
```

Метод CalculateAndPrintDistance використовується для обчислення відстані між двома гравцями. Це значення необхідне для того, щоб визначити, чи знаходяться гравці на достатній відстані один від одного для нанесення шкоди при атаці. Наведемо програмну реалізацію функції CalculateAndPrintDistance:

```
public float CalculateAndPrintDistance()
{
 if (player1coordinat != null && player2coordinat != null)
{
   distance = Vector3.Distance(player1coordinat.transform.position, player2coordinat.transform.position);
   return distance;
}
 return 100f;
}
```

Якщо обидва гравці ініціалізовані, метод використовує Vector3.Distance для обчислення відстані між ними.

Vector3.Distance - це метод, який обчислює евклідову відстань між двома точками в тривимірному просторі. Він приймає дві позиції (координати) і повертає відстань між ними як число типу float.

Відстань, обчислена за допомогою Vector3.Distance, зберігається у змінній distance і повертається методом.

Змінна distance використовується для того, щоб визначити, чи знаходяться гравці на достатній відстані один від одного для нанесення шкоди при атаці. Наприклад, якщо гравці знаходяться достатньо близько один до одного, персонаж може виконати атаку і нанести шкоду супротивнику. Якщо ж вони знаходяться надто далеко, атака не спрацює.

Метод hilka в обоих классах Hero и Hero2 відповідає за відновлення життя героя. якщо герой не заморожений, метод перевіряє, чи є ще можливість відновити життя. Якщо так, він збільшує життя героя до максимального значення, оновлює значення HP за допомогою події OnHpChanged і створює візуальний ефект відновлення життя (наприклад, вибух чи анімація).

Після відновлення життя кількість можливих відновлень зменшується на одиницю (ToDoHilka--), щоб герой не міг безкінечно відновлювати себе. Наведемо програмну реалізацію функції hilka:

```
public void hilka()
{
    if(isFrozen)
        return;

    if (ToDoHilka > 0)
    {
        lives = 100;
        OnHpChanged?.Invoke(lives);
        GameObject explosion = Instantiate(healingPrefab,
transform.position, Quaternion.identity);
        Destroy(explosion, 1.5f);
        ToDoHilka--;
    }
}
```

Коли гравець обирає свого персонажа, метод make_choose_player1 викликається з назвою обраного персонажа як параметр, і цей персонаж стає активним для першого гравця.

Таким чином, код дозволяє гравцям обирати свої шкірні покриви, які вони хочуть використовувати в грі, і зберігає ці налаштування для подальшого використання. Наведемо програмну реалізацію функції make_choose_player1:

```
public void make_choose_player1(string Choosed_Player)
{
    PlayerPrefs.SetString("choosed_player", Choosed_Player);
    chosenSkinPlayer1 = Choosed_Player; //add
    Debug.Log("choosed_player" + Choosed_Player);
    repaintSkinsP1();
}
```

Класи hero, hero2 мають подію OnHpChanged, яка відбувається при зміні кількості здоров'я героя. Ця подія викликається при зменшенні кількості здоров'я об'єкта героя, і підписники можуть реагувати на цю подію, оновлюючи шкалу

здоров'я героя на екрані. Ця подія визначена в класі героя як public event changeHp OnHpChanged;. Вона оголошує делегат changeHp, який приймає цілочисельний параметр - кількість здоров'я героя.

Наведемо приклад реалізації проведення підписки на подію:

```
OnHpChanged += UpdateHealthBar;
```

Метод UpdateHelathBar приймає нове значення кількості здоров'я героя і оновлює відображення шкали здоров'я на екрані відповідно до цього значення. Наведемо приклад реалізації методу UpdateHealthBar:

```
void UpdateHealthBar(int health)
{
 healthBar.value = health;
}
```

Під час гри, коли кількість здоров'я героя змінюється (наприклад, при отриманні урону), викликається подія OnHpChanged, передаючи нове значення кількості здоров'я підписникам. Наведемо приклад виклику події оновлення інформації о здоро'ї гравця:

```
OnHpChanged?.Invoke(lives);
```

Під час гри, коли один з героїв (або обидва) втрачають усе здоров'я або закінчується час раунду, викликається метод CheckRoundOutcome().Наведемо приклад реалізації методу CheckRoundOutCome():

```
if (roundOver) return;
if (hero.lives <= 0 || hero2.lives <= 0 || roundTime <= 0)
{
    roundOver = true;
    whoWinRound();
    if (player1RoundWins >= 2 || player2RoundWins >= 2)
    {
        finishFight();
    }
}
```

В цьому методі перевіряється кількість здоров'я кожного героя та час раунду. Якщо кількість здоров'я одного з героїв стає менше або рівно 0, або закінчується час раунду, то раунд вважається завершеним.

Після завершення всіх раундів перевіряється, який гравець здобув 2 перемоги у раундах. Якщо один з гравців досяг цього результату, викликається метод finishFight(), який визначає переможця бою.

Після закінчення кожного раунду, гравці повертаються на початкові позиції, їхні життя повертаються до початкового стану, і починається наступний раунд. Якщо немає необхідності в новому раунді, і один з гравців вже здобув перемогу у бою, то виконується перехід до іншої сцени, де відображається переможець (див. рис. 3.3).

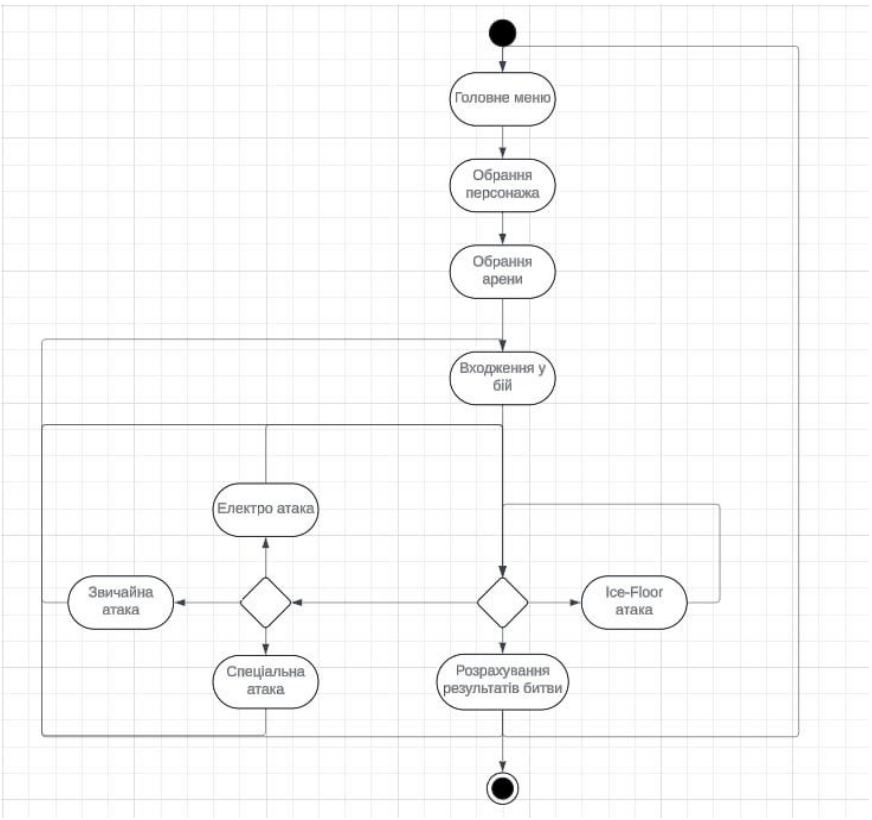


Рисунок 3.3 – Діаграма діяльності

Завантаживши гру, користувача зустрічає головне меню, на якому розташована кнопка «Старт» та «Керування». Кнопка «Керування» переводить користувача на сцену, де демонструється інформація про керування грою для ігрока 1 та ігрока 2. Після ознайомлення з інформацією про керування,

користувач може перейти в головне меню, натиснувши кнопку «Назад». Реалізація переходу до меню керування гравцями реалізована в скрипті howToUse.cs.

Кнопка «Старт» в головному меню гри запускає ігрову сесію та переводить користувача до сцени, на якій перший ігрок обирає свого персонажа для гри. На сцені представлені чотири героя, одного з яких перший гравець може вибрати для бою. Вибраний герой ініціалізується відповідною анімацією при натисканні на нього. Щоб зберегти свій вибір, перший гравець має натиснути кнопку «Next». Ця кнопка переводить користувачів на наступну сцену, на якій свого героя обирає другий гравець. Персонаж, вибраний першим гравцем, на цій сцені не доступний для вибору. Він не відображається на сцені, лише його контур .

В структурі проекту створена дека з персонажами, в ній всі чотори героя доступні в деці для Player1 та Player2 у вигляді префабів. Ця логіка реалізована в скрипті skin.cs. Вибору гравців зберігаються і передаються на сцену бою. Реалізація збереження персонажів реалізована в скрипті SpawnPlayer

Після того, як два гравця обрали своїх героїв, вони переходять на сцену вибору арени для бою, на якій представлені три кнопки з зображеннями та назвами арен. Обравши одну з них, гравці переходять на відповідну сцену боя. Функціонування вибору арени реалізовано в скрипті chooseArena.cs

На кожній сцені боя створені точки для спауна першого та другого героя. В ці точки завантажуються відповідні префаби з дек з героями Player1 та Player2. Герой першого гравця позначається тегом Player1Choosed. Герой другого гравця позначається тегом Player2Choosed. Поведінка героїв реалізована в скрипті HeroBehaviour, який успадковують клас Hero та Hero2.

Бій складається із трьох раундів, кожен раунд починається з відпвідної анімованої заставки. Раунд триває 90 секунд. Раунд вважається завершеним, коли закінчується час раунди або здоров’я одного з героїв дорівнює 0. В разі, якщо час раунду вичерпаний, а здоров’я першого героя дорівнює здоров’ю другого героя, то починається реванш. Якщо час раунду вичерпаний, в раунді перемагає той гравець, здоров’я героя якого більше (див. рис. 3.4).

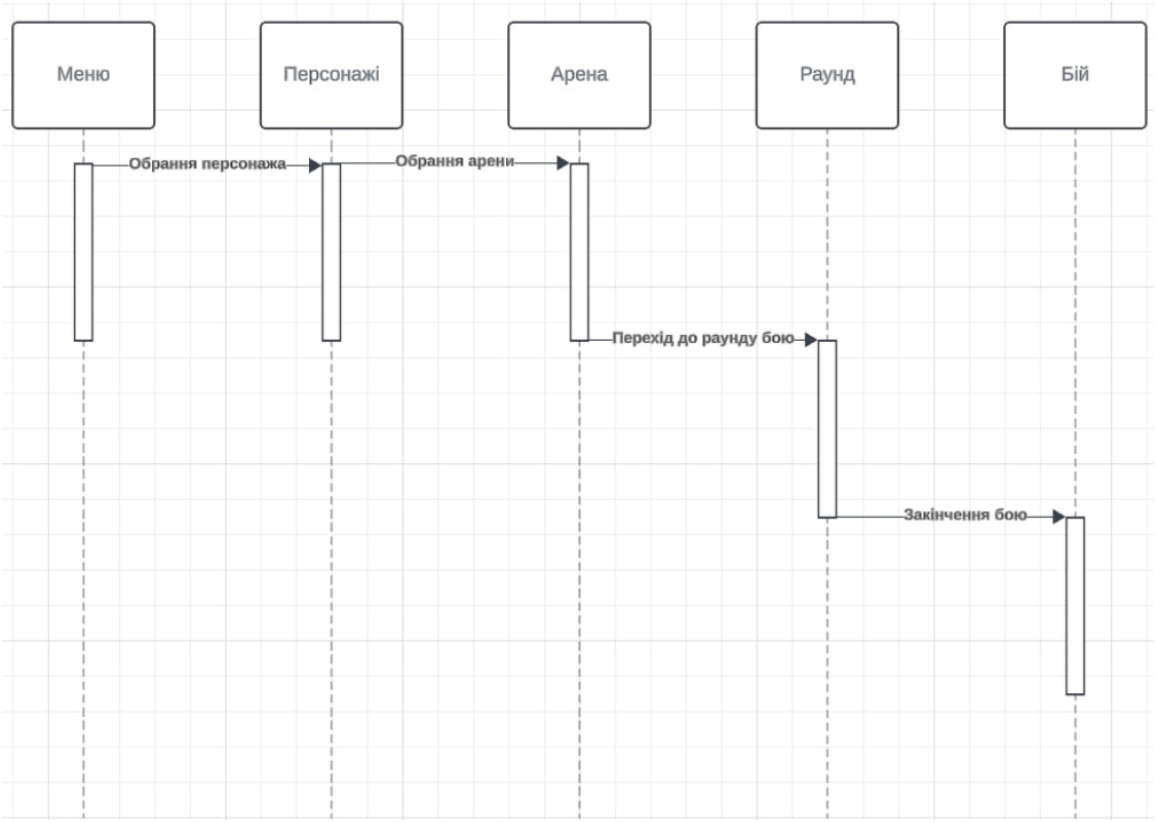


Рисунок 3.4 – Sequence diagram

Шкала здоров’я першого і другого героя відображаються в верхній частині екрана. За логіку відображання інформації про здоров’я героя відповідає скрипт hpSlider.cs. Кожна перемога ідентифікується зірочкою, яка відображається під шкалою здоров’я героя. Після завершення раунда, викликається подія оновлення здоров’я та регенерації гравців у точках спауна.

3.2 Візуальна складова

3.2.1 Оточення

Для візуалізації ігрового оточення використовуються тайлсети, створені елементи декорації рівнів, це створює відповідно атмосферу занурення у гру. Візуалізація ігрового оточення є ключовим елементом у створенні захоплюючої атмосфери для гравців, і використання тайлсетів та декоративних елементів має вирішальне значення в цьому процесі (див. рис. 3.5).

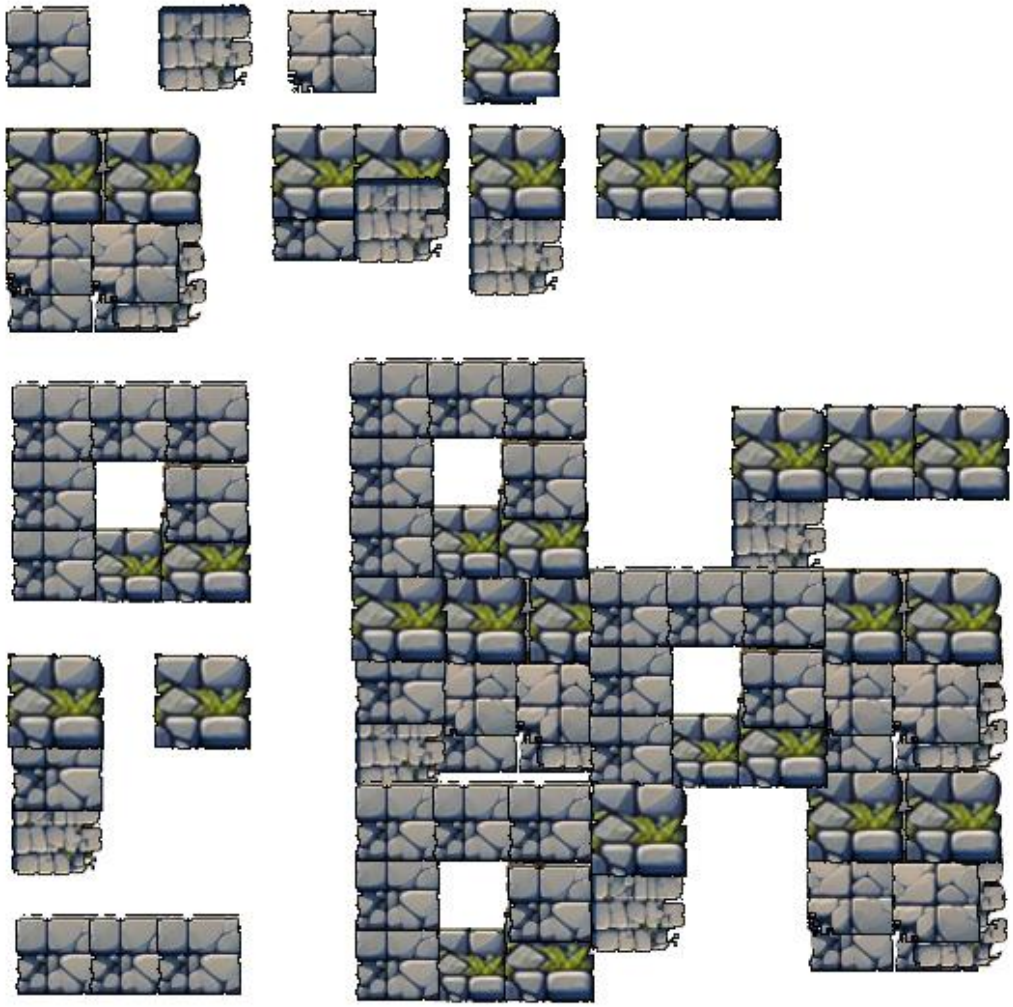


Рисунок 3.5 - Тайлест для створення підлоги

Використання декоративних елементів дозволяє створювати унікальну атмосферу гри. Це впливає на настрій гравців і їхнє сприйняття ігрового світу.

3.2.2 Персонажі

Гравець може обрати одного з 4 персонажів (див. рис. 3.6, 3.7, 3.8, 3.9). Кожен персонаж був намальований, кожен з них має унікальний зовнішній вигляд. Це дозволяє гравцям обирати героя, який найбільше відповідає їхнім вподобанням та стилю гри. Детальне промальовування кожного героя створює відчуття індивідуальності та унікальності. Всі персонажі мають набір зображень для створення анімацій руху, покою, стрибка, атаки та нокауту.

Анімації кожного персонажа ретельно розроблені для того, щоб зробити гру більш захоплюючою та реалістичною. Від плавного руху під час ходьби та бігу до динамічних стрибків і атак, кожен рух персонажа виглядає природньо і привабливо. Особлива увага приділяється деталям.

Гравці можуть насолоджуватися високоякісними текстурами, плавними анімаціями та живими кольорами, що підкреслюють індивідуальність кожного героя. Це робить гру не тільки цікавою з точки зору геймплею, але й приємною для ока, що значно підвищує загальний досвід від гри.

Рисунок 3.6 – Набір зображень для створення анімація для першого персонажу

Рисунок 3.7 – Набір зображень для створення анімація для другого персонажу

Рисунок 3.8 – Набір зображень для створення анімація для третього персонажу

Рисунок 3.9 – Набір зображень для створення анімація для четвертого персонажу

Кожен з персонажів у грі має унікальну зброю. Різноманіття зброї також підкреслює індивідуальність кожного персонажа, створюючи унікальний образ і відчуття героїзму. Гравці можуть експериментувати з різними типами зброї для знаходження оптимального підходу до досягнення своїх цілей у грі. Це створює стратегічні варіанти для гравців, які можуть вибирати персонажів залежно від своїх уподобань та стилю гри.

3.2.3Користувацький інтерфейс

Для гри були розроблені такі схематичні макети сцен, щоб забезпечити інтуїтивно зрозумілий та зручний інтерфейс для гравців. Кожен макет ретельно спроектований з урахуванням найкращих практик UX/UI дизайну.

Головне меню є першою сценою, яку бачить гравець при запуску гри. Воно включає логотип гри, розташований у центрі екрана, і яскраву фонова картинку, що створює атмосферу гри (див. рис. 3.10). На екрані також розміщені кнопки «Старт», «Керування». Кожна кнопка інтуїтивно зрозуміла та легко доступна, забезпечуючи зручну навігацію по меню.

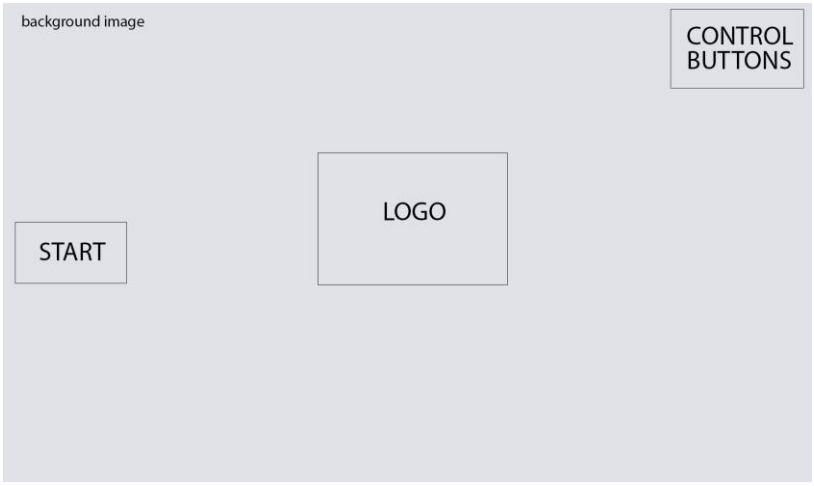


Рисунок 3.10 – Макет головного меню

На сцені вибору героя перший гравець обирає свого персонажа. Інтерфейс містить зображення доступних персонажів (див. рис. 3.11).

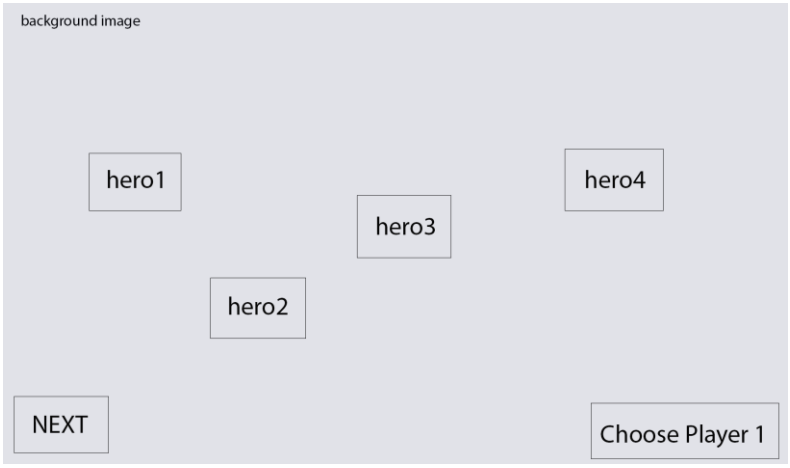


Рисунок 3.11– Макет сцени вибору персонажа для першого гравця

Макет сцени для вибору персонажа для другого гравця аналогічний до сцени вибору персонажа для першого гравця, але призначений для другого гравця (див. рис. 3.12). Він дозволяє другому гравцю обрати свого персонажа із списку доступних. Це дозволяє обом гравцям зробити свій вибір незалежно один від одного.

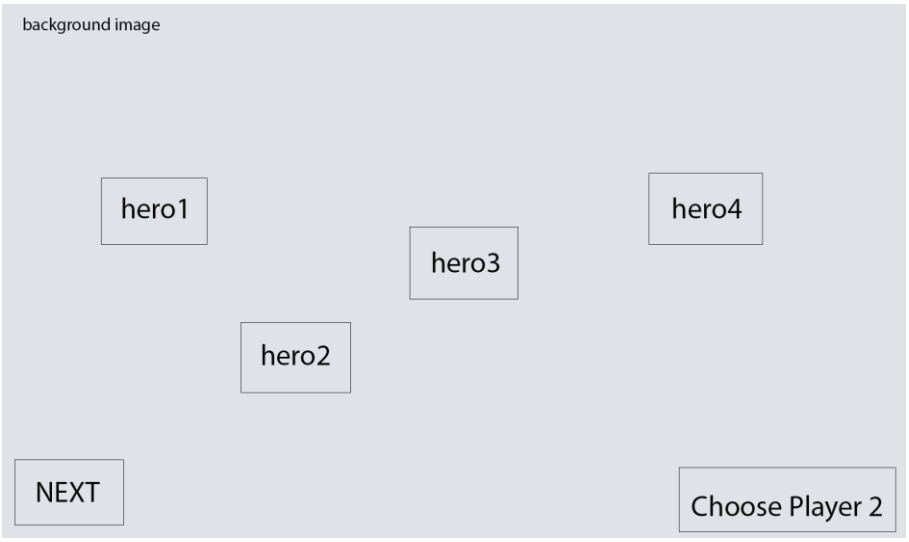


Рисунок 3.12 – Макет сцени для вибору персонажа для другого гравця

Після вибору персонажів гравці переходять до вибору арени для битви. Інтерфейс містить зображення доступних арен, кожна з яких має унікальний дизайн та характеристики. Гравці можуть переглядати і вибирати арену, на якій вони хочуть битися. Вибір арени також супроводжується коротким описом або прев'ю, що допомагає гравцям зробити обґрунтований вибір (див. рис. 3.13).

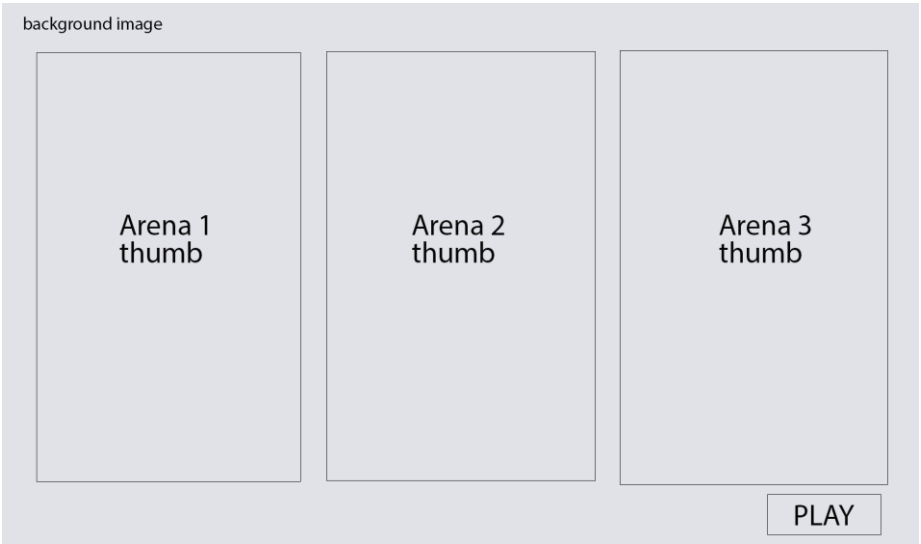


Рисунок 3.13 – Макет сцени для вибору арени для гри

Макет сцени бою - це основна сцена гри, де відбуваються битви між гравцями. Інтерфейс включає шкалу здоров'я для кожного гравця, розташовану у верхній частині екрана, а також таймер раунду. На екрані також відображаються анімації атак, рухів та спеціальних прийомів. Після кожного раунду показується зірочка під шкалою здоров'я переможця, яка відображає кількість виграних раундів (див. рис. 3.14).

Кожен з цих макетів був ретельно розроблений, щоб забезпечити комфортне користування та задоволення від гри. Вони допомагають гравцям легко орієнтуватися в грі, робити вибір і насолоджуватися процесом бою, створюючи повне і захоплююче ігрове середовище.

Макет сцени бою - це основна сцена гри, де відбуваються битви між гравцями. Інтерфейс включає шкалу здоров'я для кожного гравця, розташовану у верхній частині екрана, а також таймер раунду. На екрані також відображаються анімації атак, рухів та спеціальних прийомів.

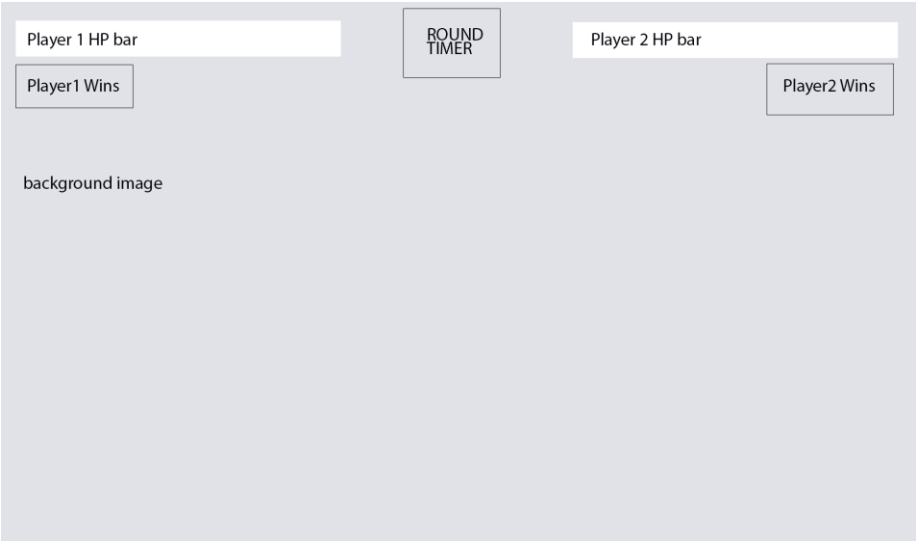


Рисунок 3.14 – Макет сцени бою

Кожен з цих макетів був ретельно розроблений, щоб забезпечити комфортне користування та задоволення від гри.

Вони допомагають гравцям легко орієнтуватися в грі, робити вибір і насолоджуватися процесом бою, створюючи повне і захоплююче ігрове середовище.

3.3 Створення UI/UX

Створюючи головне меню гри, я прагнув передати атмосферу та побудувати зручний, зрозумілий інтерфейс. Розробка головного меню для гри - важлива частина процесу розробки, яка формує перші враження гравця та забезпечує зручний доступ до ключових функцій. Головне меню гри – лаконічне та зрозуміле.

Гра має назву Fatal Ring. Логотип з назвою стилізован в жанрі аркадних ігор, є яскравим, розташований в центрі екрана. Весь концепт гри будується на боротьбі на мечах, що візуалізовано на фоні головного меню у вигляді меча, встромленого в камень. Для створення динаміки на екрані головного меню додані анімовані ефекти палаючого вогню та додана звукова доріжка з музичним файлом. На екрані головного меню є кнопка «Старт» та «Керування» (див. рис. 3.15).

Кнопки "Старт" та "Керування" роблять основні функції гри доступними з першого погляду, забезпечуючи зручний та швидкий доступ до початку гри або перегляду інструкцій. Такий підхід дозволяє гравцеві швидко зануритися в гру та почати свою пригоду без зайвих перешкод.

Рисунок 3.15 – Головне меню гри

У процесі розробки фону для сцени аркадної файтінгової гри, я експериментував з різними варіантами, намагаючись створити вражаючий та емоційно насичений вигляд. В рамках дипломної роботи я розрабив 3 макети ігрової арени для гри.

Перший макет представляє собою ігрову арену, розміщену серед високих гірських піків. Ця сцена поєднує в собі східну тематику та елементи фентезі, створюючи захопливий та загадковий світ (див. рис. 3.16).

На сцену додані архітектурні елементи східного стилю, такі як колони, павільйони та сходи. Вони гармонійно вписуються у гірський пейзаж, створюючи контраст між природою та людськими спорудами. Ця суміш культур надає сцені своєрідної загадковості та інтриги.

Майданчик для бою, облаштований на кам'яній платформі. Ця деталь підкреслює, що ця арена призначена для епічних поєдинків між воїнами. Цей макет ігрової арени демонструє вправне поєднання різних елементів, таких як природа, архітектура, фентезі та східна тематика.

Рисунок 3.16 – Макет фону ігрової арени

Другий макет ігрової арени представляє собою епічний пейзаж з похмурими гірськими утвореннями та руїнами стародавньої цивілізації. Атмосфера наповнена напруженістю та таємничістю, створюючи чудове тло для епічних битв у фентезійному світі (див. рис. 3.17).

Центральним елементом сцени є величезна тріщина у землі, наповнена туманов, що створює ефект занурення у світ фентезі. Цей ефект посилюється завдяки анімованим язикам полум'я, які танцюють на поверхні землі, випромінюючи тепло та світло. Звук палаючого вогню додає ще більшої реалістичності та занурює гравця у цей загрозливий світ.

Навколо лавового розлому розкидані решткі колишньої цивілізації у вигляді зруйнованих колон, сходів та арок. Клубки туману, що огортають ці уламки, додають місцю відчуття таємничості та загадковості. Язики полум’я реалізовані у вигляді префабів.

Клубки туману, що огортають уламки руїн, додають таємничості і загадковості цьому місцю. Туман, що легко ллється навколо занурених у туман об'єктів, створює відчуття невизначеності та стимулює ігроків до дослідження і творчого підходу до гри.

Рисунок 3.17 – Макет фону ігрової арени

Макет для третьої ігрової арени демонструє захоплюючу атмосферу фентезійного світу, об'єднуючи елементи природи та стародавньої архітектури. Сцена розгортається посеред величезних заледенілих гірських піків, що здіймаються у хмари, створюючи епічне та грандіозне тло (див. рис. 3.18).

Центральним елементом є велика кругла арена для боїв, виконана з темного кам'яного матеріалу. Її облямовують два монументальні портали з колонами, прикрашеними орнаментами в стилі стародавніх цивілізацій. Ці архітектурні деталі не лише надають арені відчуття священності, а й створюють враження, що вона є частиною більш масштабного комплексу споруд.

Ці елементи не лише створюють візуальний і аудіальний контекст, а й можуть впливати на хід гри, роблячи кожен бій унікальним та непередбачуваним.

Поєднання суворої природи, древньої архітектури та загадкового світіння створює унікальну атмосферу, яка обіцяє захоплюючі та непередбачувані поєдинки на цій арені.

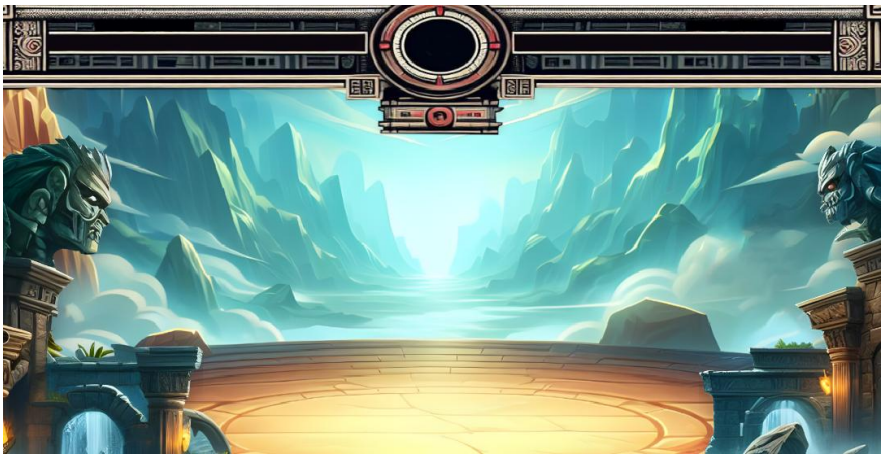


Рисунок 3.18 – Макет фону ігрової арени

Вибір персонажів для бою відбувається послідовно, спочатку перший гравець обирає персонажа для бою. При натисканні на персонажа, за ним включається анімація, яка ідентифікує вибір користувача. Після свого вибору перший гравець настискає на кнопку «Select». Інформація про вибір користувача зберігається у системі (див. рис. 3.19).

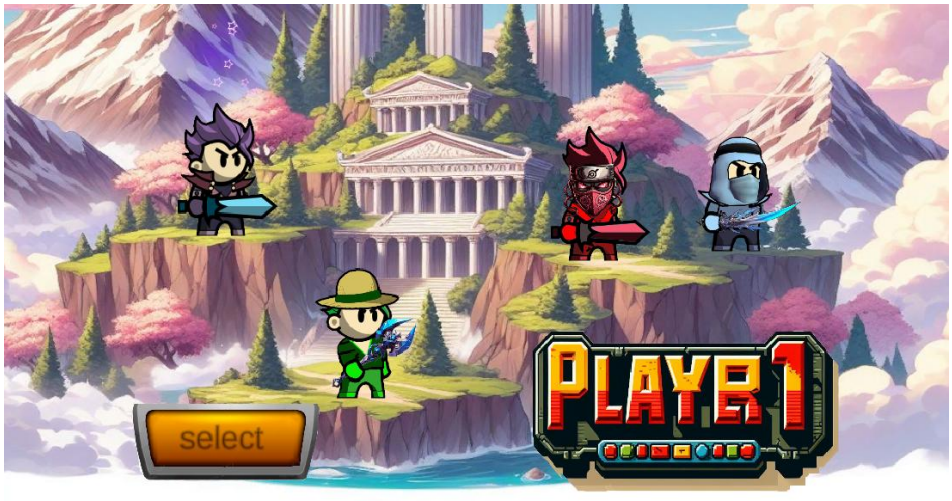


Рисунок 3.19 – Вибір персонажа першим гравцем

Для другого гравця персонаж, який був вибраний першим гравцем стає недоступним для вибору. Він відображається на сцені з ефектом напівпрозорості, що візуально ідентифікує його як недоступного для вибору

другим гравцем. Вибраний другим гравцем персонаж також ідентифікується за допомогою анімованого ефекту (див. рис. 3.20).

Рисунок 3.20 – Вибір персонажа другим гравцем

В рамках даного дипломного проекту було розроблено 3 арени для бою. Для можливості доповнення гри новими аренами, передбачени навігаційні стрілки, які вказують на можливість додавання додаткових локацій у майбутньому. Це відкриває перспективи для подальшого розширення контенту гри та забезпечення її довго строковості. Назви арен - "Lunar Peak", "Forgotten Citadel" та "Glacial" - натякають на їхні відмінні настрої та середовища, які можуть варіюватися від місячних краєвидів і стародавніх руїн до крижаних льодовиків. Такий різноманітний вибір дозволяє задовольнити різні уподобання гравців та забезпечити свіжий досвід під час кожної битви (див. рис. 3.21).

Рисунок 3.21 – Сцена вибору арени

Був розроблений інтерфейс керування для гри. На екрані відображаються дві панелі з призначеними клавішами для керування двома гравцями - Player 1 і Player 2 (див. рис. 3.22).

Рисунок 3.22 - Інтерфейс керування для гри

Ліва панель містить список дій та відповідних клавіш для першого гравця. Права панель містить список дій та відповідних клавіш для другого гравця. Ці панелі дозволяють гравцям легко перевірити та пригадати, які клавіші відповідають за певні дії під час гри. Це особливо корисно для ігор з інтенсивним геймплеєм, де швидке реагування та запам'ятовування комбінацій клавіш має вирішальне значення. Внизу екрану розташована велика кнопка "BACK", яка, дозволяє гравцям повернутися до головного меню.

Кольорова схема інтерфейсу відіграє важливу роль у його функціональності та естетиці. Фіолетовий фон створює контраст з білим текстом, що полегшує читання інформації. Клавіші атаки виділені яскравим помаранчевим кольором для обох гравців, підкреслюючи їх важливість у грі. Це кольорове кодування допомагає гравцям швидко ідентифікувати ключові елементи керування.

Розташування дій у списку також має логічну структуру. Базові рухи (вліво, вправо, стрибок) розміщені на початку списку, за ними йдуть більш складні дії, такі як регенерація здоров'я та спеціальні атаки. Це полегшує запам'ятовування та доступ до різних команд під час гри, особливо для нових гравців, які поступово освоюють механіку гри.

4 ОПИС ПРИЙНЯТИХ ПРОГРАМНИХ РІШЕНЬ

4.1 Реалізація перевірки знаходження персонажа на поверхні землі

У грі реалізовано механізм перевірки контакту персонажа із землею. Це важливо для правильної роботи механік стрибка, анімацій та інших аспектів геймплею.

Основний об'єкт героя (heroClone) має дочірній об'єкт groundCheck. Цей об’єкт розташований в районі ніг персонажа. Це зроблено для того, щоб відслідковувати позицію персонажа відносно поверхні землі (див. рис. 4.1).

Рисунок 4.1 – Розташування об’єкта groundCheck відносно героя

У скрипті HeroBehaviour визначено відповідне поле GroundCheck типу Transform. Наведемо приклад коду визначення поля GroundCheck:

```
public Transform GroundCheck;
```

В інспекторі Unity поле GroundCheck зв'язане з об'єктом groundCheck, що дозволяє скрипту отримати доступ до його позиції (див. рис. 4.2).

Перевірка здійснюється методом CheckGround(). Він використовує функцію Physics2D.OverlapCircle(), яка створює уявне коло з центром у позиції groundCheck. Радіус цього кола визначається змінною checkRadius (встановлено на 0.5 одиниць). Функція перевіряє, чи перетинається це коло з будь-яким

колайдером на шарі, визначеному як "Ground". Наведемо приклад реалізації метода CheckGround():

```
public void CheckGround()
{
onGround = Physics2D.OverlapCircle(GroundCheck.position, checkRadius,
Ground);
}
```

Результат перевірки зберігається у булевій змінній onGround. Якщо виявлено перетин з землею, onGround стає true, інакше - false. Для актуальності даних метод CheckGround() потрібно викликати регулярно, наприклад, у методі Update().

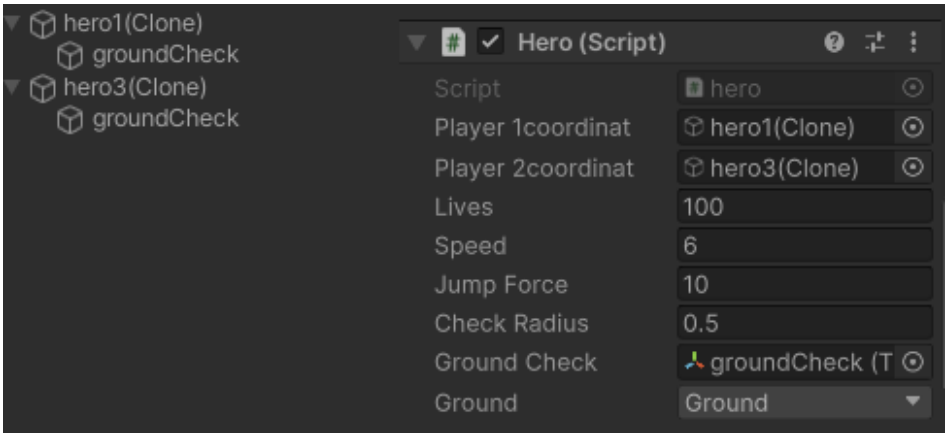


Рисунок 4.2 - Налаштування параметрів скрипта Hero в інспекторі Unity

Цей підхід дозволяє точно визначати, чи стоїть персонаж на землі, що критично важливо для реалізації різних ігрових механік. Розробник може використовувати значення onGround для керування логікою гри, наприклад, дозволяючи стрибки лише коли персонаж на землі або змінюючи анімації залежно від стану контакту з поверхнею.

4.2 Реалізація спауну персонажів на сцені

Функція вибору та спауну персонажів реалізована через скрипти ChoosePlayer та SpawnPlayer. В інспекторі створені дві окремі папки player1 та player2, в яких зберігаються префаби персонажів (див. рис. 4.3).

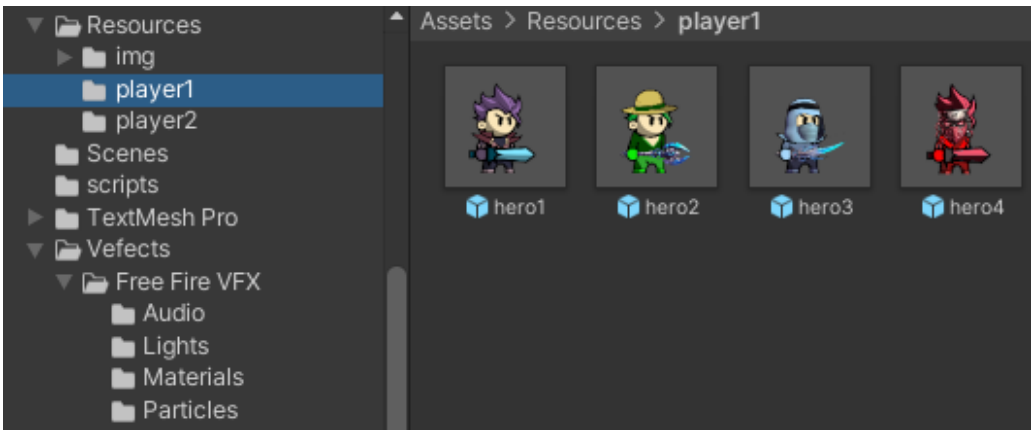


Рисунок 4.3 – Папка для збереження префабів персонажів

Скрипт ChoosePlayer керує вибором персонажів для обох гравців. Він використовує PlayerPrefs для збереження вибору гравців. Наведемо приклад реалізації збереження вибору гравця:

```
csharpCopyPlayerPrefs.SetString("choosed_player", Choosed_Player);
PlayerPrefs.SetString("choosed_player2", Choosed_Player);
```

Скрипт має окремі методи для вибору першого та другого гравця. Наведемо приклад реалізації вибору персонажів:

```
csharpCopypublic void make_choose_player1(string Choosed_Player)
public void make_choose_player2(string Choosed_Player)
```

Після вибору, скрипт оновлює візуальне відображення вибраних персонажів. Наведемо приклад застосування скіна для персонажа:

```
csharpCopyprivate void repaintSkinsP1()
private void repaintSkinsP2()
```

Скрипт SpawnPlayer відповідає за створення персонажів на ігровій сцені. У методі Start() відбувається instantiate (створення) обраних персонажів. Це гарантує, що обидва гравці починають гру з відповідних позицій, що сприяє чесному та збалансованому старту бою. вони розміщуються на попередньо визначених точках спауну на арені, заданих об'єктами playerSpawnPoint та player2SpawnPoint (див. рис. 4.4).

Наведемо приклад реалізації спауну пермонажів на арені:

```
player1Spawned = Instantiate(Resources.Load<Hero>("player1/" +
PlayerPrefs.GetString ("choosed_player", "hero1")),
playerSpawnPoint.transform.position, Quaternion.identity);
player2Spawned = Instantiate(Resources.Load<Hero2>("player2/" +
PlayerPrefs.GetString("choosed_player2", "hero2")),
player2SpawnPoint.transform.position, Quaternion.identity);
```

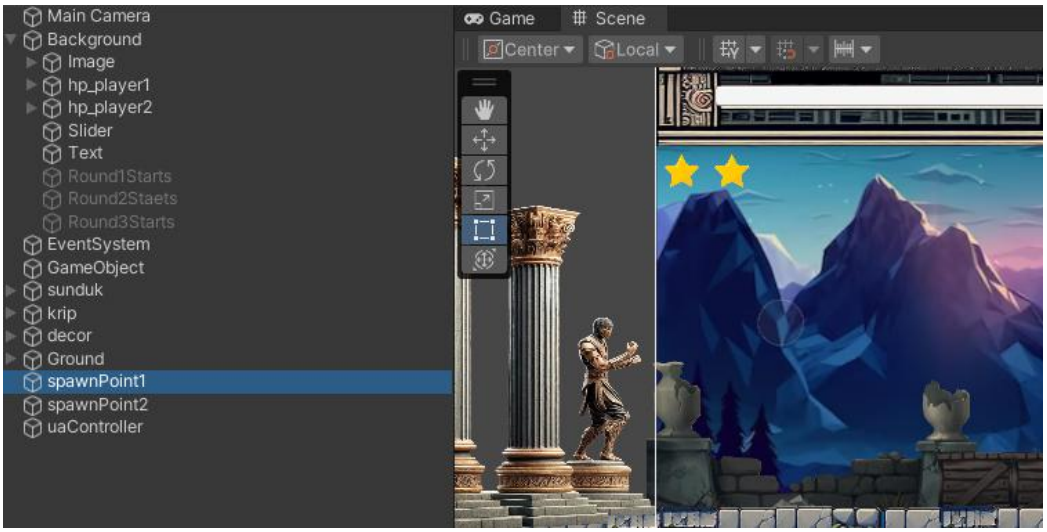


Рисунок 4.4 – Точка спауна для персонажів на арені

Персонажі створюються в заздалегідь визначених точках спавну playerSpawnPoint та player2SpawnPoint. Це гарантує, що персонажі будуть з’являтися в одній і тій самій точці під час початку кожного раунду.

4.3 Реалізація повернення персонажів в початкові точки після закінчення раунду

Після закінчення раунду, один з гравців отримує перемогу або між гравцями відбувається ничія, то ж їм необхідно повернутися в початкові точки для початку нового раунду.

Метод ResetPlayerPositions() дозволяє повернути персонажів на початкові позиції після закінчення раунду. Коли раунд завершений, метод whoWinRound() визначає переможця і викликає метод finishRound() для підготовки до нового раунду. Під час цього процесу відбувається виклик методу ResetPlayerPositions(),

який повертає об'єкти player1Spawned та player2Spawned до їхніх початкових координат, встановлених у playerSpawnPoint і player2SpawnPoint. Ця функціональність забезпечує справедливий старт кожного нового раунду, гарантуючи, що персонажі завжди розпочинають з однакових позицій, незалежно від того, де вони знаходилися наприкінці попереднього раунду. Наведемо приклад реалізації повернення персонажів в початкові точки:

```
    public void ResetPlayerPositions()
    {
player1Spawned.transform.position = playerSpawnPoint.transform.position;
player2Spawned.transform.position = player2SpawnPoint.transform.position;
    }
```

4.4 Реалізація розміщення камери на сцені гри

Скрипт CameraControl керує поведінкою камери у грі, зокрема її позиціонуванням та масштабуванням відносно двох гравців. Знаходяться координати точок, в яких в даний момент часу знаходяться персонажі. Камера розпалагається завжди між ними (див. рис. 4.5). Під час руху персонажів, скрипт вираховує відстань між гравцями та розміщує камеру строго посеред ними.

Рисунок 4.5 – Розміщення камери на сцені гри

Наведемо приклад реалізації розташування камери:

```
Vector3 centerPoint = new Vector3(((player1.transform.position.x
+ player2.transform.position.x) / 2, 0, 0);
transform.position = centerPoint + offset;
```

Камера автоматично знаходить гравців за тегами, що дозволяє динамічно підключатися до них. Камера завжди центрується між двома гравцями, що добре підходить для ігор на двох.Використання LateUpdate() забезпечує плавний рух камери.

4.5 Реалізація логіки бою

Бій складається з трьох раундів. Кожен раунд триває до 90 секунд. Перемагає у бою той гравець, хто переміг у двох раундах. Логіка бою з трьох раундів реалізована за допомогою класу roundControl. Цей клас керує процесом бою, включаючи відлік часу, визначення переможця кожного раунду, та ініціалізацію нового раунду (див. рис. 4.6).

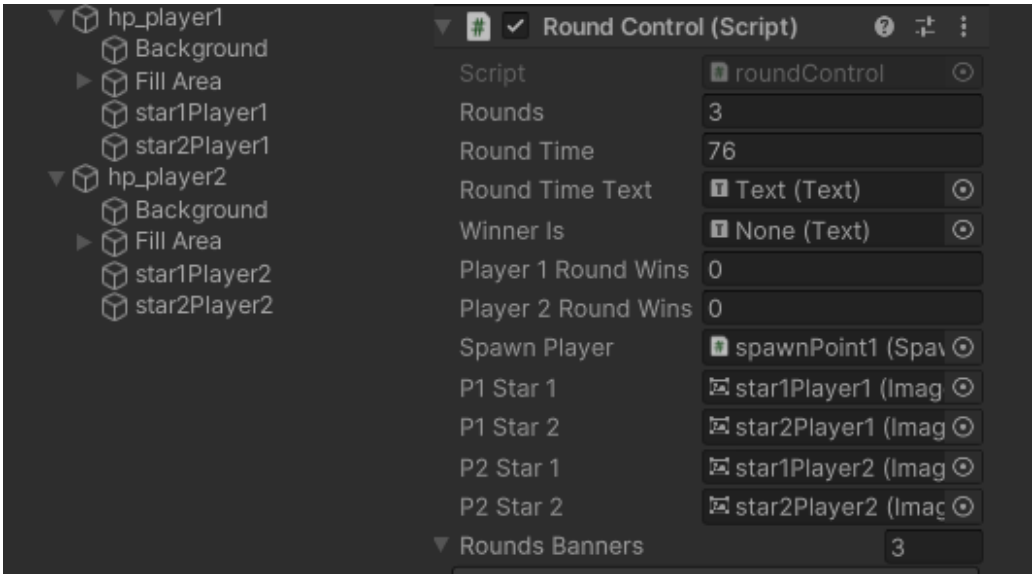


Рисунок 4.6 – Метод Round Control в editor Unity

На початку гри, у методі Start(), викликаються кілька функцій для підготовки до бою. Спочатку за допомогою методу Invoke("ident", waitTime) встановлюються об'єкти гравців та їхні початкові кількості життів.

Встановлюється прозорість зірок, що відображають кількість виграних раундів, щоб їх було видно лише при виграші раундів.

Наведемо приклад реалізації ініціалізації зірок, які відповідають кількості виграних раундів:

```
    void Start()
    {
    roundsBanners[2].gameObject.SetActive(true);
    Invoke("ident", waitTime);
    InvokeRepeating("DecreaseRoundTime", 1f, 1f);
    starColor = p1Star1.color;
p1Star1.color = new Color32(starColor.r, starColor.g, starColor.b, 20);
p1Star2.color = new Color32(starColor.r, starColor.g, starColor.b, 20);
p2Star1.color = new Color32(starColor.r, starColor.g, starColor.b, 20);
p2Star2.color = new Color32(starColor.r, starColor.g, starColor.b, 20);
    }
```

Метод InvokeRepeating("DecreaseRoundTime", 1f, 1f) запускає зворотний відлік часу раунду, який оновлюється щосекунди. Значення roundTime кожну секунду оновлюється у вигляді тексту на екрані (див. рис. 4.6). Наведемо приклад реалізації таймера:

```
    public void DecreaseRoundTime()
    {
    if (roundTime > 0)
       {
       roundTime--;
       roundTimeText.text = roundTime.ToString();
       if (roundTime <= 0
          {
          CheckRoundOutcome();
          }
       }
    }
```

Рисунок 4.6 – Реалізація таймеру раунда

Метод CheckRoundOutcome() перевіряє, чи завершився раунд через закінчення часу або через втрату всіх життів одним із гравців. Якщо раунд завершився, встановлюється прапор roundOver і викликається метод whoWinRound(), який визначає переможця раунду. Наведемо приклад реалізації методу перевірки завершення раунду:

```
public void CheckRoundOutcome()
{
if (roundOver) return;
if (hero.lives <= 0 || hero2.lives <= 0 || roundTime <= 0)
{
     roundOver = true;
     whoWinRound();
     if (player1RoundWins >= 2 || player2RoundWins >= 2)
           {
           finishFight();
           }
           }
  }
```

Метод finishRound() зменшує кількість раундів, що залишилися, та викликає StartNewRound() для початку нового раунду після короткої затримки. Наведемо приклад реалізації методу завершення раунду:

```
public void finishRound()
{
    rounds--;
    Invoke("StartNewRound", 1.5f);
}
```

Метод StartNewRound() скидає прапор roundOver, відновлює час раунду до початкового значення та оновлює текстове поле, що відображає час. Він також викликає методи ResetPlayerPositions() та ResetPlayerLives() для повернення персонажів до початкових позицій та відновлення їхніх життів. Також активується відповідний банер для раунду. Наведемо приклад реалізації методу початку нового раунду:

```
public void StartNewRound()
{
    roundOver = false;
    roundTime = 90;
    roundTimeText.text = roundTime.ToString();
    ResetPlayerPositions();
    ResetPlayerLives();
    roundsBanners[rounds-1].gameObject.SetActive(true);
}
```

4.6 Створення набору констант для представлення різних станів персонажа

Використання перерахування станів дозволяє створити набір констант для представлення різних станів персонажа в грі. Це спрощує управління станами та покращує читабельність коду. Перерахування States визначає набір можливих станів, у яких може перебувати персонаж гри.

У даному випадку це стани idle, run, attack, nockout та jump. Перерахування дозволяє легко відстежувати та змінювати стан персонажа під час гри (див. рис. 4.7).

В подальшому у коді викликаються стани героїв, які відповідають анімованим ефектам персонажів.

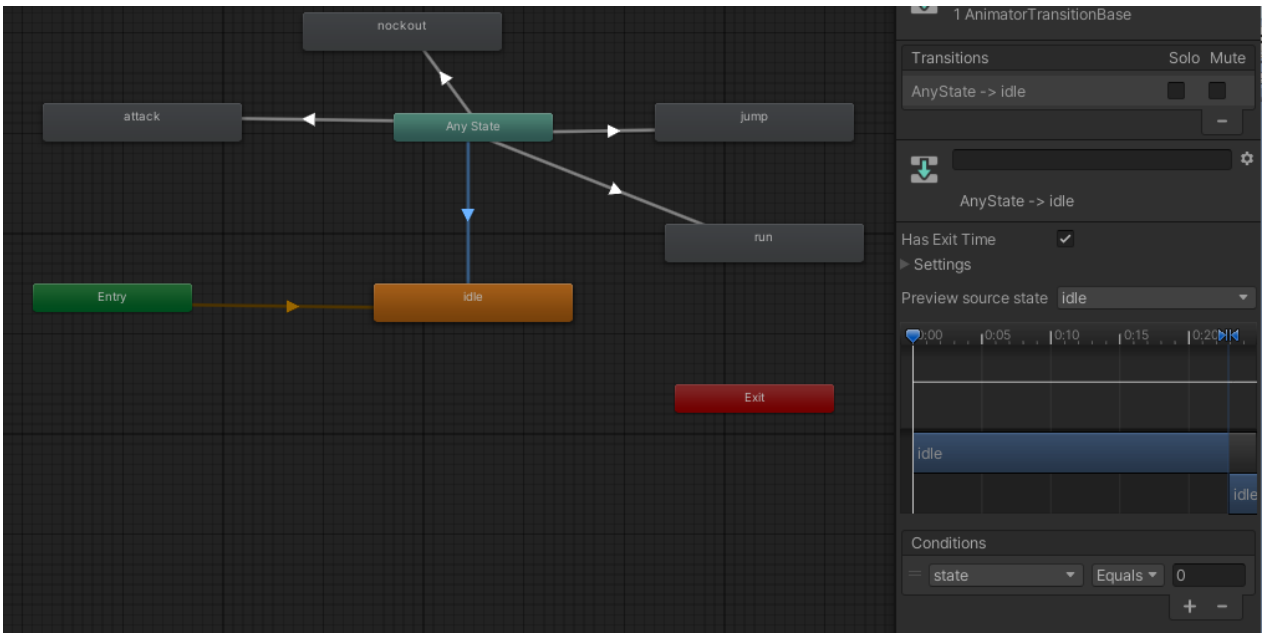


Рисунок 4.7 – Animator в Unity для персонажа

Для реалізації переходів між анімаціями персонажа було створено змінну state, яка відповідає за поточний стан персонажа. Кожному стану присвоєно унікальне числове значення.

Такий підхід дозволяє ефективно керувати переходами між різними анімаціями персонажа. При зміні значення змінної state відбувається відповідна зміна анімації. Наприклад, коли гравець натискає клавішу руху, значення state змінюється на 1, що активує анімацію бігу.

Animator в Unity використовує ці значення state для визначення умов переходу між анімаціями. У вкладці Conditions можна налаштувати умови переходу, базуючись на значенні state. Це забезпечує плавні та логічні переходи між різними станами персонажа, створюючи реалістичну поведінку у грі.

4.7 Реалізація анімованих ефектів для персонажів

У скрипті HeroBehaviour використовується кілька префабів для створення візуальних ефектів у грі. Ці префаби представляють різні анімовані ефекти, які можуть бути викликані під час гри для візуалізації різних дій або станів персонажа (див. рис. 4.8).

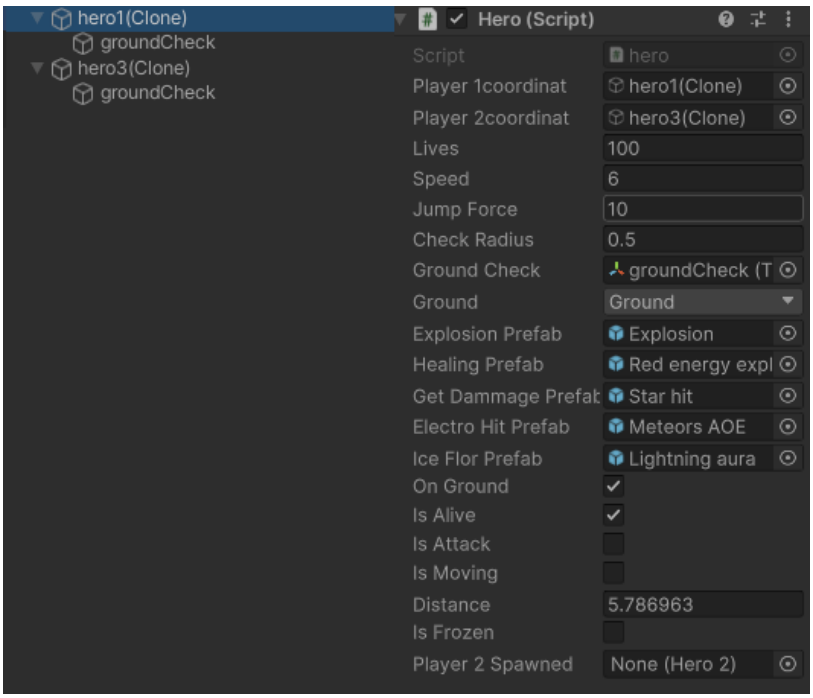


Рисунок 4.8 – Додавання ефектів в editor Unity

Explosion Prefab використовується для створення ефекту вибуху. Він викликається під час спеціальної атаки (див. рис. 4.9). У коді є змінна explosionPrefab, яка посилається на цей префаб. Наведемо приклад реалізації виклику анімованого ефекту від супер удару:

```
player2Spawned.TakeDamage(25);
GameObject explosion = Instantiate(explosionPrefab, player2Object.transform.position, Quaternion.identity);
Destroy(explosion, 1f);
```

Рисунок 4.9 – Реалізація анімованого ефекту від супер удару

Get Dammage Prefab відображає візуальний ефект отримання пошкоджень. Він викликається, коли персонаж отримує удар мечем від ворога. У коді він представлений змінною getDammagePrefab (див. рис. 4.10). Наведемо приклад виклику префабу Get Dammage Prefab:

```
 public void showGetDammageEffect (){
        GameObject player2Object =
GameObject.FindGameObjectWithTag("player2choosed");

        GameObject      getDammage      =      Instantiate(getDammagePrefab,
player2Object.transform.position, Quaternion.identity);
                    Destroy(getDammage, 1f);
    }
```

Рисунок 4.10 – Реалізація анімованого ефекту від удару мечем

Electro Hit Prefab відображає візуальний ефект отримання пошкоджень. Він викликається, коли персонаж отримує удар від ворога. У коді він представлений змінною electroHitPrefab (див. рис. 4.11). Наведемо приклад виклику префабу electroHitPrefab:

```
      Hero2 player2Spawned = player2Object.GetComponent<Hero2>();
     if (player2Spawned != null)
         { if (distance < 5f)
                {
      player2Spawned.TakeDamage(5);
      GameObject electroHit = Instantiate(electroHitPrefab,
player2Object.transform.position, Quaternion.identity);
                    Destroy(electroHit, 1f);
          }}
```

Рисунок 4.11 – Реілазація анімованого ефекту від вогняного дощу

Ice Flor Prefab відображає візуальний ефект отримання пошкоджень. Він викликається, коли персонаж льодяний удар від ворога. У коді він представлений змінною IceFlorPrefab (див. рис. 4.12).

Ця атака наносить 25 урону противнику, заморожує його. Під час заморозки, гравець не може рухатися та атакувати.

Наведемо приклад виклику префабу IceFlorPrefab:

```
Hero2 player2Spawned = player2Object.GetComponent<Hero2>();
        if (player2Spawned != null)
        {
            if (distance < 5f)
            {
                player2Spawned.TakeDamage(25);
                GameObject IceFlorHit = Instantiate(IceFlorPrefab,
player2Object.transform.position, Quaternion.identity);
                Destroy(IceFlorHit, 2f);
                player2Spawned.FreezeHero(2f);
                Debug.Log("Игрок 2 заморожен");
            }
            else
            {
                Debug.Log("Miss!!!");
            }
        }
```

Рисунок 4.12 – Реілазація анімованого ефекту від електро атаки

Healing Prefab відображає візуальний ефект відновлення здоров’я. Він викликається, коли персонаж хоче відновити своє здоров’я. Доступна ця функція один раз за всю гру.

Healing Prefab є важливою складовою гри, забезпечуючи механіку відновлення здоров’я персонажа. Цей префаб активується, коли гравець вирішує скористатися єдиною можливістю в грі для повного відновлення здоров’я свого героя. У коді він представлений змінною healingPrefab (див. рис. 4.13). Наведемо приклад виклику префабу Healing Prefab:

```
public void hilka()
    {
        if(isFrozen)
            return;

        if (ToDoHilka > 0)
        {
            lives = 100;
            OnHpChanged?.Invoke(lives);
            GameObject explosion = Instantiate(healingPrefab,
transform.position, Quaternion.identity);
            Destroy(explosion, 1.5f);
            ToDoHilka--;
        }
    }
```

Рисунок 4.13 – Реалізація анімованого ефектв відновлення здоров’я

Візуальний ефект відновлення не тільки підкреслює важливість цієї дії, але й додає атмосферності ігровому процесу, роблячи момент відновлення здоров’я більш запам'ятовуваним.

У випадку, якщо умова дозволяє використання ефекту, перевіряється ще одна умова – наявність можливості відновлення здоров’я (ToDoHilka > 0). Якщо ця умова виконується, здоров’я персонажа миттєво відновлюється до максимального значення (100 одиниць). Подія OnHpChanged інформує інші частини гри про зміну здоров’я, що може викликати різні додаткові ефекти або оновлення інтерфейсу користувача.

Створення візуального ефекту відбувається за допомогою команди Instantiate, яка розміщує префаб у позиції персонажа (transform.position) з використанням стандартної орієнтації (Quaternion.identity). Це створює вражаючий візуальний ефект відновлення здоров’я, який триває 1.5 секунди. Після цього ефект автоматично видаляється з гри за допомогою команди Destroy.

Гравці повинні ретельно обдумувати, коли і як використовувати цю можливість для досягнення максимального ефекту в критичні моменти гри.

5 ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Для тестування гри використовувалося мануальне тестування та тестування продуктивності. Основними підвидами мануального тестування є як функціональне, так і не функціональне.

5.1 Тестування продуктивності

Тестування продуктивності перевіряло, чи гра працює без фрізів і затримок навіть при високому навантаженні. Це включало перевірку FPS (frames per second), час відгуку на дії гравця та використання пам'яті.

Для тестування продуктивності гри я використовував вбудований інструмент в Unity - Profiler. Це вид тестування Performance Testing, який відноситься до динамічного та інструментального підходу тестування (див. рис. 5.1).

Рисунок 5.1 - Графік продуктивності Unity Profiler

Гра в цілому працює стабільно на рівні 60 FPS, що вказує на хорошу оптимізацію. Спостерігається один значний пік навантаження, де час обробки кадру зростає до приблизно 33 мс (30 FPS). Цей пік пов'язаний із завантаженням ресурсів ігреків на арені. Після піку продуктивність швидко повертається до

нормального рівня, демонструючи ефективне управління ресурсами. Більша частина обчислювальних ресурсів витрачається на рендеринг, що типово для ігрових застосунків. Інші системи (скрипти, фізика, анімація) займають меншу, але помітну частину ресурсів, і їхнє навантаження виглядає збалансованим.

5.1.2 Тестування користувацького інтерфейсу (UI Testing)

Було перевірено, чи всі елементи користувацького інтерфейсу відображаються і працюють правильно. Зокрема, перевірка відображення шкали здоров'я, таймерів, зірок за перемогу в раундах та інших елементів інтерфейсу.

У верхній частині екрану присутня шкала з часом до закінчення раунда посередині. Перевірено, чи коректно відображається ця шкала та чи оновлюється значення в реальному часі. Перевірено, чи коректно відображаються обидва ігрові персонажі на екрані, включаючи їхню анімацію та позиціонування. Протестовано правильність відображення фонового пейзажу з горами та місяцем, а також його взаємодію з іншими елементами інтерфейсу.

В ході виконання даного тесту було виявлено, що UI елемент з декоративною колоною зліва перекриває UI елемент відображення зірок, які ідентифікують перемогу у раунді.

Рисунок 5.2 – Виявлення багу при UI тестуванні

5.1.3 Модульне тестування (Unit Testing)

Модульне тестування проводилося для перевірки окремих функцій і методів. Це дозволило переконатися, що кожна функція працює правильно незалежно від інших частин системи. Зокрема, були протестовані методи класів Hero та Hero2.

В ході виконання данного тестування було виявлеено баг, при якому некорректно нараховувало перемогу гравцю у випадку, коли час раунда закінчувався, я у обох гравців кількість здоров'я булла більша нуля. При цьомуперемога нікому не нараховувалась.

5.1.4 Функціональне тестування (Functional Testing)

Функціональне тестування було спрямоване на перевірку функціональних вимог гри. Було протестовано, чи всі заплановані функції працюють відповідно до вимог. Наприклад, перевірка коректності роботи спеціальних атак, відновлення здоров'я, заморожування персонажів та інше.

Було проведено мануальне тестування, метою тестування було перевірити всі можливі комбінації персонажів, щоб забезпечити коректну роботу гри у всіх сценаріях. Було використано метод вичерпного тестування, де кожен персонаж був протестований у грі проти кожного іншого персонажа. Це забезпечило повне покриття всіх можливих комбінацій. Була розроблена матриця тестування для відстеження всіх комбінацій (див. табл. 8). Для кожного тесту записувались вибрані персонажі, спостереження під час гри, будь-які виявлені проблеми або аномалії.

Для кожної комбінації двох персонажів:

- перший гравець вибирав персонажа A;
- другий гравець вибирав персонажа B;
- запускалась гра;
- спостерігалась взаємодія персонажів;
- фіксувались результати.

Таблиця 5.1 – Матриця тестування для відстеження всіх комбінацій

				
	Не можливо почати бій	Проблема відображення анімації idle	Проблема отримання урону гравцем 2	Проблем не виявлено
	Проблеми не виявлено	Не можливо почати бій	Проблема відображення анімації knockout	Проблема відображення анімації knockout
	Проблеми не виявлено	Проблема отримання урону гравцем	Не можливо почати бій	Проблема отримання урону гравцем
	Проблем не виявлено	Проблем не виявлено	Проблем не виявлено	Не можливо почати бій

При виконанні цього виду тестування були виявлені проблеми з відображенням анімацій станів у деяких персонажів.

Результати тестування показали, що гра працює стабільно і відповідає вимогам. Усі виявлені помилки були виправлені, а повторне тестування підтвердило їх відсутність. Тестування продуктивності показало, що гра працює без значних затримок і лагів. Функціональні тести підтвердили, що всі основні функції гри працюють коректно.

Проведене тестування дозволило забезпечити високу якість розробленої гри і підготувати її до випуску.

Таблиця 5.2 – Тест-кейс №1 (таблиця виконана самостійно)

Інформація про тест-кейс			
Ідентифікатор тесту:	Тест-кейс №1		
Опис функції:	Проходження циклу бою		
Власник тесту:	Блох Денис Сергійович		
Дата створення:	05.06.2024		
Мета тесту:	Перевірка реалізації послідовності проходження бою		
Передумова			
№	Опис випадку	Очікуваний результат	Висновок
1	Ігрок 1 обрав персонажа	Персонаж вибран ігроком	Пройдено
2	Ігрок 2 обрав персонажа	Персонаж вибран ігроком	Пройдено
3	Арена для бою вибрана	Арена вибрана коректно	Пройдено
Послідовність проходження бою			
№	Опис випадку	Очікуваний результат	Висновок
1	Гравці з’являються на арені	Гравці з'являються на своїх початкових позиціях	Пройдено
2	Показується банер 1 раунда	Банер "Раунд 1" відображається на екрані	Пройдено
3	Починається відлік часу	Таймер починає зворотній відлік	Пройдено
4	Перший ігрок атакує	Анімація атаки відтворюється коректно з кожним ударом	Пройдено
5	Життя другого героя стає 0	Шкала здоров'я другого героя зменшується до нуля	Пройдено
6	Перший ігрок отримую зірку	Лічильник перемог першого гравця збільшується на 1	Пройдено
7	Ігроки переносяться в початкові позиції	Гравці повертаються на стартові позиції	Пройдено

8	Показується банер 2 раунду	Банер "Раунд 2" відображається на екрані	Пройдено
9	Починається відлік часу	Таймер починає зворотній відлік	Пройдено
10	Другий ігрок атакує	Анімація атаки відтворюється коректно	Пройдено
11	Життя першого ігрока стає 0	Шкала здоров'я першого героя зменшується до нуля	Пройдено
12	Другий ігрок отримує зірку	Лічильник перемог другого гравця збільшується на 1	Пройдено
13	Ігроки переносяться в початкові позиції	Гравці повертаються на стартові позиції	Пройдено
14	Показується банер 3 раунда	Банер "Раунд 3" відображається на екрані	Пройдено
15	Починається відлік часу	Таймер починає зворотній відлік	Пройдено
16	Перший ігрок атакує	Анімація атаки відтворюється коректно з кожним ударом	Пройдено
17	Життя другого героя стає 0	Шкала здоров'я другого героя зменшується до нуля	Пройдено
18	Перший ігрок отримую другу зірку	Лічильник перемог першого гравця збільшується до 2	Пройдено
19	З’являється повідомлення про перемогу 1 ігрока	Відображається екран з результатами бою та оголошенням переможця	Пройдено

Результати тестування		
Тестувальник: Блох Д.С.	Дата прогону тесту: 05.06.2024	Результат тесту (P/F/B): ПРОЙДЕНО (P)

Даний тест-кейс допоможе протестувати коректність послідовності проведення раундів у грі. При успішному його проходжені ігрок отримує повідомлення про перемогу.

ВИСНОВКИ

Аркадні ігри, здебільшого, визначаються своєю легкістю та веселим геймплеєм, що робить їх привабливими для широкого кола гравців. Серед них можна виявити різноманіття стилів та тематик, що відзначається яскравою графікою та захоплюючими моментами.

Це підтверджує високий рівень конкуренції у сегменті аркадних ігор, адже розробники намагаються створити унікальний ігровий продукт, який привертає увагу гравців та отримує велику кількість завантажень. Цей тренд свідчить не лише про поширену популярність жанру, але і про велику конкуренцію серед розробників, що прагнуть вивести свої ігрові застосунки вперед у ігровій індустрії.

Ігрова індустрія є найбільш швидко зростаючою. Відеоігри вже сьогодні досить сильно впливають на сфери нашого життя. Усе більшу роль у світовій економіці відводять ігровим компаніям, що формують тренди та створюють інновації. Частка ринку невпинно збільшується, а сам ринок стає більш глибоким. У ньому є місце як передовим технологіям віртуальної реальності, так і кіберспортивним змаганням. Необхідно слідкувати за розвитком ігрової індустрії, адже в подальшому вона займатиме ще більшу частку на світовому ринку. На підставі проаналізованих даних можна дійти висновку, що Азіатсько-Тихоокеанський та Північна Америка – це регіони одні з найкращих та найбільш прибуткових ігрових ринків. Прибутки за 2019 р. у цих регіонах становили 72,2 млрд дол. та 36,6 млрд дол. відповідно, що в загалом становить 73% від доходу ігрової індустрії. Україна має потенціал до розвитку в даній галузі (якісні кадри та низька конкуренція). Кожного року в Україні з’являються все нові й нові компанії, що свідчить про зацікавленість у цій галузі.

Ігрова індустрія майже не чуттєва до економічних криз. Вона продовжує свій розвиток навіть під час них. Також цей вид діяльності приносить великі прибутки в країнах, де вона розвинена. Продукція розповсюджується на безліч різних платформ, з яких користувачі отримують до неї доступ.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Сохацька О.М. Біржова справа: підручник. – 2-ге вид. змін. й доп. – Тернопіль: Карт-бланш, 2008. – 632 с.

2. Акметов К. (2013) Взаємодія людини і комп'ютера: тенденції, дослідження, майбутнє. Форсайт, т. 7, № 2. с. 58–68.

3. Сальникова Н.М. (2014) Комп'ютерні ігри як новий вид медіа-бізнесу. Вісник Університету, т. 1, с. 135–138.

4. Брумштейн Ю., Харитонов Д. (2015) Комп'ютерні ігри: синтез творчості і сучасних технологій. Інтелектуальна власність. Авторське право і суміжні права, т. 10, с. 41–53.

5. Hart C.B. (2017) The Evolution and Social Impact of Video Game Economics / Hart. Lanham, Boulder, New York, London: Lexington Books. 163 p.

6. Thorn, A. (2014). Pro Unity Game Development with C#. [Електронний ресурс] – URL: https://link.springer.com/book/10.1007/978-1-4302-6745-4

7. Nicoll, B., Keogh, B. (2019). The Unity Game Engine and the Circuits of Cultural Software [Електронний ресурс] – URL: https://link.springer.com/book/10.1007/978-3-030-25012-6

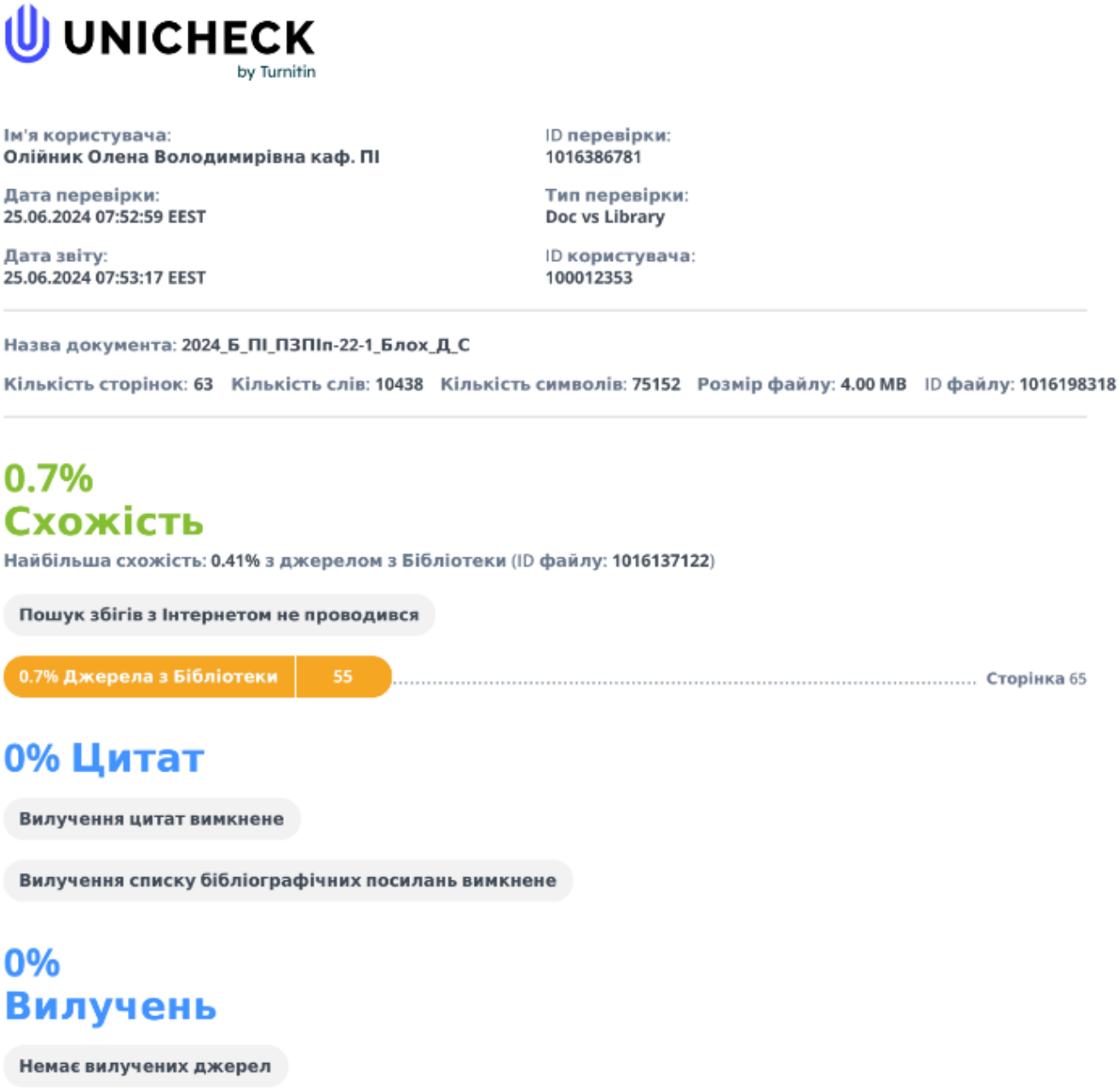
8. Piero, C.M.W., Cabanillas-Carbonell, M. (2023). Video Game in Unity for the Learning Process in the Mathematics Course in Children. DOI: 10.1007/978-981-99-3236-8_66

9. Satria, T.G., Priyanto, R.R., Azzahra, Y.Q. (2023). Casual game design to introduce jamu. DOI: https://doi.org/10.1016/j.protcy.2022.08.249

10. Daineko, Ye. A., Aitmagambetov, A. Z., Tsoy, D. D., Kulakayeva, A. E., & Ipalakova, M. T. (2022). Computer Simulation of a Spectrum Analyzer Based on the Unity Game Engine. DOI: 10.1007/978-3-031-15546-8_8

ДОДАТОК А

Звіт результатів перевірки на унікальність тексту в базі ХНУРЕ



UNICHECK
by Turnitin

Ім'я користувача:
Олійник Олена Володимирівна каф. ПІ

ID перевірки:
1016386781

Дата перевірки:
25.06.2024 07:52:59 EEST

Тип перевірки:
Doc vs Library

Дата звіту:
25.06.2024 07:53:17 EEST

ID користувача:
100012353

Назва документа: 2024_Б_ПІ_ПЗПІп-22-1_Блох_Д_С

Кількість сторінок: 63 Кількість слів: 10438 Кількість символів: 75152 Розмір файлу: 4.00 MB ID файлу: 1016198318

0.7%
Схожість

Найбільша схожість: 0.41% з джерелом з Бібліотеки (ID файлу: 1016137122)

Пошук збігів з Інтернетом не проводився

0.7% Джерела з Бібліотеки 55 ..... Сторінка 65

0% Цитат

Вилучення цитат вимкнене

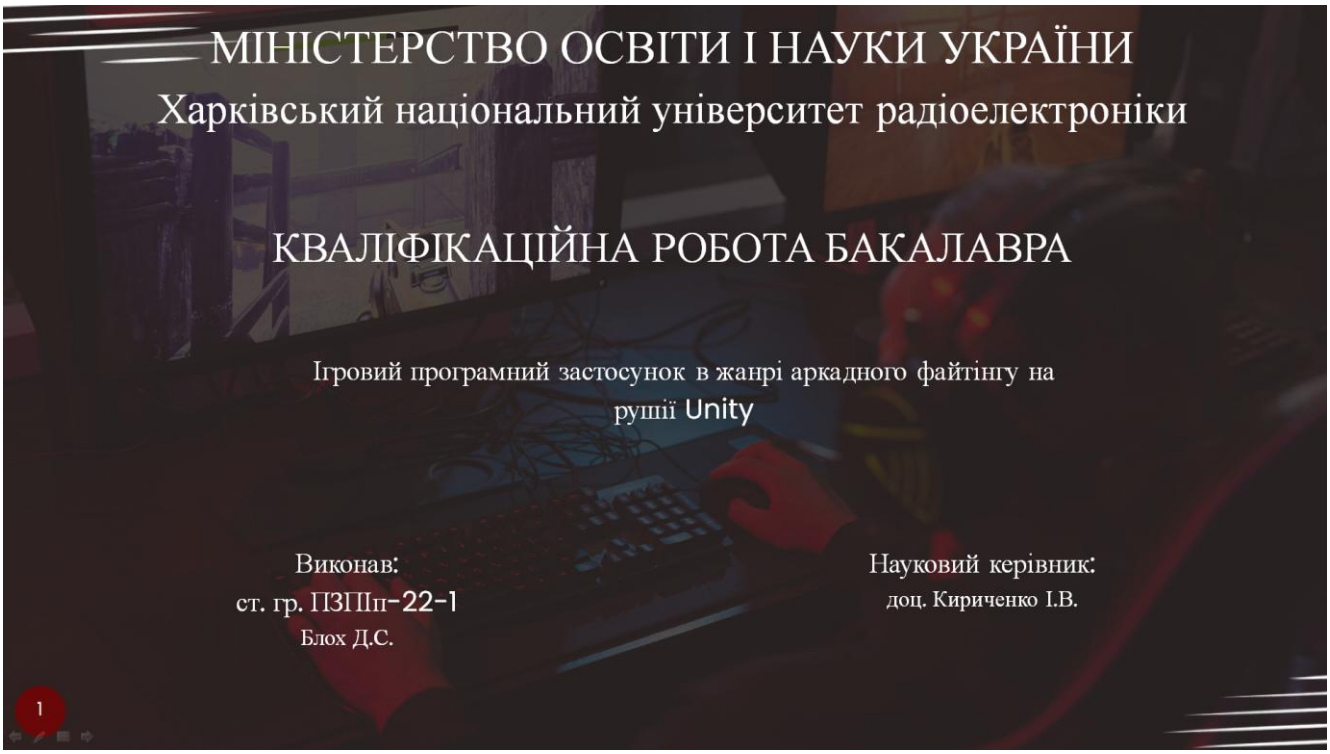
Вилучення списку бібліографічних посилань вимкнене

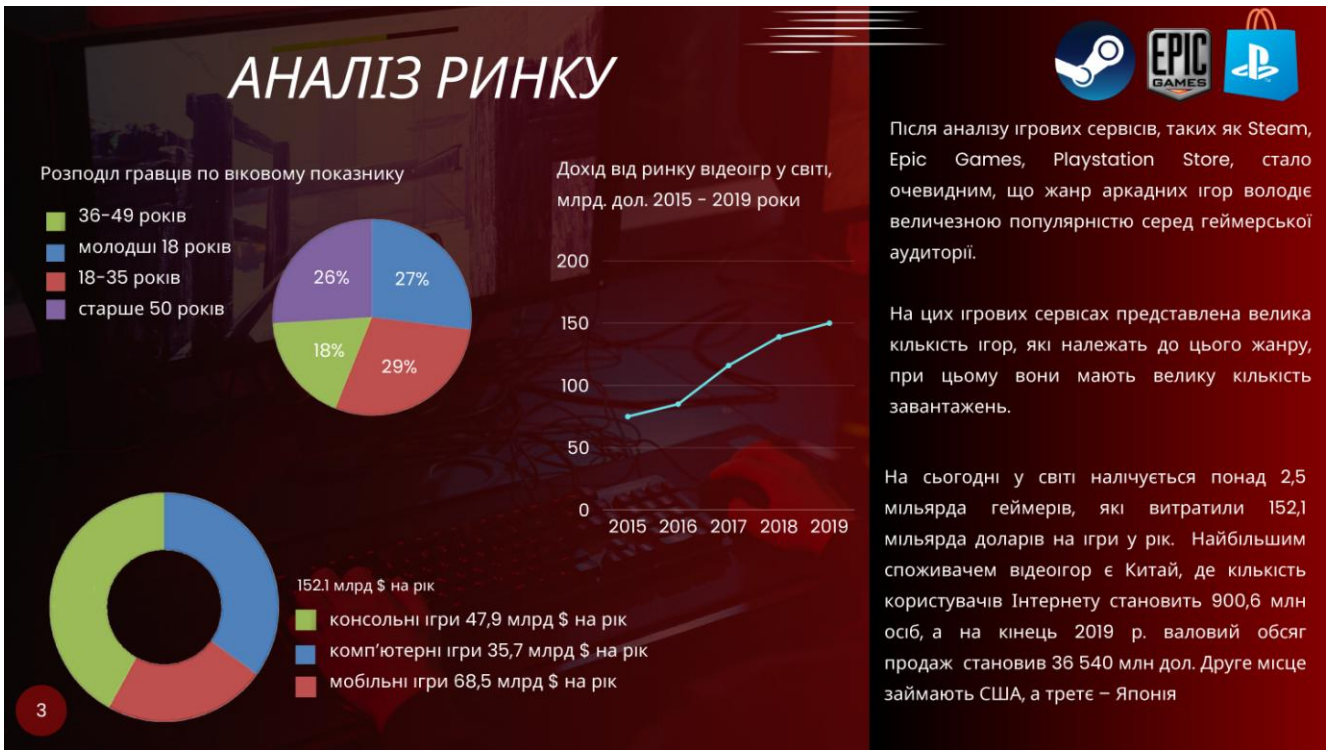
0%
Вилучень

Немає вилучених джерел

ДОДАТОК Б

Слайди презентації




АНАЛІЗ РИНКУ
Розподіл гравців по віковому показнику
36-49 років
молодші 18 років
18-35 років
старше 50 років
26%
27%
18%
29%
Дохід від ринку відеоігр у світі, млрд. дол. 2015 - 2019 роки
200
150
100
50
0
2015 2016 2017 2018 2019
152.1 млрд $ на рік
консольні ігри 47,9 млрд $ на рік
комп'ютерні ігри 35,7 млрд $ на рік
мобільні ігри 68,5 млрд $ на рік
Після аналізу ігрових сервісів, таких як Steam, Epic Games, Playstation Store, стало очевидним, що жанр аркадних ігор володіє величезною популярністю серед геймерської аудиторії.
На цих ігрових сервісах представлена велика кількість ігор, які належать до цього жанру, при цьому вони мають велику кількість завантажень.
На сьогодні у світі налічується понад 2,5 мільярда геймерів, які витратили 152,1 мільярда доларів на ігри у рік. Найбільшим споживачем відеоігор є Китай, де кількість користувачів Інтернету становить 900,6 млн осіб, а на кінець 2019 р. валовий обсяг продаж становив 36 540 млн дол. Друге місце займають США, а третє – Японія
3


ПРО СВІТ ІГОР
Найбільшу популярність серед комп'ютерних ігор в індустрії отримували проекти, які мали свої унікальні фішки, підходи чи ефекти. Наприклад, культова гра Max Payne, яку зробив відомою ефект стрибка у повітрі, що дозволяв гравцям пережити незабутні моменти у стилі "bullet time".
Гра "Корсари" від компанії Акелла, перша реалізувала відкритий світ, що включав як сушу, так і море, створюючи новий рівень занурення у піратську тематику.
Counter-Strike об'єднав мільйони геймерів у мережевій грі, ставши еталоном кіберспортивної дисципліни і встановивши стандарти для багатьох інших шутерів.
Тому надзвичайно важливо, щоб ігровий проект мав свої унікальні елементи, які могли б запам'ятатися гравцям і виділити його серед конкурентів. Це можуть бути інноваційні ігрові механіки, неповторний сюжет, революційна графіка чи незвичайний стиль гри.
4

На початковому етапі розробки було проведено аналіз конкурентів, зокрема відібрано 5 ігор: Street Fighter 6, Dragon Ball Fighterz, Mortal Kombat 1, Street Fighter V та Punch Club. Порівняльний аналіз виявив ключові фактори успіху файтингів на платформі Steam, включаючи графіку, бойові механіки, баланс персонажів та підтримку спільноти. Це дозволило оцінити конкурентоспроможність мого проекту Fatal Ring та визначити напрями для покращення.

5

РОЗРАХУНОК ЕРГОНОМІЧНОСТІ ТА ЯКОСТІ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Для оцінки ергономічності та якості програмного забезпечення з точки зору користувача я сформулюю ряд характеристик програмного продукту: цікавий сюжет, зручний інтерфейс, якісна графіка, система бонусів і нагород у грі, надійність програмного забезпечення.

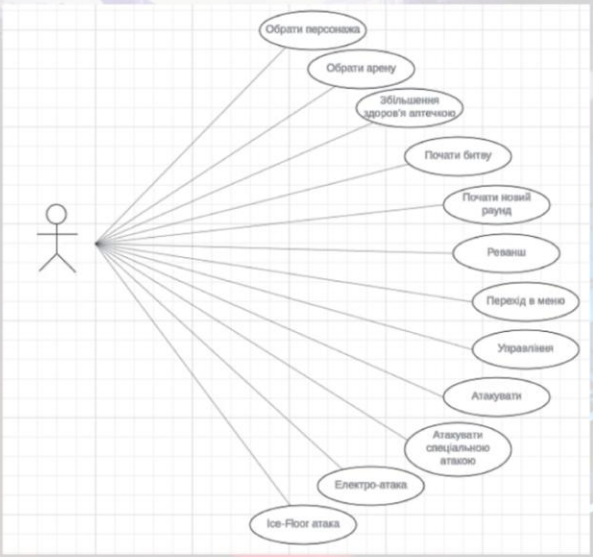
Для проведення оцінки було залучено кілька експертів, які оцінили кожну з вказаних характеристик для різних ігрових проектів. Оцінки проводилися на основі ваг, присвоєних кожній характеристиці, що дозволяє відобразити їх відносну важливість.

	Вага	Fatal Ring	Street Fighter 6	Dragon Ball	Mortal Kombat 1	Street Fighter 5	Punch lub	Fatal Ring	Street Fighter 6	Dragon Ball	Mortal Kombat 1	Street Fighter 5	Punch lub
Експерт 1	0.3	41	38	36	42	40	36	10.6	11.4	10.8	12.6	12	10,8
Експерт 2	0.2	51	46	43	37	49	51	10.2	9.2	8.6	7.4	9.8	10.2
Експерт 3	0.2	48	40	34	40	48	42	9.6	8	6.8	8	9.6	8.4
Експерт 4	0.3	43	38	39	44	44	44	12.9	11.4	11.7	13,2	13,2	13,2
Середня	1							10,8	10	9,5	10.3	11.1	10.6

Проект, який розробляється входить в трійку лідерів по оцінюванню, а саме займає друге місце. Це свідчить про конкурентоспроможність ігрового продукту та його актуальність.

6

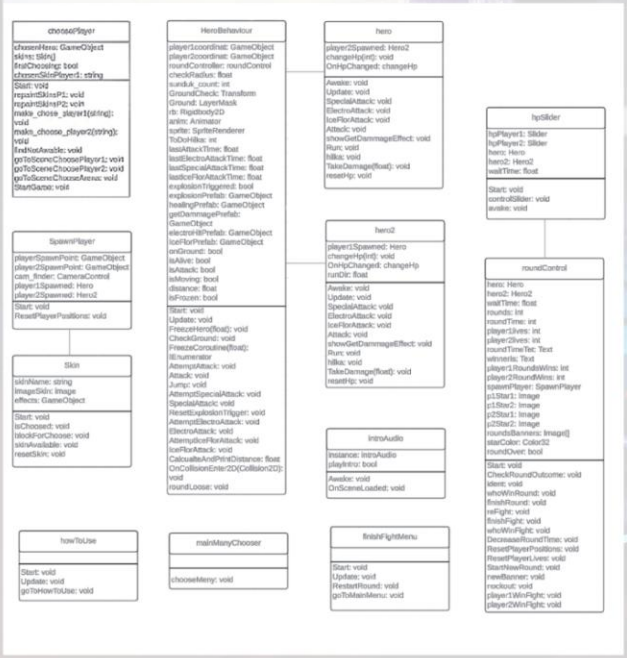
АРХІТЕКТУРА ТА ПРОЕКТУВАННЯ



Перед початком розробки ігрового застосунку були визначені основні функції зі сторони гравця. Користувачами ігрового застосунку є два гравця. При завантаженні гри гравці по черзі обирають собі персонажів, обирають арену для битви та починають бій. Виграє той, хто першим переможе у двох раундах. Кожен раунд триває до 90 секунд.

7

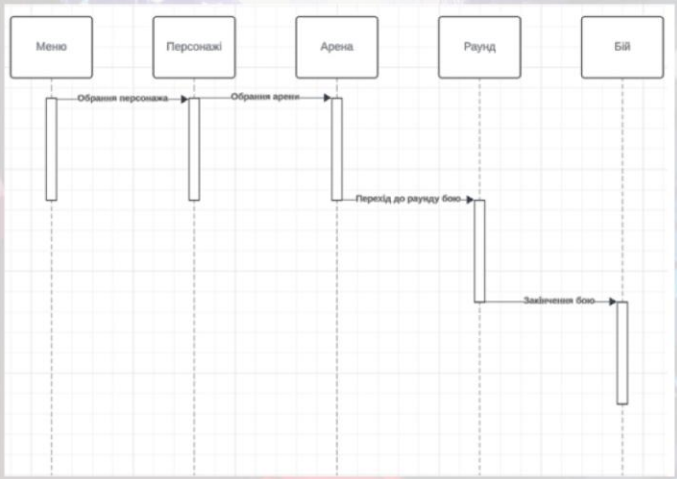
АРХІТЕКТУРА ТА ПРОЕКТУВАННЯ



На слайді зображена діаграма класів. Коротко скажу о головних з них.
Клас HeroBehaviour служить базовим класом для класів Hero і Hero2. Він містить загальні методи і властивості, які використовуються обома типами персонажів.
Клас RoundControl керує логікою організації раундів. Клас CameraControl керуэ камерою. Клас ChoosePlayer відповідає за збереження вибору гравців при завантаженні рівня.

8

АРХІТЕКТУРА ТА ПРОЕКТУВАННЯ



В системі роглядаються 5 основних компонентів:
- меню;
- персонажі;
- арена;
- раунд;
- бій.

Послідовність взаємодій між компонентами, показана вертикальними лініями та стрілками. Діаграма демонструє загальний потік дій у системі, починаючи з вибору персонажа та арени, переходу до бойового раунду і закінчуючи завершенням бою.

9

СТВОРЕННЯ UI/UX

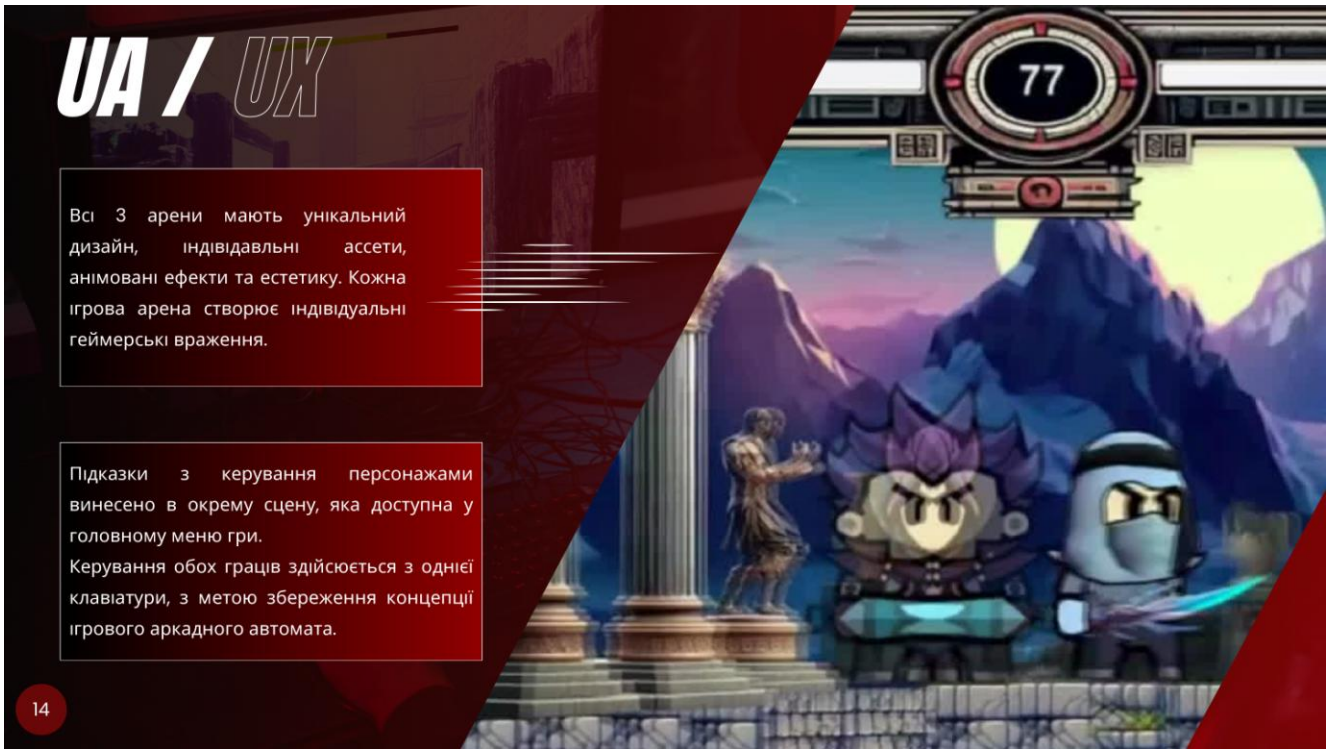
Створюючи головне меню гри, я прагнув передати атмосферу та побудувати зручний, зрозумілий інтерфейс.
Для створення динаміки на екрані головного меню додані анімовані ефекти палаючого вогню та додана звукова доріжка з музичним файлом. На екрані головного меню є кнопка «Старт» та «Керування»

10


ДОСТУПНІ
персонажі
Гравці можуть обирати одного з чотирьох унікальних персонажів, кожен з яких має власні анімації та стилізований зовнішній вигляд.
FATALRING arcade fighting
11


select
PLAYER 1
Вибір персонажів для бою відбувається послідовно, спочатку перший гравець обирає персонажа для бою. При натисканні на персонажа, за ним включається анімація, яка ідентифікує вибір користувача.
Для другого гравця персонаж, який був вибраний першим гравцем стає недоступним для вибору. Він відображається на сцені з ефектом напівпрозорості, що візуально ідентифікує його як недоступного для вибору другим гравцем.
12


choose arena
Lunar Peak
Forgotten Citadel
Було розроблено 3 арени для бою. Для можливості доповнення гри новими аренами, передбачени навігаційні стрілки, які вказують на можливість додавання додаткових локацій у майбутньому. Це відкриває перспективи для подальшого розширення контенту гри та забезпечення її довгостроковості.
13


UA / UX
Всі 3 арени мають унікальний дизайн, індівідавльні ассети, анімовані ефекти та естетику. Кожна ігрова арена створює індівідуальні геймерські враження.
Підказки з керування персонажами винесено в окрему сцену, яка доступна у головному меню гри.
Керування обох грацв здійсюється з однієї клавіатури, з метою збереження концепції ігрового аркадного автомата.
77
14

GROUNDCHECK РЕАЛІЗАЦІЯ

У грі реалізовано механізм перевірки контакту персонажа із землею. Це важливо для правильної роботи механік стрибка, анімацій та інших аспектів геймплею. Основний об'єкт героя (heroClone) має дочірній об'єкт groundCheck. Цей об'єкт розташований в районі ніг персонажа. Це зроблено для того, щоб відслідковувати позицію персонажа відносно поверхні землі.

Перевірка здійснюється методом CheckGround(). Він використовує функцію Physics2D.OverlapCircle(), яка створює уявне коло з центром у позиції groundCheck. Радіус цього кола визначається змінною checkRadius (встановлено на 0.5 одиниць). Функція перевіряє, чи перетинається це коло з будь-яким колайдером на шарі, визначеному як "Ground".

```
PUBLIC VOID CHECKGROUND()
  {
ONGROUND = PHYSICS2D.OVERLAPCIRCLE(GROUNDCHECK.POSITION,
CHECKRADIUS, GROUND);
  }
```

15

SPAWNPLAYER РЕАЛІЗАЦІЯ

Функція вибору та спауну персонажів реалізована через скрипти ChoosePlayer та SpawnPlayer. В інспекторі створені дві окремі папки player1 та player2, в яких зберігаються префаби персонажів. Скрипт ChoosePlayer керує вибором персонажів для обох гравців. Він використовує PlayerPrefs для збереження вибору гравців.

Персонажі створюються в заздалегідь визначених точках спавну playerSpawnPoint та player2SpawnPoint. Це гарантує, що персонажі будуть з'являтися в одній і тій самій точці під час початку кожного раунду. Скрипт ChoosePlayer керує вибором персонажів для обох гравців. Він використовує PlayerPrefs для збереження вибору гравців.

```
PLAYER1SPAWNED = INSTANTIATE(RESOURCES.LOAD<HERO>("PLAYER1/" +
PLAYERPREFS.GETSTRING ("CHOOSED_PLAYER", "HERO1")),
PLAYERSPAWNPOINT.TRANSFORM.POSITION, QUATERNION.IDENTITY);

PLAYER2SPAWNED = INSTANTIATE(RESOURCES.LOAD<HERO2>("PLAYER2/" +
PLAYERPREFS.GETSTRING("CHOOSED_PLAYER2", "HERO2")),
PLAYER2SPAWNPOINT.TRANSFORM.POSITION, QUATERNION.IDENTITY);
```

16

RESETPLAYERPOSITIONS РЕАЛІЗАЦІЯ

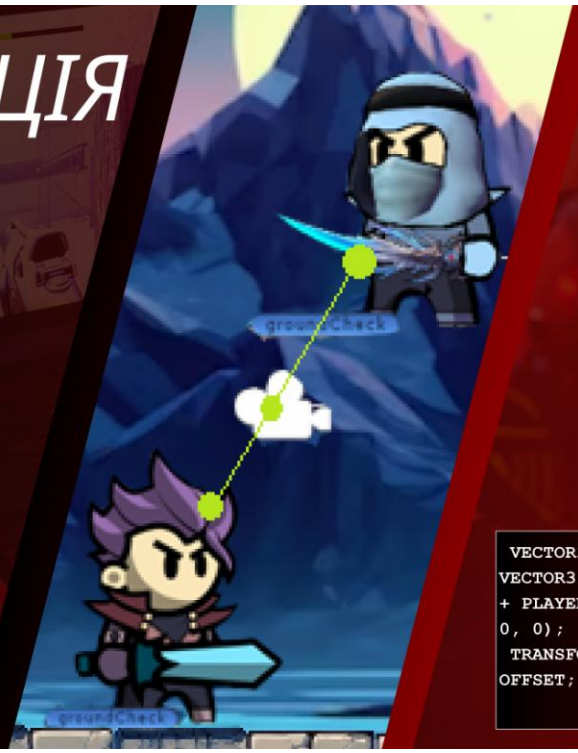
Після закінчення раунду, один з гравців отримує перемогу або між гравцями відбувається ничія, то ж їм необхідно повернутися в початкові точки для початку нового раунду. Метод ResetPlayerPositions() дозволяє повернути персонажів на початкові позиції після закінчення раунду.

Ця функціональність забезпечує справедливий старт кожного нового раунду, гарантуючи, що персонажі завжди розпочинають з однакових позицій, незалежно від того, де вони знаходилися наприкінці попереднього раунду.

```
PUBLIC VOID RESETPLAYERPOSITIONS()
{
PLAYER1SPAWNED.TRANSFORM.POSITION =
PLAYERSPAWNPOINT.TRANSFORM.POSITION;
PLAYER2SPAWNED.TRANSFORM.POSITION =
PLAYER2SPAWNPOINT.TRANSFORM.POSITION;
}
```

17

РЕАЛІЗАЦІЯ CAMERA CONTROL



Камера автоматично знаходить гравців за тегами, що дозволяє динамічно підключатися до них. Камера завжди центрується між двома гравцями, що добре підходить для ігор на двох.Використання LateUpdate() забезпечує плавний рух камери.

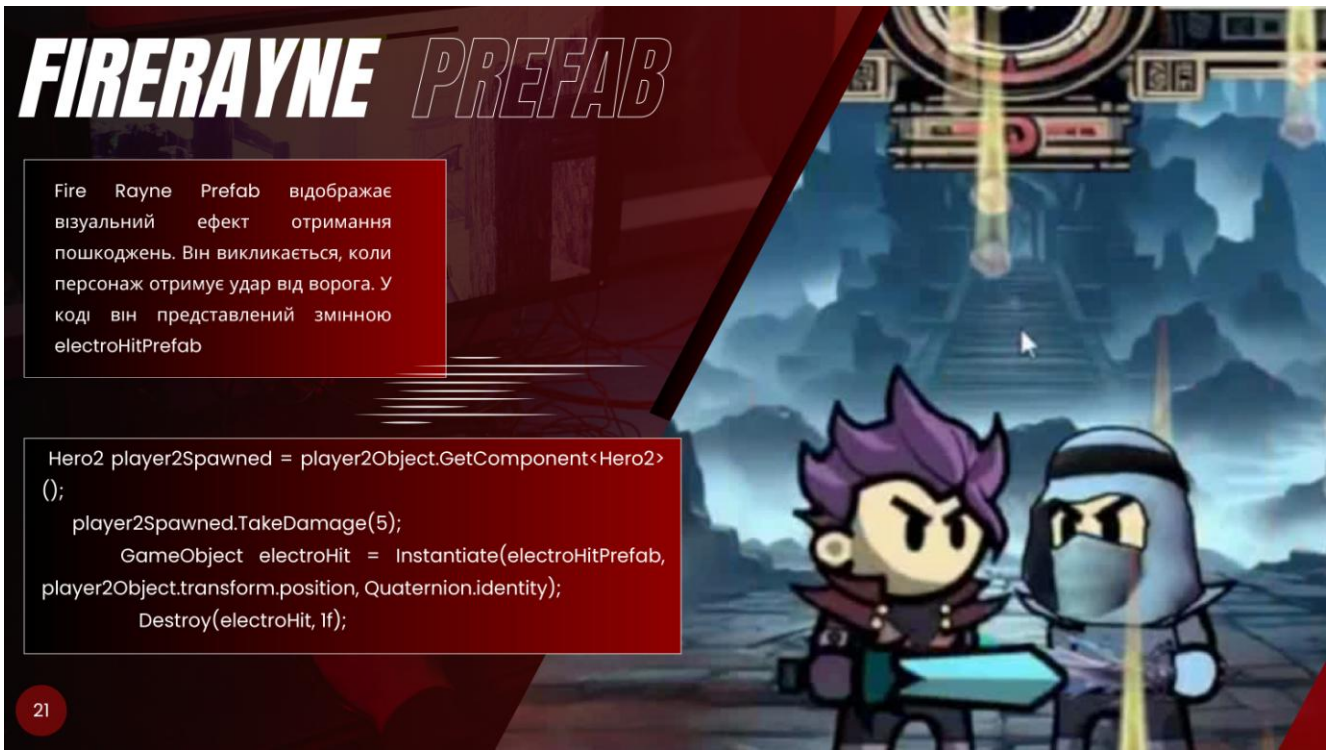
Скрипт CameraControl керує поведінкою камери у грі, зокрема її позиціонуванням та масштабуванням відносно двох гравців. Знаходяться координати точок, в яких в даний момент часу знаходяться персонажі. Камера розпалагається завжди між ними. Під час руху персонажів, скрипт вираховує відстань між гравцями та розміщує камеру строго посеред ними.

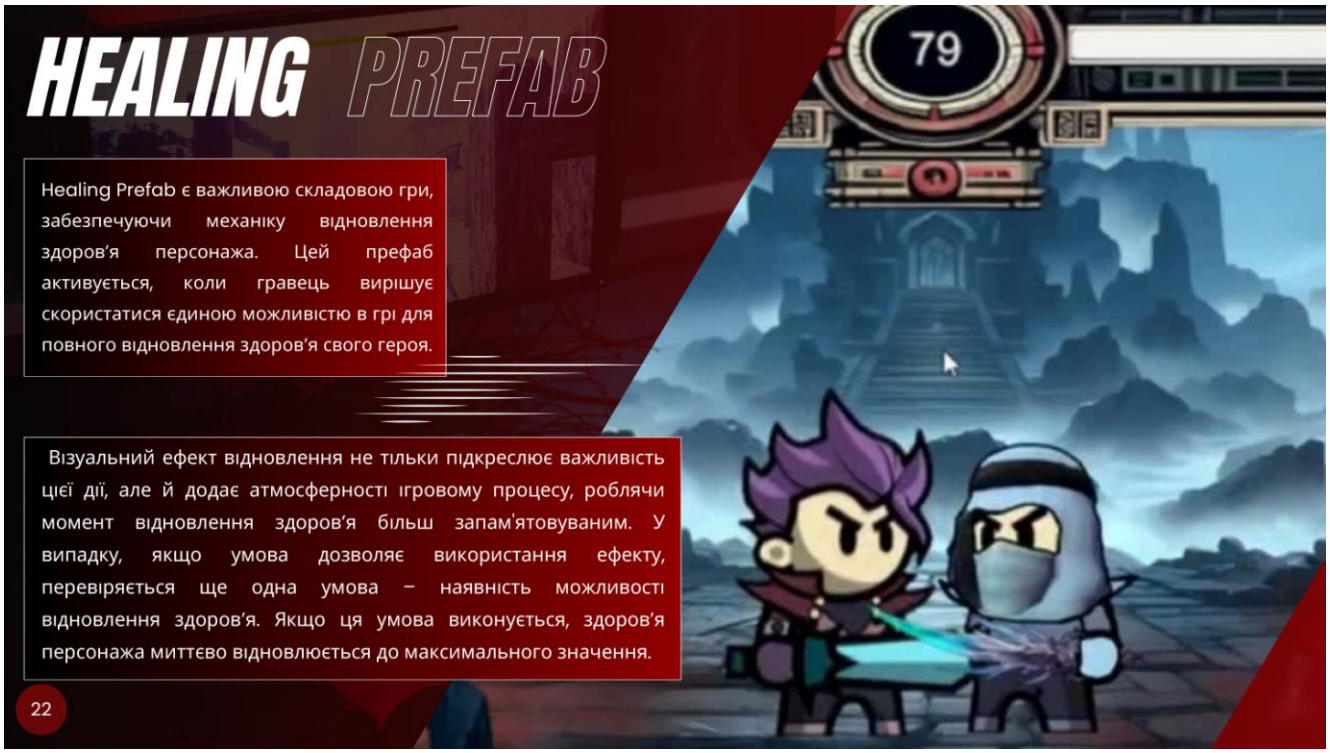
```
VECTOR3 CENTERPOINT = NEW
VECTOR3(((PLAYER1.TRANSFORM.POSITION.X
+ PLAYER2.TRANSFORM.POSITION.X) / 2,
0, 0);
TRANSFORM.POSITION = CENTERPOINT +
OFFSET;
```

18


ROUND CONTROL
87
Перемога в раундах
Зірки розташовані на екрані для обох гравців і заповнюються, коли гравець виграє раунд. Наприклад, коли гравець 1 виграє раунд, його зірка стає повністю видимою завдяки зміні прозорості кольору зірки (p1Star1.color та p1Star2.color). Таким чином, гравці можуть візуально бачити, скільки раундів вони виграли.
Таймер раунда
Таймер закінчення раунду, який зменшується щосекунди за допомогою функції DecreaseRoundTime(). Якщо таймер досягає нуля, викликається функція CheckRoundOutcome(), яка перевіряє, хто з гравців має більше життів або чи залишилися у них взагалі життя, щоб визначити переможця раунду.
82
Індікатор HP
Гравці починають з повними шкалами, і коли у одного з гравців закінчуються життя, раунд завершується, і викликається функція whoWinRound(), яка визначає переможця раунду.
19


EXPLOSION PREFAB
Explosion Prefab використовується для створення ефекту вибуху. Він викликається під час спеціальної атаки. У коді є змінна explosionPrefab, яка посилається на цей префаб.
player2Spawned.TakeDamage(25);
GameObject explosion = Instantiate(explosionPrefab, player2Object.transform.position, Quaternion.identity);
Destroy(explosion, 1f);
82
20


FIRERAYNE PREFAB
Fire Rayne Prefab відображає візуальний ефект отримання пошкоджень. Він викликається, коли персонаж отримує удар від ворога. У коді він представлений змінною electroHitPrefab
Hero2 player2Spawned = player2Object.GetComponent<Hero2>();
player2Spawned.TakeDamage(5);
GameObject electroHit = Instantiate(electroHitPrefab, player2Object.transform.position, Quaternion.identity);
Destroy(electroHit, 1f);
21


HEALING PREFAB
Healing Prefab є важливою складовою гри, забезпечуючи механіку відновлення здоров'я персонажа. Цей префаб активується, коли гравець вирішує скористатися єдиною можливістю в грі для повного відновлення здоров'я свого героя.
Візуальний ефект відновлення не тільки підкреслює важливість цієї дії, але й додає атмосферності ігровому процесу, роблячи момент відновлення здоров'я більш запам'ятовуваним. У випадку, якщо умова дозволяє використання ефекту, перевіряється ще одна умова – наявність можливості відновлення здоров'я. Якщо ця умова виконується, здоров'я персонажа миттєво відновлюється до максимального значення.
79
22

Тестування

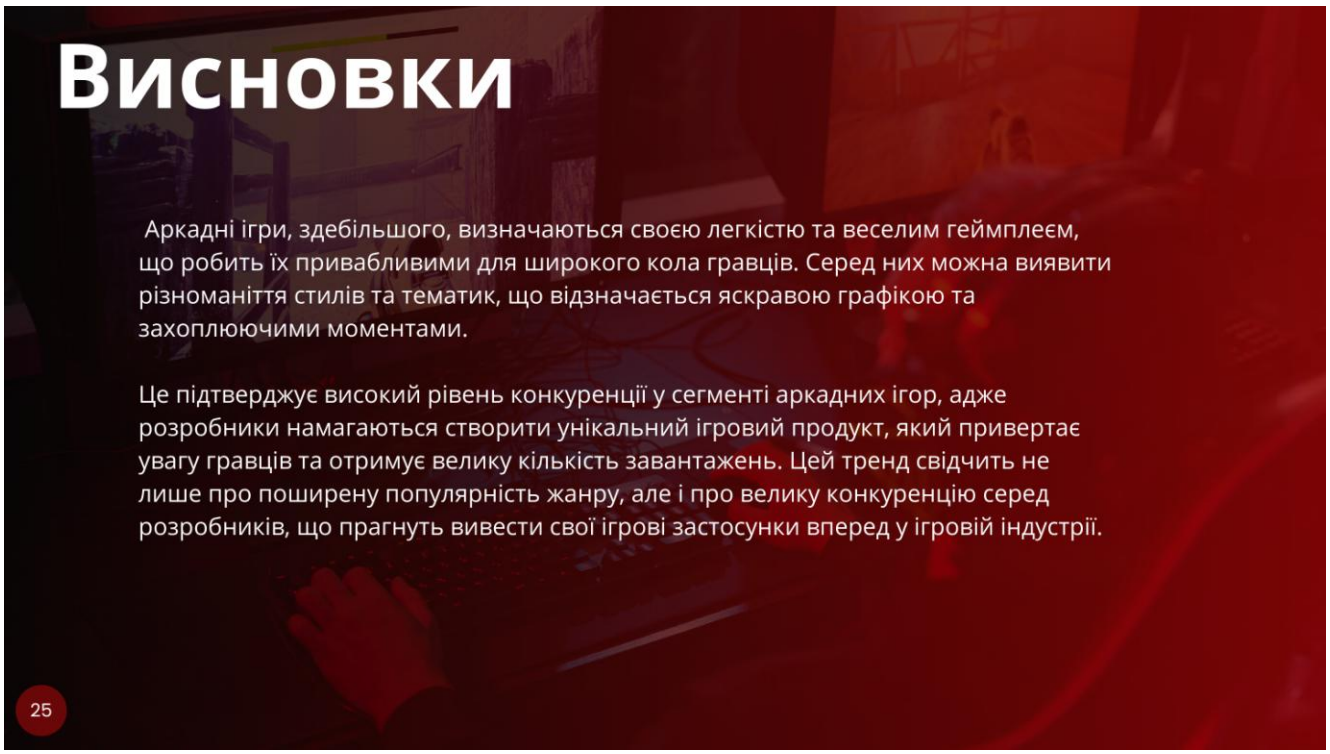
Для тестування продуктивності гри я використовував вбудований інструмент в Unity - Profiler. Це вид тестування Performance Testing, який відноситься до динамічного та інструментального підходу тестування.
Гра в цілому працює стабільно на рівні 60 FPS, що вказує на хорошу оптимізацію. Спостерігається один значний пік навантаження, де час обробки кадру зростає до приблизно 33 мс (30 FPS). Цей пік пов'язаний із завантаженням ресурсів ігреків на арені.

23

Тестування

Було перевірено, чи всі елементи користувацького інтерфейсу відображаються і працюють правильно. Зокрема, перевірка відображення шкали здоров'я, таймерів, зірок за перемогу в раундах та інших елементів інтерфейсу.
У верхній частині екрану присутня шкала з часом до закінчення раунда посередині. Перевірено, чи коректно відображається ця шкала та чи оновлюється значення в реальному часі. Перевірено, чи коректно відображаються обидва ігрові персонажі на екрані, включаючи їхню анімацію та позиціонування. Протестовано правильність відображення фонового пейзажу з горами та місяцем, а також його взаємодію з іншими елементами інтерфейсу.
В ході виконання даного тесту було виявлено, що UI елемент з декоративною колоною зліва перекриває UI елемент відображення зірок, які ідентифікують перемогу у раунді.

24


Висновки
Аркадні ігри, здебільшого, визначаються своєю легкістю та веселим геймплеєм, що робить їх привабливими для широкого кола гравців. Серед них можна виявити різноманіття стилів та тематик, що відзначається яскравою графікою та захоплюючими моментами.
Це підтверджує високий рівень конкуренції у сегменті аркадних ігор, адже розробники намагаються створити унікальний ігровий продукт, який привертає увагу гравців та отримує велику кількість завантажень. Цей тренд свідчить не лише про поширену популярність жанру, але і про велику конкуренцію серед розробників, що прагнуть вивести свої ігрові застосунки вперед у ігровій індустрії.
25

Додаток В

Код програми

//CameraControl.cs

```
using UnityEngine;

public class CameraControl : MonoBehaviour
{
    private GameObject player1;
    private GameObject player2;

    private Vector3 offset;
    private float cameraDistance = 10f;
    private float zoomSpeed = 2f;

    void Start()
    {

    }

    private void LateUpdate()
    {

        if (player1 == null)
        {
            player1 = GameObject.FindGameObjectWithTag("player1choosed");
        }

        if (player2 == null)
        {
            player2 = GameObject.FindGameObjectWithTag("player2choosed");
        }

        if (player1 != null && player2 != null)
        {

            Vector3 centerPoint = new Vector3(   ((player1.transform.position.x +
player2.transform.position.x) / 2), 0, 0);

            transform.position = centerPoint + offset;
            float scroll = Input.GetAxis("Mouse ScrollWheel");
            cameraDistance = Mathf.Clamp(cameraDistance - scroll * zoomSpeed, 5f, 20f);
            offset.z = -cameraDistance;
        }
    }

}
```

//ChooseArena.cs

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

public class ChooseArena : MonoBehaviour
{
    public static GameObject chosenArena;
```

```
    private void Start()
    {
    }



    public void goToArena1()
    {
        SceneManager.LoadScene(4);
    }

    public void goToArena2()
    {
        SceneManager.LoadScene(5);
    }

    public void goToArena3()
    {
        SceneManager.LoadScene(6);
    }

}
```

//ChoosePlayer.cs

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

public class ChoosePlayer : MonoBehaviour
{
    public static GameObject chosenHero;
    public skin[] skins;
    [SerializeField] private bool firstChoosing = true;
    public static string chosenSkinPlayer1 = ""; //add

    private void Start()
    {
       if (firstChoosing)
        {
            repaintSkinsP1();
        }
        else
        {
            repaintSkinsP2();
        }
    }

    private void repaintSkinsP1()
    {
        string chosenSkin = PlayerPrefs.GetString("choosed_player", "hero1");

        if (skins != null)
        {
            foreach (skin sk in skins)
            {
                // Перевірка на null
                if (sk != null)
                {
                    if (sk.skinName == chosenSkin)
                    {
```

```
                    sk.isChoosed();
                }
                else
                {
                    sk.skinAvailable();
                }
            }
        }
    }
}

private void repaintSkinsP2()
{
    string chosenSkin2 = PlayerPrefs.GetString("choosed_player2", "");

    if (skins != null)
    {
        foreach (skin sk in skins)
        {
            // Перевірка на null
            if (sk != null)
            {
                if (sk.skinName == chosenSkinPlayer1)
                {
                    sk.blockForChoose();
                }
                else if (sk.skinName == chosenSkin2)
                {
                    sk.isChoosed();
                }
                else
                {
                    sk.skinAvailable();
                }
            }
        }
    }
}

public void make_choose_player1(string Choosed_Player)
{
    PlayerPrefs.SetString("choosed_player", Choosed_Player);
    chosenSkinPlayer1 = Choosed_Player; //add
    Debug.Log("choosed_player" + Choosed_Player);
    repaintSkinsP1();
}

public void make_choose_player2(string Choosed_Player)
{
    PlayerPrefs.SetString("choosed_player2", Choosed_Player);
    Debug.Log("choosed_player2" + Choosed_Player);
    repaintSkinsP2();
}

public void findNotAwable() { }

public void goToSceneChoosePlayer1()
{
    SceneManager.LoadScene(1);
}

public void goToSceneChoosePlayer2()
{
    SceneManager.LoadScene(2);
}
```

```
    public void goToSceneChooseArena()
    {
        SceneManager.LoadScene(3);
    }


    public void StartGame()
    {
        foreach (skin sk in skins)
        {
            sk.resetSkin();
        }

        if (firstChoosing)
        {
            SceneManager.LoadScene(2);
        }
        else
        {
            SceneManager.LoadScene(3);
        }
    }

}
```

//hero.cs

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

public class Hero : HeroBehaviour
{

    public Hero2 player2Spawned;
    public delegate void changeHp(int hp);
    public event changeHp OnHpChanged;


    private void Awake()
    {
        rb = GetComponent<Rigidbody2D>();
        anim = GetComponent<Animator>();
        sprite = GetComponentInChildren<SpriteRenderer>();
        state = States.idle;
        GameObject player2Object = GameObject.FindGameObjectWithTag("player2choosed");
    }

    void Update()

    {
        if (!isAlive)
        {
            return;
        }
        if (!isFrozen)
        {
            CheckGround();
            if (Input.GetButton("Horizontal"))
                isMoving = true;
            else
                isMoving = false;
```

```
            Run();
            if (Input.GetButtonDown("Jump") && onGround)
                Jump();
            if (Input.GetButtonDown("Fire1"))
                AttemptAttack();

            if (Input.GetKeyDown(KeyCode.R))
                hilka();

            if (Time.time - lastSpecialAttackTime >= 30f && Input.GetKeyDown(KeyCode.Q))
                AttemptSpecialAttack();

            if (Input.GetKeyDown(KeyCode.E))
                AttemptElectroAttack();

            if (Input.GetKeyDown(KeyCode.T))
                AttemptIceFlorAttack();
        }

    }

    protected override void SpecialAttack()
    {
        if(isFrozen)
            return;

        if (explosionTriggered) return;

        state = States.attack;
        Debug.Log("Player 1`s special attacking");

        GameObject player2Object = GameObject.FindGameObjectWithTag("player2choosed");
        if (player2Object != null)
        {
            Hero2 player2Spawned = player2Object.GetComponent<Hero2>();
            if (player2Spawned != null)
            {
                if (distance < 2.5f && distance > 1.8f)
                {
                    player2Spawned.TakeDamage(25);
                    GameObject explosion = Instantiate(explosionPrefab,
player2Object.transform.position, Quaternion.identity);
                    Destroy(explosion, 1f);
                }
                else if (distance < 1.8f)
                {
                    player2Spawned.TakeDamage(40);
                    GameObject explosion = Instantiate(explosionPrefab,
player2Object.transform.position, Quaternion.identity);
                    Destroy(explosion, 1f);
                }
                else
                {
                    Debug.Log("Miss!!!");
                }
            }
            else
            {
                Debug.LogError("player2Spawned is null!");
            }
        }
        else
        {
            Debug.LogError("GameObject with tag 'player2choosed' not found!");
```

```
        }

        explosionTriggered = true;
        Hero heroInstance = GetComponent<Hero>();
        heroInstance.ResetExplosionTrigger();
        Invoke("ResetExplosionTrigger", 30f);
    }

    protected override void ElectroAttack()
    {
        if(isFrozen)
            return;

        state = States.attack;
        Debug.Log("Player 1`s electro attacking");

        GameObject player2Object = GameObject.FindGameObjectWithTag("player2choosed");
        if (player2Object != null)
        {
            Hero2 player2Spawned = player2Object.GetComponent<Hero2>();
            if (player2Spawned != null)
            {
                if (distance < 5f)
                {
                    player2Spawned.TakeDamage(5);
                    GameObject electroHit = Instantiate(electroHitPrefab,
player2Object.transform.position, Quaternion.identity);
                    Destroy(electroHit, 1f);
                }
                else
                {
                    Debug.Log("Miss!!!");
                }
            }
            else
            {
                Debug.LogError("player2Spawned is null!");
            }
        }
        else
        {
            Debug.LogError("GameObject with tag 'player2choosed' not found!");
        }

        Hero hero1Instance = GetComponent<Hero>();
    }

    protected override void IceFlorAttack()
    {

         if(isFrozen)
            return;

        state = States.attack;
        Debug.Log("Player 1`s ice flor attacking");

        GameObject player2Object = GameObject.FindGameObjectWithTag("player2choosed");
        if (player2Object != null)
        {
            Hero2 player2Spawned = player2Object.GetComponent<Hero2>();
            if (player2Spawned != null )
            {
                if (distance < 5f)
                {
                    player2Spawned.TakeDamage(25);
```

```
                    GameObject IceFlorHit = Instantiate(IceFlorPrefab,
player2Object.transform.position, Quaternion.identity);
                    Destroy(IceFlorHit, 2f);
                    player2Spawned.FreezeHero(2f);
                    Debug.Log("Игрок 2 заморожен");
                }
                else
                {
                    Debug.Log("Miss!!!");
                }
            }
            else
            {
                Debug.LogError("player2Spawned is null!");
            }
        }
        else
        {
            Debug.LogError("GameObject with tag 'player2choosed' not found!");
        }

        Hero hero1Instance = GetComponent<Hero>();
    }

    protected override void Attack()
    {
        if(isFrozen)
            return;

        state = States.attack;
        Debug.Log("Player 1`s attacking");

        GameObject player2Object = GameObject.FindGameObjectWithTag("player2choosed");
        if (player2Object != null)
        {
            Hero2 player2Spawned = player2Object.GetComponent<Hero2>();
            if (player2Spawned != null)
            {
                if (distance < 2.0f && distance > 1.8f)
                {

                    Invoke("showGetDammageEffect", 0.3f);
                    player2Spawned.TakeDamage(3);

                }
                else if (distance < 1.8f)
                {

                     Invoke("showGetDammageEffect", 0.3f);
                      player2Spawned.TakeDamage(5);

                }
                else
                {
                    Debug.Log("Miss!!!");
                }
            }
            else
            {
                Debug.LogError("player2Spawned is null!");
            }
        }
        else
        {
            Debug.LogError("GameObject with tag 'player2choosed' not found!");
```

```
        }

    }



    public void showGetDammageEffect (){
        GameObject player2Object = GameObject.FindGameObjectWithTag("player2choosed");

        GameObject getDammage = Instantiate(getDammagePrefab,
player2Object.transform.position, Quaternion.identity);
                    Destroy(getDammage, 1f);
    }

    private void Run()
    {
        if(isFrozen)
            return;

        if (state != States.nockout && isMoving)
        {
            state = States.run;

            float horizontalInput = Input.GetAxis("Horizontal");
            Vector3 dir = transform.right * horizontalInput;
            transform.position = Vector3.MoveTowards(transform.position, transform.position +
dir, speed * Time.deltaTime);
            sprite.flipX = dir.x < 0.0f;
        }
        else
        {
            state = States.idle;
        }
    }

    public void hilka()
    {
        if(isFrozen)
            return;

        if (ToDoHilka > 0)
        {
            lives = 100;
            OnHpChanged?.Invoke(lives);
            GameObject explosion = Instantiate(healingPrefab, transform.position,
Quaternion.identity);
            Destroy(explosion, 1.5f);
            ToDoHilka--;
        }
    }

    public void TakeDamage(float damage)
    {
       if (!isAlive || isFrozen){
            return;
        }


        lives -= (int)damage;
        Debug.Log("Player1 получает урон " + damage + ". Осталось HP: " + lives);

        OnHpChanged?.Invoke(lives);

        if (lives <= 0)
        {
```

```
            roundController.CheckRoundOutcome();
        }
    }

    public void resetHp()
    {
        lives = 100;
        OnHpChanged?.Invoke(lives);
        isAlive = true;
    }

}
```

//roundControl.cs

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

public class roundControl : MonoBehaviour
{
    private Hero hero;
    private Hero2 hero2;
    private float waitTime = 0.5f;
    public int rounds = 3;
    public int roundTime = 90;
    private int player1lives;
    private int player2lives;
    public Text roundTimeText;
    public Text winnerIs;

    public int player1RoundWins = 0;
    public int player2RoundWins = 0;
    public SpawnPlayer spawnPlayer;

    public Image p1Star1;
    public Image p1Star2;
    public Image p2Star1;
    public Image p2Star2;

    public Image[] roundsBanners;

    private Color32 starColor;

    private bool roundOver = false;

    void Start()
    {
        roundsBanners[2].gameObject.SetActive(true);

        Invoke("ident", waitTime);
        InvokeRepeating("DecreaseRoundTime", 1f, 1f);
        starColor = p1Star1.color;
        p1Star1.color = new Color32(starColor.r, starColor.g, starColor.b, 20);
        p1Star2.color = new Color32(starColor.r, starColor.g, starColor.b, 20);
        p2Star1.color = new Color32(starColor.r, starColor.g, starColor.b, 20);
        p2Star2.color = new Color32(starColor.r, starColor.g, starColor.b, 20);
    }

    public void CheckRoundOutcome()
```

```
    {
        if (roundOver) return;

        if (hero.lives <= 0 || hero2.lives <= 0 || roundTime <= 0)
        {
            roundOver = true;
            whoWinRound();

            if (player1RoundWins >= 2)
            {
                finishFight();
            }
            if (player2RoundWins >= 2)
            {
                finishFight();
            }
        }
    }

    private void ident()
    {
        hero = GameObject.FindGameObjectWithTag("player1choosed").GetComponent<Hero>();
        if (hero != null)
        {
            player1lives = hero.lives;
        }

        hero2 = GameObject.FindGameObjectWithTag("player2choosed").GetComponent<Hero2>();
        if (hero2 != null)
        {
            player2lives = hero2.lives;
        }
    }

    public void whoWinRound()
    {
        if (roundTime <= 0)
        {
            if (hero.lives > hero2.lives)
            {
                player1RoundWins++;
                Debug.Log("Player 1 win");

                hero2.roundLoose();
                p1Star1.color = (player1RoundWins >= 1) ? new Color32(starColor.r,
starColor.g, starColor.b, 255) : p1Star1.color;
                p1Star2.color = (player1RoundWins >= 2) ? new Color32(starColor.r,
starColor.g, starColor.b, 255) : p1Star2.color;

                finishRound();
            }
            else if (hero2.lives > hero.lives)
            {
                player2RoundWins++;
                hero.roundLoose();
                Debug.Log("Player 2 win");
                p2Star1.color = (player2RoundWins >= 1) ? new Color32(starColor.r,
starColor.g, starColor.b, 255) : p2Star1.color;
                p2Star2.color = (player2RoundWins >= 2) ? new Color32(starColor.r,
starColor.g, starColor.b, 255) : p2Star2.color;
                finishRound();
            }
            else
            {
                Debug.Log("Ничья");
```

```
                reFight();
            }
        }
        else if (hero.lives <= 0)
        {
            player2RoundWins++;
            p2Star1.color = (player2RoundWins >= 1) ? new Color32(starColor.r, starColor.g,
starColor.b, 255) : p2Star1.color;
            p2Star2.color = (player2RoundWins >= 2) ? new Color32(starColor.r, starColor.g,
starColor.b, 255) : p2Star2.color;
            hero.roundLoose();
            Debug.Log("Player 2 win");
            finishRound();
        }
        else if (hero2.lives <= 0)
        {
            player1RoundWins++;
            p1Star1.color = (player1RoundWins >= 1) ? new Color32(starColor.r, starColor.g,
starColor.b, 255) : p1Star1.color;
            p1Star2.color = (player1RoundWins >= 2) ? new Color32(starColor.r, starColor.g,
starColor.b, 255) : p1Star2.color;
            hero2.roundLoose();
            Debug.Log("Player 1 win");
            finishRound();
        }
    }

    public void finishRound()
    {
        rounds--;
        Debug.Log("Round finished");
        Invoke("StartNewRound", 1.5f);
    }

    public void reFight()
    {
        StartNewRound();
    }

    public void finishFight()
    {
        whoWinFight();
        Invoke("nockout", 1f);
    }

    public void whoWinFight()
    {
        if (player1RoundWins >= 2)
        {
            Debug.Log("The winner is PLAYER 1");
        }
        else if (player2RoundWins >= 2)
        {
            Debug.Log("The winner is PLAYER 2");
        }
    }

    public void DecreaseRoundTime()
    {
        if (roundTime > 0)
        {
            roundTime--;
            roundTimeText.text = roundTime.ToString();
            if (roundTime <= 0)
            {
```

```
                CheckRoundOutcome();
            }
        }
    }

    private void ResetPlayerPositions()
    {
        spawnPlayer.ResetPlayerPositions();
    }

    private void ResetPlayerLives()
    {
        hero.resetHp();
        hero2.resetHp();
    }

    public void StartNewRound()
    {
        roundOver = false;
        roundTime = 90;
        roundTimeText.text = roundTime.ToString();
        ResetPlayerPositions();
        ResetPlayerLives();
        roundsBanners[rounds-1].gameObject.SetActive(true);
    }

    public void newBanner()
    {
        ResetPlayerPositions();
        ResetPlayerLives();
    }

    public void nockout()
    {
              Debug.Log("The fight is over");
        if (player1RoundWins == 2) {
            Debug.Log("Player 1 win");
            player1WinFight();

        }
        else if (player2RoundWins == 2){
            Debug.Log("Player 2 Win");
            player2WinFight();

        }
    }

    public void player1WinFight(){
        PlayerPrefs.SetInt("prevFightSceneIndex", SceneManager.GetActiveScene().buildIndex);
        SceneManager.LoadScene(7);
    }

     public void player2WinFight(){
        PlayerPrefs.SetInt("prevFightSceneIndex", SceneManager.GetActiveScene().buildIndex);
        SceneManager.LoadScene(8);
    }



}
```