Вінницький національний технічний університет
(повне найменування вищого навчального закладу)

Факультет інтелектуальних інформаційних технологій та автоматики
(повне найменування інституту, назва факультету (відділення))

Кафедра комп'ютерних наук
(повна назва кафедри (предметної, циклової комісії))

Бакалаврська дипломна робота на тему:

«КОМП'ЮТРЕНА ГРА ЛОНГБОРДИНГ НА БАЗІ ІГРОВОГО РУШІЯ UNITY»

Виконав: студент 4 курсу, групи 3КН-20б
спеціальності 122 – Комп'ютерні науки
(шифр і назва напряму підготовки, спеціальності)
Матвієнко А. І.
(прізвище та ініціали)

Керівник: асистент кафедри КН
Малініч І. П.
(прізвище та ініціали)

«____» _____ 2024 р.
Рецензент: к.т.н., доцент кафедри АІІТ
Кулик Я. А.
(прізвище та ініціали)

«07» 06 2024 р.

Допущено до захисту
Зав. кафедри Яровий А. А.
«07» 06 2024 р.

Вінниця ВНТУ – 2024 рік

Вінницький національний технічний університет

Факультет інтелектуальних інформаційних технологій та автоматизації

Кафедра Комп'ютерних наук

Рівень вищої освіти – перший (бакалаврський)

Галузь знань – 12 «Інформаційні технології»

Спеціальність – 122 «Комп'ютерні науки»

Освітньо-професійна програма – «Комп'ютерні науки»

ЗАТВЕРДЖУЮ

Завідувач кафедри КН

проф., д.т.н. Яровий А. А.

«07» 03 2024р.

ЗАВДАННЯ

НА БАКАЛАВРСЬКУ ДИПЛОМНУ РОБОТУ СТУДЕНТУ

Матвієнко Андрію Ігоровичку

1. Тема роботи: Комп'ютерна гра лонгбординг на базі ігрового рушія Unity

керівник роботи: Малініч І. П. асистент. каф. КН
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу «11» 03 2024 року №80

2. Термін подання студентом роботи: 27.05.2024 р.

3. Вихідні дані до роботи: кількість гравців – 1, розширення екрану – 1920x1080, частота кадрів – 60 кадрів на секунду.

4. Зміст текстової частини (перелік питань, які потрібно розробити): аналіз предметної області, аналіз існуючих концепцій ігор жанру "лонгбординг", розробка і проектування програмного модуля, проектування інтерфейсу користувача, проектування ігрового циклу, розробка комп'ютерної гри жанру "лонгбординг" на базі ігрового рушія Unity, розробка графічних матеріалів, розробка користувацького інтерфейсу, розробка основних модулів додатку, тестування додатку.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень): схеми користувацького інтерфейсу, блок–схема алгоритму роботи програми, функціональна модель персонажа, модель реалізації ігрової системи, діаграма ігрового циклу, зображення ігрового процесу.

6. Консультанти розділів проекту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1, 2, 3	Маліній І. П., асистент каф.КН		

7. Дата видачі завдання 07.03.2024

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проекту (роботи)	Строк виконання етапів проекту (роботи)		Примітка
		Початок	Закінчення	
1	Обгрунтування доцільності створення комп'ютерної гри лонгбординг на базі ігрового рушія Unity	06.05.24	11.05.24	
2	Аналіз існуючих продуктів, ігрового рушія та середовищ розробки	11.05.24	20.05.24	
3	Розробка і проектування програмного модуля комп'ютерної гри	20.05.24	24.05.24	
4	Розробка комп'ютерної гри «лонгбординг» на базі ігрового рушія Unity	24.05.24	27.05.24	
5	Тестування комп'ютерної гри	27.05.24	29.05.24	
6	Оформлення пояснювальної записки і графічного матеріалу	29.05.24	06.06.24	

Студент _____ Матвієн
(підпис) (прізвище т

Керівник проекту (роботи) _____ Малі
(підпис) (прізвище т

АНОТАЦІЯ

Бакалаврська дипломна робота складається з 86 сторінок формату А4, на яких є 65 рисунки, список використаних джерел містить 14 найменувань.

Бакалаврська робота присвячена створенню комп’ютерної гри «Лонгбординг» на базі ігрового рушія Unity. У цій бакалаврській роботі були розроблені скрипти, необхідні для функціонування гри та всі необхідні елементи інтерфейсу на базі Unity з використанням мови програмування C#.

Ключові слова: комп’ютерна гра, лонгборд, Unity, збереження, скрипт, геймплей.

ABSTRACT

The bachelor thesis consists of 86 pages of A4 format, on which there are 65 figures, the list of used sources contains 14 names.

The bachelor thesis is devoted to the creation of the computer game "Longboarding" based on the Unity game engine. In this bachelor's work, the scripts necessary for the functioning of the game and all the necessary interface elements were developed based on Unity using the C# programming language.

Keywords: computer game, longboard, save Unity, script, gameplay.

ЗМІСТ

ВСТУП

Актуальність теми

Розробка комп’ютерної гри «Лонгбординг» є дуже актуальною, оскільки існує багато користувачів, яким цікава тема лонгбордингу в цілому. Часто саме цим користувачам потрібен певний спосіб перевірити, чи дійсно цей вид спорту може бути їм цікавий без великих фінансових інвестицій. Комп’ютерна гра може бути найкращим способом надання такої інформації, оскільки дозволяє легко та зручно отримувати імітацію відчуття катання на лонгборді.

Буде проведено аналіз найбільш популярних ігор з тематикою лонгбордингу. Переваги і недоліки кожної гри зважуватимуться для вдосконалення фінального продукту. Крім того, аналізуватимуться різні ігрові рушії та середовища розробки для оптимізації процесу розробки. Важливим етапом роботи також буде розробка скриптів для деяких функцій гри, таких як фізика руху, управління персонажем, таймер тощо.

Дослідження допоможе створити ефективну комп’ютерну гру, яка полегшить гравцям занурення в світ лонгбордингу з урахуванням сучасних технологічних вимог і змін на ринку.

Мета дослідження

Мета дослідження полягає у розширенні функціональних можливостей комп’ютерних ігор жанру лонгбординг та впровадженні ефективних методів проектування, розробки та тестування 3D гри у жанрі лонгбординг, що забезпечить високу якість користувацького досвіду, стабільність роботи гри та її конкурентоспроможність на ринку комп'ютерних ігор.

Задачі дослідження

Для досягнення поставлених цілей необхідно подальше вирішення наступних завдань дослідження:

- Обгрунтування доцільності створення комп’ютерної гри «Лонгбординг»

- Аналіз існуючих рішень: Провести аналіз існуючих 2D і 3D ігор у жанрі лонгбординг для визначення найкращих практик та виявлення можливостей для інновацій.
- Проектування програмного модуля комп’ютерної гри.
- Реалізація комп’ютерної гри “лонгбординг” на базі ігрового рушія Unity.
- Тестування комп’ютерної гри, щоб впевнетись у відсутності збоїв у грі.

Предметом дослідження є алгоритми роботи копм’ютерної гри жанру “лонгбординг”, моделі реалізації ігрових систем, комп’ютерна гра жанру “лонгбординг” з вибором рівнів, користувацький інтерфейс, ігровий цикл.

Об'єктом дослідження є процес створення 3D гри у жанрі лонгбординг, включаючи проектування, розробку, тестування та оптимізацію її компонентів, таких як користувацький інтерфейс, ігровий цикл, алгоритми роботи та інтеграція ігрових систем.

Практичною цінністю одержаних результатів є модульні скрипти, написані у процесі створення гри, які розробники можуть використовувати у інших проектах.

Публікації

За результатами бакалаврської дипломної роботи опублікована теза доповіді на конференції «IX Міжнародна науково-практична конференція INNOVATIVE DEVELOPMENT OF SCIENCE, TECHNOLOGY AND EDUCATION» (м. Ванкувер, Канада) [1].

Впровадження

Результати бакалаврської роботи було впроваджено згідно з актом впровадження № 2 / 31.05.2024.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ КОМП'ЮТЕРНОЇ ГРИ «ЛОНГБОРДИНГ»

1.1 Обгрунтування доцільності

Розробка комп'ютерної гри "Лонгбординг" на базі ігрового рушія Unity є важливим кроком у розвитку комп'ютерних ігор, що відповідають сучасним запитам користувачів та ринковим тенденціям. Це обґрунтовується низкою причин.

Популярність лонгбордингу та інтерес до нього. Лонгбординг, як вид спорту, набуває все більшої популярності серед молоді та дорослих. Багато людей цікавляться цим видом активності, але не всі мають можливість випробувати його на практиці через фінансові, географічні або фізичні обмеження. Комп'ютерна гра може стати доступним та безпечним способом випробувати лонгбординг, що дозволяє гравцям відчути адреналін та насолоду від цього спорту без ризику для здоров'я та значних фінансових витрат.

Відсутність достатньої кількості якісних ігор на ринку. Аналіз існуючих рішень показує, що на ринку наразі існує обмежена кількість високоякісних ігор у жанрі лонгбордингу. Більшість з них або занадто прості, або не відповідають сучасним вимогам щодо графіки та фізики руху. Створення гри на базі Unity, яка враховує переваги і недоліки попередників, дозволить створити інноваційний продукт з високою якістю користувацького досвіду.

Розглянуті нижче ігри використовували ігровий рушій Unity. Unity є потужним ігровим рушієм, який використовується для створення дво- та тривимірних ігор, а також інтерактивних додатків. Unity забезпечує розробників зручним середовищем для розробки, яке включає в себе фізику, анімацію, рендеринг та багато інших функцій, необхідних для створення сучасних ігор. [2]

Основними перевагами Unity є:

1. Гнучкість та підтримка: Unity та пов'язані з ним технології широко поширені та добре підтримуються. Це надає розробникам гнучкість

у створенні ігор з різними функціями та дизайном. Завдяки великій спільноті користувачів та розробників Unity, ці технології мають багато ресурсів та підтримки.

2. Приваблива крива навчання: Щоб працювати з Unity, не потрібно володіти глибокими знаннями всіх перерахованих технологій. Новачки можуть використовувати візуальний інтерфейс Unity для створення базової гри без коду. Досвідчені розробники можуть використовувати розширені можливості Unity для створення складних і функціональних ігор з унікальним дизайном. Використання C# при розробці гри на Unity дозволяє створювати сучасні, динамічні, візуально привабливі та зручні для користувача ігри.
3. Спільнота: Unity має велику спільноту розробників, яка надає багато безкоштовних та платних курсів, ресурсів та підтримки. Тому, вирішуючи завдання, легко можна знайти відповідь на поставлене запитання, що робить процес розробки більш ефективним та приємним.

1.2 Аналіз існуючих продуктів

Ігри в жанрі «лонгбординг» є досить рідкісним явищем для ігрової індустрії у зв’язку з низькою обізнаністю щодо відмінностей лонгбордингу від звичайного скейтбордингу. Для пошуку існуючих продуктів було використано вебсайт itch.io.

На рисунку 1.1 подано знімок екрану геймплейного процесу гри «Tanuki Sunset». Впродовж геймплейного процесу відчутно особливості механіки лонгборду та видно своєрідну цікаву стилізацію гри. Під час геймплейного процесу можна спостерігати багато наявних функцій гри, такі як своєрідне керування, особливості фізики та привабливий геймплейний процес [3].

До переваг описаного додатка можна віднести цікавий та своєрідний стиль, широкий спектр механік. Також не обійшлося без мінусів, серед яких

один з найбільших – нереалістичність керування, адже вона позиціонує себе саме як гра Лонгбординг, але більшість механік гри є занадто аркадними та не мають нічого спільного з лонгбордингом. Також до мінусів можна віднести навколишнє середовище, адже воно має занадто яскраві кольори і високий контраст, який може викликати негативні відчуття у гравців.

Рисунок 1.1 – Знімок екрану під час геймплейного процесу гри «Tanuki Sunset»

Розглянемо дизайн геймплейного процесу гри «Turn your Longboard into a game controller», який наведений на рисунку 1.2. Ця гра має унікальну особливість, яка робить її виділяється серед інших – можливість використання власного лонгборду у якості контроллера. Це додає грі особливого шарму і може привабити любителів лонгбордингу, оскільки вони зможуть відчути справжню їзду на лонгборді, керуючи ним під час гри [4].

Сама гра також позиціонує себе як гра з тематикою Лонгбординг, але її візуальний дизайн значно спрощений. Вона не має складних графічних ефектів або деталізованих моделей, що може бути як плюсом, так і мінусом, залежно від уподобань гравців. Основною «родзинкою» гри є саме можливість використання

власного лонгборду у якості контроллера, що є інноваційним підходом і може значно покращити ігровий досвід.

Таким чином, гра «Turn your Longboard into a game controller» має потенціал завдяки своїй унікальній особливості, але для досягнення справжнього успіху необхідно вдосконалити її візуальний дизайн та геймплей. Це дозволить привабити більше гравців і зробити гру більш конкурентоспроможною на ринку ігор.

Рисунок 1.2 – Знімок екрану під час геймплейного процесу гри «Turn your Longboard into a game controller»

Проаналізуємо геймплей та дизайн гри «Skater-girl (2018)» [5] (рис. 1.3). Гра виконана у мінімалістичному дизайні, який не перенавантажує очі гравця, геймплей максимально простий, але частково реалістичний, у гри немає основної «Родзинки», яка робила б її відмінною від ігор про скейтбординг в цілому. З головних переваг варто виділити мінімалістичний дизайн гри, який є її

найбільшим плюсом. Цей дизайн дозволяє гравцям зосередитися на самій грі, а не відволікатися на другорядні деталі. Крім того, простота графіки дозволяє грі працювати плавно навіть на менш потужних пристроях, що робить її доступною для ширшої аудиторії. З недоліків – відсутність цілі та загальне відчуття незавершеності у проєкта. Гра могла б значно виграти від додавання чітких завдань та досягнень, які б мотивували гравців продовжувати грати. Крім того, введення нових режимів гри або додаткових елементів, таких як кастомізація персонажа або змагальні режими, могло б зробити гру більш захопливою та різноманітною. Таким чином, «Skater-girl (2018)» є приємною грою з мінімалістичним дизайном, але вона потребує деяких поліпшень, щоб стати більш привабливою для широкого кола гравців. Важливо, щоб розробники врахували відгуки гравців і додали в гру елементи, які б підвищили її реіграбельність та зробили її більш унікальною на ринку ігор про скейтбординг.

Рисунок 1.3 – Знімок екрану під час геймплейного процесу гри «Skater-girl (2018)»

На основі вище наведених характеристик було створено таблицю переваг та недоліків існуючих рішень і створюваного продукту.

Таблиця 1.1 – Порівняльна характеристика ігор

Назва	Tanuki Sunset	Turn your Longboard into a game controller	Skater-girl (2018)	Розроблений додаток
Цікавість геймплею	Так	Так	Так	Так
Реалістичність геймплею	Ні	Ні	Так	Так
Графічна привабливість	Так	Ні	Так	Так
Особлива схема керування	Ні	Так	Ні	Ні
Графічне перенавантаження рівнів	Так	Ні	Ні	Ні
Основна ціль	Так	Ні	Ні	Так
Відслідковування прогресу гравця	Так	Ні	Ні	Так

На основі проведеного аналізу існуючих ігор на тему лонгбордингу можна дійти висновку, що створення комп’ютерної гри на тему «Лонгбординг» є доцільним та має практичне значення.

Така гра буде направлена саме в геймплей та симуляцію відчуття катання на лонгборді, а не аркадність та особливі схеми керування. Така гра буде безкоштовною. Ключовими особливостями створення гри «Лонгбординг» є налаштування ігрового інтерфейсу та приділення уваги підбору кольорових відтінків. Налаштування інтерфейсу та рівнів буде виконано з акцентом на привабливий дизайн та вибір гармонійних кольорів, щоб забезпечити візуально приємний ігровий досвід. Особливостями розробки стануть наявність стильного та унікального дизайну, особливого реалістичного геймплею.

1.3 Постановка задачі дослідження

Проаналізувавши існуючі ігри жанру лонгбординг, такі як «Tanuki Sunset», «Turn your Longboard into a game controller» та «Skater-girl (2018)», було виявлено кілька суттєвих аспектів, що потребують покращення. Гра «Tanuki Sunset» вирізняється своїм цікавим стилем та широким спектром механік, проте її керування є занадто аркадним і нереалістичним для жанру лонгбордингу, а навколишнє середовище має занадто яскраві кольори, що може викликати негативні відчуття у гравців. Гра «Turn your Longboard into a game controller» має унікальну особливість використання власного лонгборду як контролера, що додає їй шарму, проте її візуальний дизайн є занадто спрощеним, що може бути як перевагою, так і недоліком, залежно від уподобань гравців. «Skater-girl (2018)» відзначається мінімалістичним дизайном, що дозволяє гравцям зосередитися на геймплеї, але гра потребує додаткових завдань та елементів, які б підвищили її реіграбельність та зробили її більш унікальною на ринку.

На основі цих висновків було сформульовано такі завдання дослідження:

Проектування програмного модуля комп’ютерної гри. Розробка архітектури програмного модуля для гри у жанрі лонгбординг, визначення основних компонентів гри та їх взаємодії, планування структури та логіки гри. Особлива увага буде приділена налаштуванню інтерфейсу та рівнів з акцентом на привабливий дизайн та гармонійні кольорові відтінки, щоб забезпечити візуально приємний ігровий досвід.

Реалізація комп’ютерної гри «Лонгбординг» на базі ігрового рушія Unity. Вибір та налаштування інструментів розробки для створення 3D гри, реалізація основних компонентів гри, включаючи моделі, анімації, фізику та механіку гри, забезпечення інтеграції всіх компонентів у єдину стабільну ігрову систему. Основна увага буде приділена реалістичності геймплею та відчуттю катання на лонгборді.

Тестування комп’ютерної гри. Проведення детального тестування гри для виявлення та виправлення помилок, забезпечення високої стабільності та якості користувацького досвіду.

Таким чином, дослідження спрямоване на створення якісного продукту у жанрі лонгбординг, що відповідатиме сучасним вимогам ринку комп'ютерних ігор та забезпечить користувачам захоплюючий та стабільний ігровий досвід.

1.4 Висновок

Було обгрунтовано доцільність розробки гри «Лонгбординг» на базі ігрового рушія Unity.

Було досліджено існуючі концепції ігор жанру лонгбординг. Ігри жанру лонгбординг зазвичай мають спрощене управління, що робить їх доступними для широкої аудиторії, включаючи дітей та дорослих. Зважено їх плюси та мінуси. Прикладом популярної гри цього жанру є Tanuki Sunset.

Було проведено аналіз ігрового рушія Unity та його доречність для реалізації гри «лонгбординг».

2 ПРОЕКТУВАННЯ ПРОГРАМНОГО МОДУЛЯ КОМП`ЮТЕРНОЇ ГРИ

2.1 Проектування інтерфейсу користувача

Інтерфейс користувача це графічні елементи, які забезпечують взаємодію користувача із застосунком. Певні вікна, меню, кнопки є найчастішими складовими елементами інтерфейсу. Основна мета розробників, коли йдеться про створення користувацького інтерфейсу – зробити його якомога функціональним, при цьому уникаючи втрати зрозумілості.

Оскільки єдиною платформою гри буде ПК, то до уваги було взято виключно схему керування за допомогою клавіатури та миші. В інструкції користувача буде зазначено, що гра не підтримує використання ігрового контролера.

У процесі проектування інтерфейсу виведено список схем, що використовуватимуться під час його фіналізації:

1. Головне меню – основне меню гри. Дане меню надаватиме доступ до наступного меню та виходу з гри. Схему головного меню зображено на рисунку 2.1
2. Меню вибору рівнів – дане меню буде відповідати за навігацію гравця між рівнями. Схему меню вибору рівнів зображено на рисунку 2.2.
3. Екран перемоги – на екрані перемоги буде зображено результати гравця. Схему екрану перемоги зображено на рисунку 2.3.
4. Екран поразки – під час геймплейного процесу також можлива поразка, після якої гравцю буде запропоновано перезапустити рівень, або повернутися до головного меню. Схему екрану поразки зображено на рисунку 2.4.

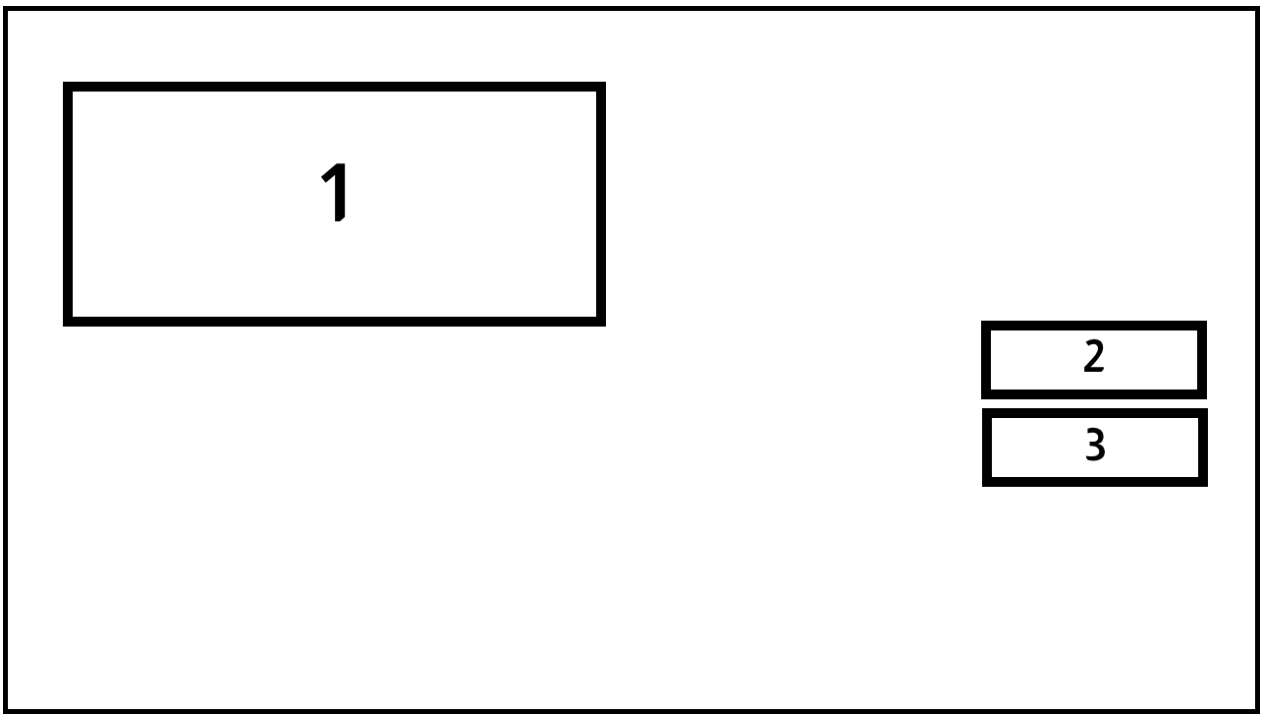

Рисунок 2.1 – Схема головного меню

На рисунку 2.1 відповідно:

1. Назва гри
2. Кнопка “Play”
3. Кнопка “Quit”

Максимально спрощене головне меню дозволить гравцеві легко орієнтуватись в ньому, що є дуже важливим для зручності користування та загального задоволення від гри. Таке спрощення сприяє тому, що навіть новачки зможуть швидко знайти потрібні функції і розпочати гру без затримок. Також, мінімалістичне головне меню виглядатиме більш стильно та довершено, додаючи грі сучасного вигляду і підкреслюючи її унікальний дизайн. Мінімалізм у дизайні може стати важливою особливістю, яка приверне увагу гравців, які цінують естетичну простоту і функціональність.

Також, невелика кількість елементів підкреслює можливість відтворення меню в середині ігрового світу для демонстрації графіки гри з першого запуску. Це означає, що гравець зможе відразу зануритись у атмосферу гри, не

відволікаючись на складні інтерфейси. Відтворення меню в середині ігрового світу може також допомогти показати графічні можливості гри та залучити гравця з перших секунд. Такий підхід дозволяє створити більш інтегроване і гармонійне ігрове середовище, де меню є частиною загального ігрового досвіду, а не окремим елементом.

Таким чином, максимально спрощене головне меню з мінімалістичним дизайном і додатковими анімаційними ефектами може значно покращити користувацький досвід, роблячи його більш зручним, стильним та інтерактивним. Відтворення меню в середині ігрового світу додатково підкреслює графічні можливості гри і допомагає гравцям швидше зануритись у її атмосферу.

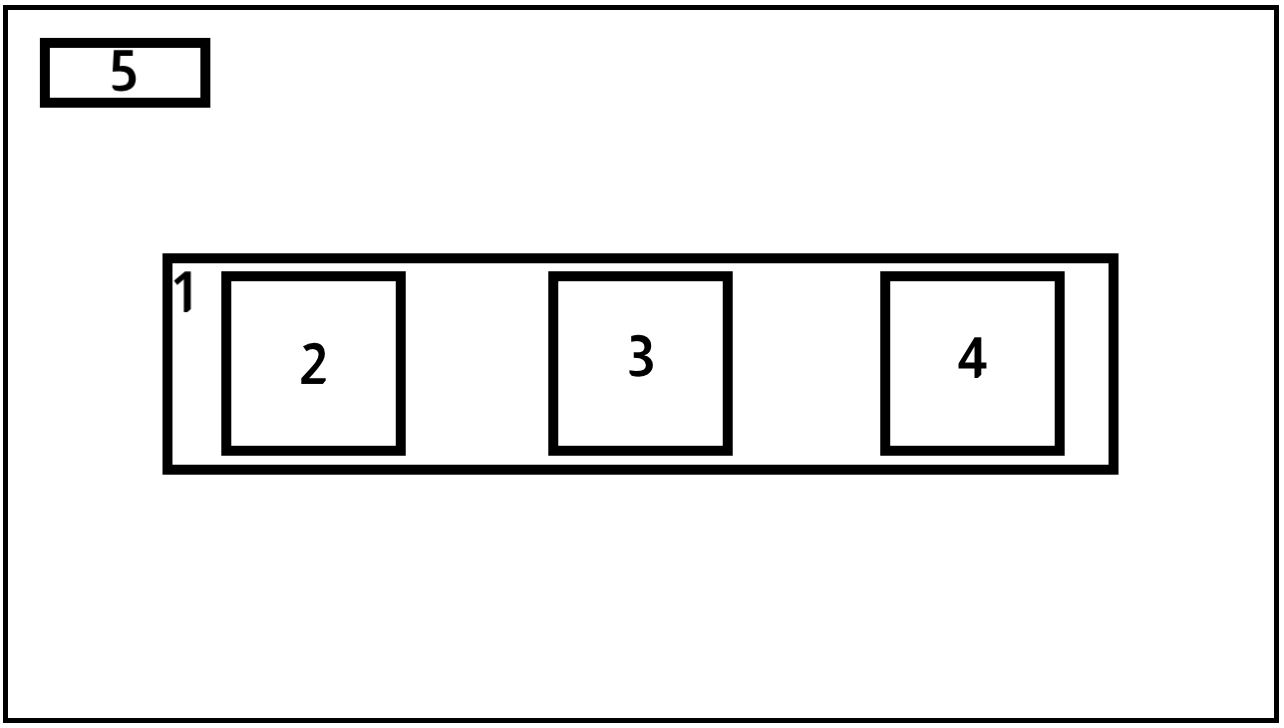

Рисунок 2.2. – Схема меню вибору рівнів

На рисунку 2.2 відповідно:

1. Панель рівнів
2. Кнопка першого рівня
3. Кнопка другого рівня
4. Кнопка третього рівня

5. Кнопка повернення до головного меню

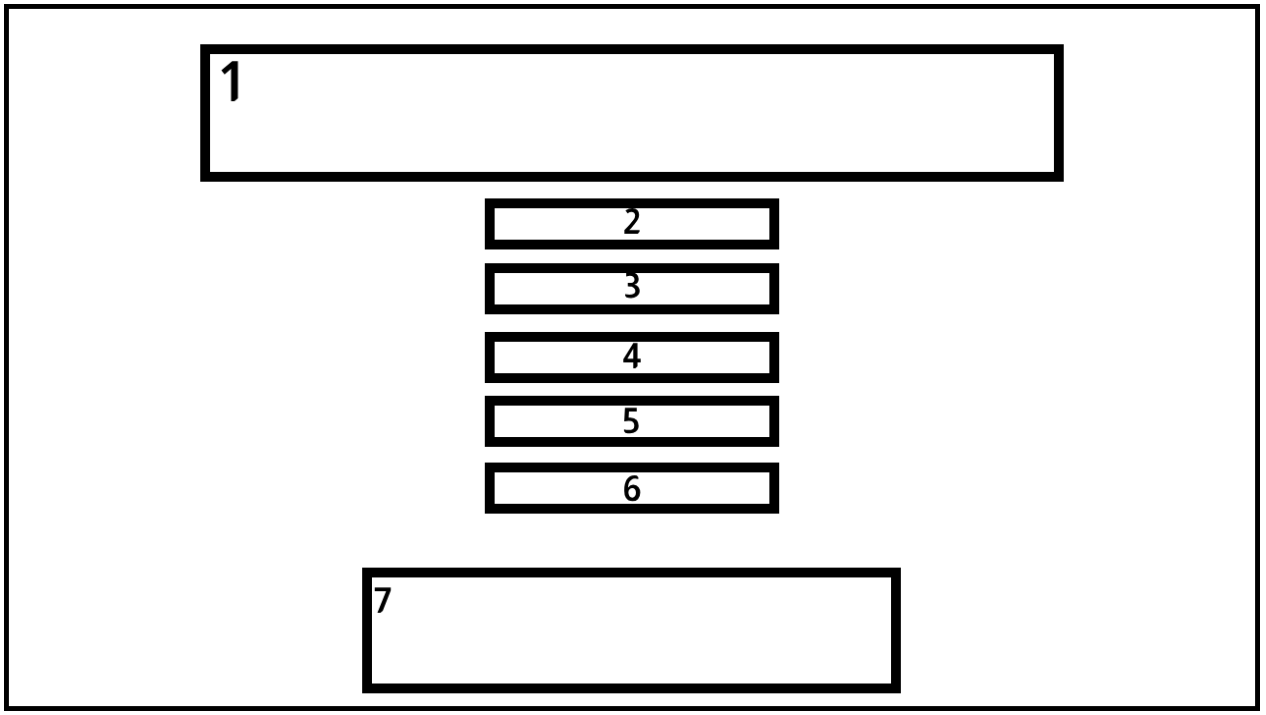

Рисунок 2.3. – Схема екрану перемоги

На рисунку 2.3 відповідно:

1. Напис “You’ve got a *назва медалі* medal!” / “No medal”
2. Час гравця
3. Час золотої медалі
4. Час срібної медалі
5. Час бронзової медалі
6. Найкращий час гравця
7. Напис для навігації далі

Виведення, яку саме медаль отримав користувач і часів, які необхідно побити для кожної медалі, підвищує реіграбельність гри шляхом надавання користувачеві мети і бажання покращити свій час, що в свою чергу збільшує довжину ігрової сесії. Такий підхід не тільки мотивує гравців намагатися досягти кращих результатів, але й створює змагальний елемент, який може залучити більше користувачів. Впровадження системи медалей також дозволяє гравцям

відчувати прогрес і досягнення, що є важливим аспектом для збереження інтересу до гри протягом тривалого часу. Крім того, додавання онлайн-таблиці лідерів, де користувачі можуть порівнювати свої результати з результатами інших гравців, ще більше підсилить конкурентний дух і стимулюватиме частіші повернення до гри.

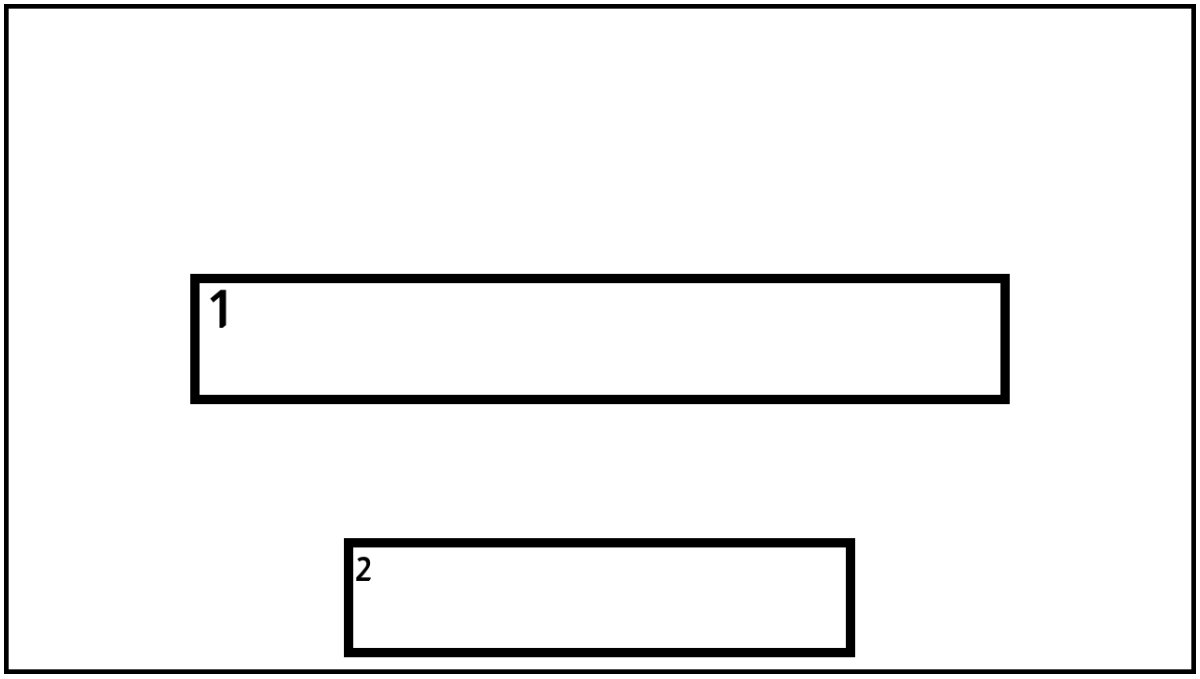

Рисунок 2.4. – Схема екрану поразки

На рисунку 2.4 відповідно:

1. Напис про поразку
2. Напис для навігації далі

2.2 Проектування алгоритму роботи програми

Алгоритм роботи 3D відеогри жанру “лонгбординг” починається з появи гравця на рівні. Гравець може вільно рухатися по рівню. Коли гравець перетинає тригер старт/фінішу, то запускається таймер, який відслідковує перетинання тригерів чекпоінтів. Головною небезпекою є контакт зі стіною при якому гравець програє.

Після перетину всіх чекпоінтів, гравець повинен перетнути тригер старт/фінішу, щоб закінчити рівень та отримати екран перемоги.

Блок-схему алгоритму роботи 3D відеогри жанру “лонгбординг” зображено на рисунку 2.5

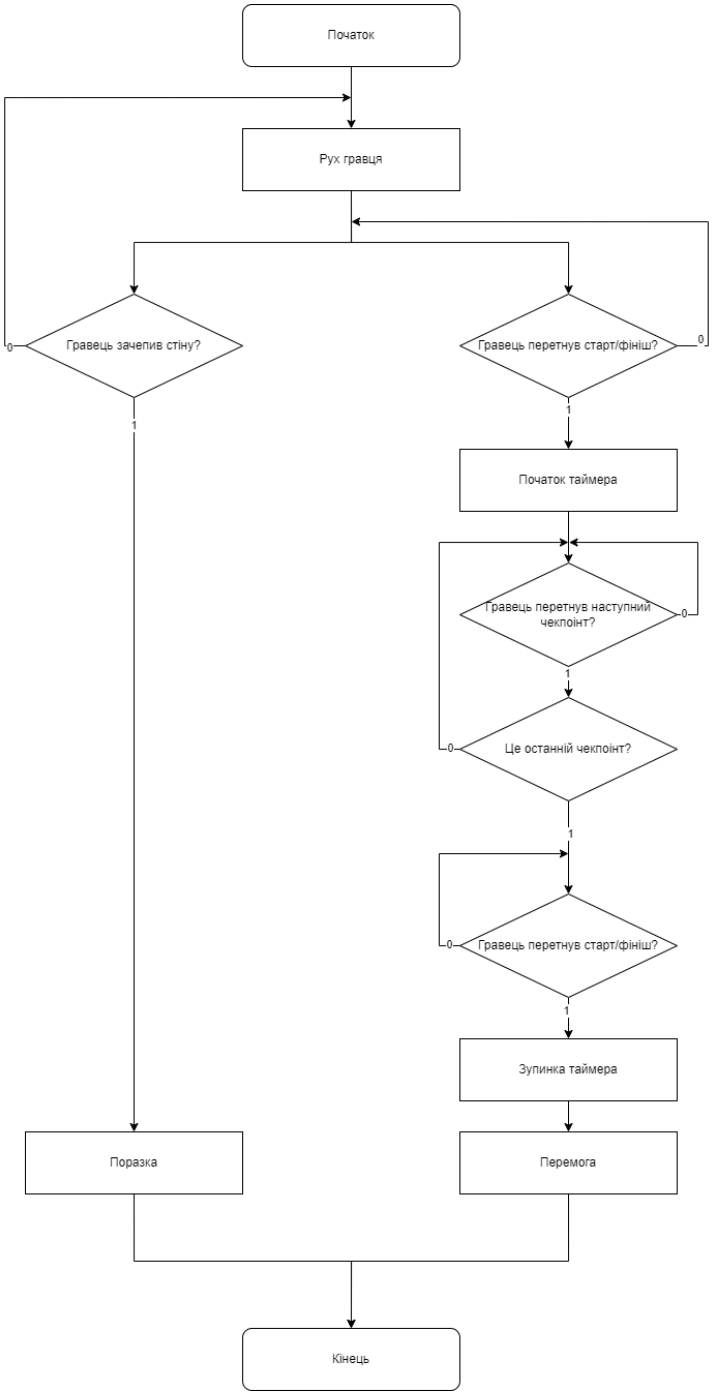

Рисунок 2.5. - Блок-схема алгоритму роботи 3D відеогри жанру “лонгбординг”

Функціональна модель персонажа зображена на рисунку 2.6.

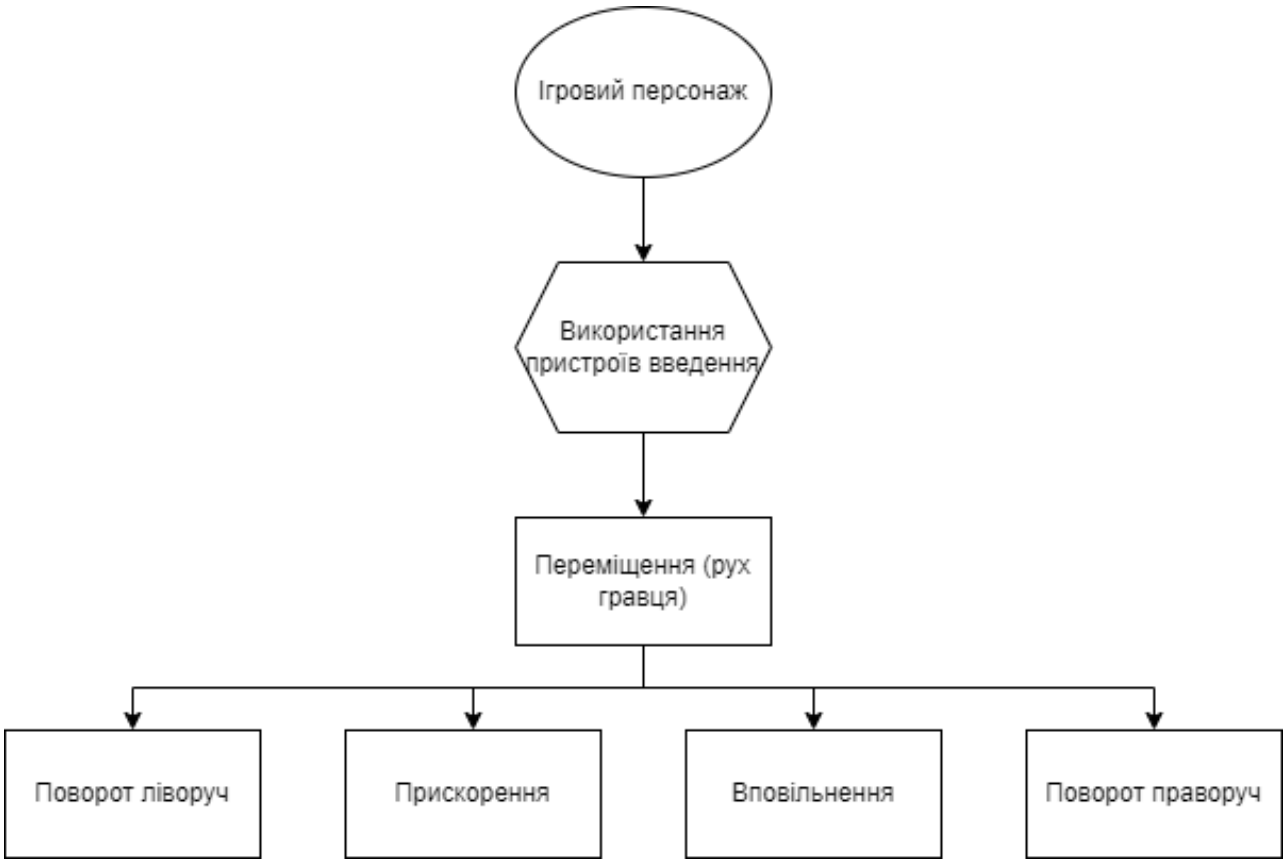

Рисунок 2.6 – Функціональна модель персонажа

На функціональній моделі персонажа виведено усі потенційні вибори, що надаються гравцю під час ігрової сесії, а саме:

1. Прискорення та вповільнення;
2. Повороти ліворуч та праворуч.

2.3 Проектування моделі реалізації ігрової системи

Після входу на ігрову сцену, модуль обробки контролю персонажем обмінюється даними з нею та модулем обробки дій гравця. Модуль обробки дій гравця надає необхідні дані аудіо модулю та обмінюється з модулем таймеру та модулем життя гравця. Модуль життя гравця надає дані ігровій сцені. Модуль таймеру надає дані ігровій сцені та модулю перемоги. Модуль перемоги надає дані ігровій сцені. На рисунку 2.7 зображено модель реалізації ігрової системи.

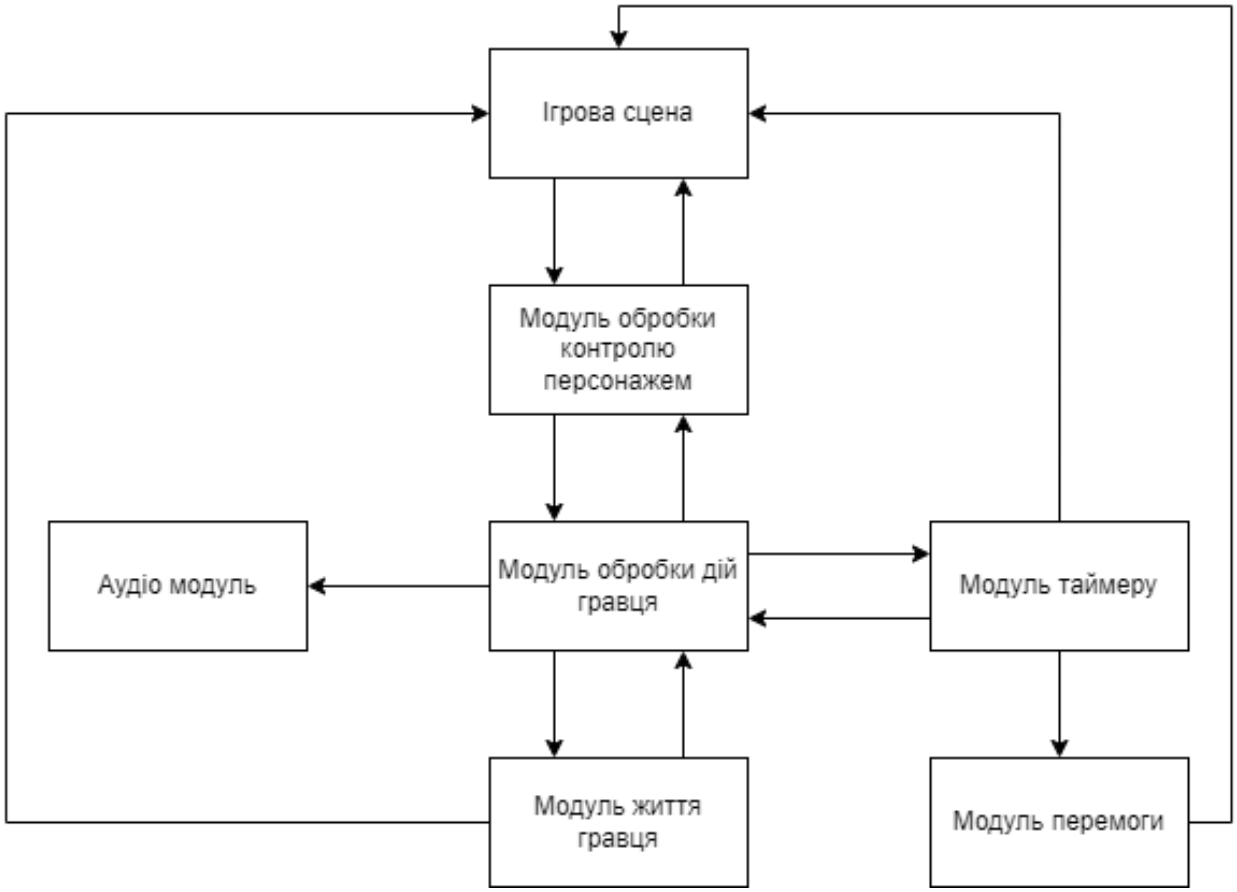

Рисунок 2.7 – Модель реалізації ігрової системи

2.4 Проектування ігрового циклу

Діаграма ігрового циклу є візуальним відображенням основного циклу функціонування ігри. Вона вказує послідовність та взаємодію між операціями, які відбуваються в грі від моменту запуску ігрової сесії до моменту її завершення [7].

Основна мета діаграми ігрового циклу - показати, як ігровий рушій опрацьовує введення від гравця, оновлює стан гри, отримує та передає дані ігровому світові та відображає оновлений стан на екрані. Ця діаграма необхідна для розуміння роботи гри та використовується усіма, хто працює над грою від розробників до технічних архітекторів для аналізу та вдосконалення ігрового процесу [7].

Зазвичай діаграма ігрового циклу включає такі етапи:

1. Перевірка вводу: Гра перевіряє будь-яке введення користувача. Найчастіше це натискання клавіш чи рух миші. Цей етап є критично важливим, оскільки дозволяє гравцеві взаємодіяти з грою в реальному часі. Введення користувача може включати також використання геймпада або інших пристроїв введення, що додає різноманіття можливостей для керування грою.

2. Оновлення стану гри: Гра оновлює стан всіх об'єктів у світі гри, таких як рухи гравця, відображення анімацій, фізичні обрахунки взаємодії тощо. Це означає, що кожен об'єкт у грі реагує на введення користувача та інші події, що відбуваються у грі. Наприклад, вороги можуть пересуватись у відповідь на дії гравця, а навколишнє середовище може змінюватись залежно від часу або подій у грі.

3. Зміна стану гри: Гра перевіряє умови для зміни стану гри, такі як колізія із перешкодою, виконання завдання тощо. На цьому етапі визначаються ключові події, які впливають на розвиток гри. Наприклад, якщо гравець зіштовхується з ворогом, гра може зменшити кількість його здоров'я або, якщо гравець досягає певної точки на карті, це може викликати початок нової місії чи рівня.

4. Відображення графіки: Гра оновлює графічне виведення на екрані, відображаючи оновлений стан гри гравцеві. Цей етап включає рендеринг всіх об'єктів і сцен, щоб гравець міг бачити результати своїх дій і поточний стан гри. Це може включати оновлення анімацій, зміну освітлення, відображення нових елементів інтерфейсу тощо.

Діаграма ігрового циклу - це важливий інструмент для розробників, який допомагає їм краще розуміти та керувати роботою гри. Вона представляє собою візуальне зображення основних етапів і процесів, які відбуваються під час виконання гри, включаючи ініціалізацію, обробку введення, оновлення стану гри та рендеринг графіки. Завдяки діаграмі ігрового циклу, розробники можуть легко ідентифікувати вузькі місця, оптимізувати продуктивність та забезпечити плавний ігровий досвід для гравців. Це особливо важливо в умовах складних сучасних ігор, де координація між різними компонентами та системами грає ключову роль. Зображення діаграми знаходиться на рисунку 2.8

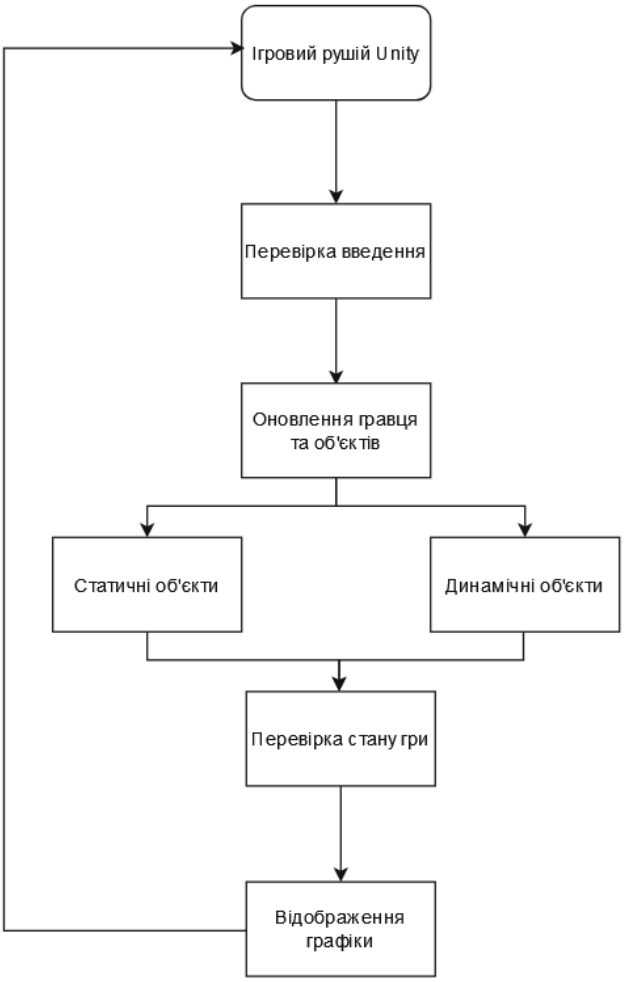

Рисунок 2.8 – Діаграма ігрового циклу

2.5 Висновок

У другому розділі дипломної роботи було спроектовано користувацький інтерфейс, розроблено алгоритм функціонування гри, спроектовано ігровий цикл та створено детальну модель реалізації ігрової системи для комп’ютерної гри «Лонгбординг».

Процес проектування користувацького інтерфейсу було почато з визначення ключових елементів, що забезпечують інтуїтивний доступ до функцій гри. На цьому етапі було створено макети інтерфейсу, які забезпечують зручну взаємодію користувача з грою.

Створення алгоритму роботи гри включало розробку механізмів обробки дій гравця та управління ігровими подіями. Основний алгоритм забезпечує стабільну роботу гри та високу продуктивність завдяки ефективним методам обробки даних та керування ресурсами. Він враховує різні ігрові сценарії, включаючи випадкові події та динамічні зміни під час гри [8].

Проектування ігрового циклу охоплювало всі необхідні стадії від ініціалізації гри до завершення сесії. Визначено ключові етапи, такі як початок гри, основна фаза, обробка зіткнень та завершення гри, що забезпечує цілісність ігрового процесу. Ігровий цикл розроблено так, щоб забезпечити плавний перехід між різними етапами гри та підтримувати інтерес гравця протягом усієї сесії.

Створення моделі реалізації ігрової системи включало розробку структурних компонентів гри, їх взаємодію та інтеграцію. Використання об'єктно-орієнтованого підходу до розробки гри дозволило створити гнучку та масштабовану систему. Це забезпечує можливість подальших модифікацій без необхідності повного перепроектування системи.

На цьому етапі також було проведено тестування окремих компонентів гри для забезпечення їх правильної роботи та інтеграції у загальну систему. Виявлені помилки були швидко виправлені, що значно підвищило стабільність та надійність ігрової системи.

Результати цього розділу створюють міцний фундамент для подальшої розробки та реалізації гри. Вони забезпечують високу якість кінцевого продукту та задоволення вимог користувачів, створюючи конкурентоспроможний продукт на ринку ігор. Запропоновані рішення дозволяють створити гру, яка буде цікавою та захоплюючою для широкого кола гравців, що, у свою чергу, сприятиме її популярності та успіху на ринку.

3 РОЗРОБКА КОМП'ЮТЕРНОЇ ГРИ «ЛОНГБОРДИНГ» НА БАЗІ ІГРОВОГО РУШІЯ UNITY

3.1 Аналіз мов програмування та IDE

C# був вибраний для даного проекту завдяки його численним перевагам у розробці ігор з використанням Unity. Оскільки Unity використовує C# як основну мову програмування, це забезпечує бездоганну інтеграцію з ігровим рушієм та доступ до потужних API, що значно спрощує реалізацію таких функцій, як фізика, анімація та управління введенням. Тому C# є найбільш оптимізованим та підтримуваним варіантом для роботи в Unity [9].

Синтаксис C# є простим і зрозумілим, що полегшує його освоєння навіть новачкам. Це дозволяє розробникам швидко розпочати роботу над проектом і зосередитися на функціональності гри. Строга типізація та підтримка об'єктно-орієнтованого програмування (ООП) допомагають писати структурований, читабельний та легкий у підтримці код.

Простота у вивченні та використанні: C# – це високорівнева мова програмування з багатою функціональністю та зручною синтаксичною структурою. Її легше вивчити, ніж багато інших мов, таких як C++ або Java, що робить її доступною для початківців. Завдяки строгій структурі та наявності автоматичного управління пам'яттю (garbage collection), C# дозволяє розробникам зосередитися на логіці гри, а не на технічних деталях управління ресурсами.

Портативність і кросплатформеність: C# дозволяє створювати кросплатформені ігри, які можуть бути запущені на різних пристроях і операційних системах. Завдяки Unity, гри, написані на C#, можуть бути експортовані на платформи, такі як Windows, macOS, Linux, iOS, Android, і навіть на консолі, такі як PlayStation, Xbox та Nintendo Switch. Це дає

можливість охопити широку аудиторію гравців без необхідності переписувати код для кожної платформи.

Висока продуктивність: C# забезпечує високу продуктивність завдяки компіляції у проміжний код (IL), який виконується в середовищі .NET. Сучасні компілятори та середовища виконання (CLR) оптимізують виконання коду, забезпечуючи високу швидкість роботи ігрових додатків. Крім того, Unity надає можливість оптимізації гри на рівні графіки та фізики, що дозволяє досягти високої якості ігрового процесу навіть на пристроях з обмеженими ресурсами.

Широкі можливості бібліотек та фреймворків: C# має багату екосистему бібліотек та фреймворків, які можуть бути використані для розробки ігор. Крім Unity, існують інші популярні фреймворки, такі як MonoGame, які також підтримують C# і дозволяють створювати високоякісні ігрові додатки. Ці бібліотеки надають широкий спектр інструментів для роботи з графікою, звуком, мережею та іншими аспектами ігрового процесу, що значно полегшує розробку.

Підтримка об'єктно-орієнтованого програмування: C# підтримує об'єктно-орієнтоване програмування (ООП), що дозволяє розробникам використовувати принципи модульності, повторного використання коду та інкапсуляції. Це дуже корисно при розробці великих і складних ігор, де важливо зберігати код чистим і організованим. ООП також полегшує командну роботу, оскільки різні частини гри можуть бути розроблені окремими командами, а потім інтегровані у єдиний проект.

Підтримка асинхронного програмування: C# має вбудовану підтримку асинхронного програмування за допомогою ключових слів async і await. Це дозволяє ефективно управляти асинхронними операціями, такими як завантаження ресурсів, мережеві запити та інші операції вводу-виводу, без блокування головного потоку гри. Завдяки цьому можна створювати більш чуйні та плавні ігрові додатки.

Потужне середовище розробки: Розробка ігор на C# зазвичай здійснюється за допомогою середовища розробки Microsoft Visual Studio або Rider від

JetBrains. Ці IDE надають широкий спектр інструментів для написання, налагодження та тестування коду. Вони підтримують інтеграцію з Unity, що робить процес розробки більш зручним та ефективним. Наявність таких інструментів, як IntelliSense, допомагає розробникам швидко писати код та зменшувати кількість помилок.

Інтегроване середовище розробки (IDE) є важливим інструментом, на якому розробники працюють для написання коду. IDE забезпечують багатофункціональні текстові редактори з можливостями налаштування плагінів, встановлення необхідних бібліотек, налаштування гарячих клавіш тощо. Такі можливості програмного середовища допомагають швидко створювати програмні продукти. Для розробки гри були обрані Visual Studio Code та Unity Hub.

Visual Studio Code (VS Code) – це безкоштовний, легкий та розширюваний редактор коду, розроблений Microsoft. Він широко використовується як для фронтенд, так і для бекенд розробки завдяки своїм потужним функціональним можливостям. [10] Вікно редактору VS Code показано на рисунку 1.4.

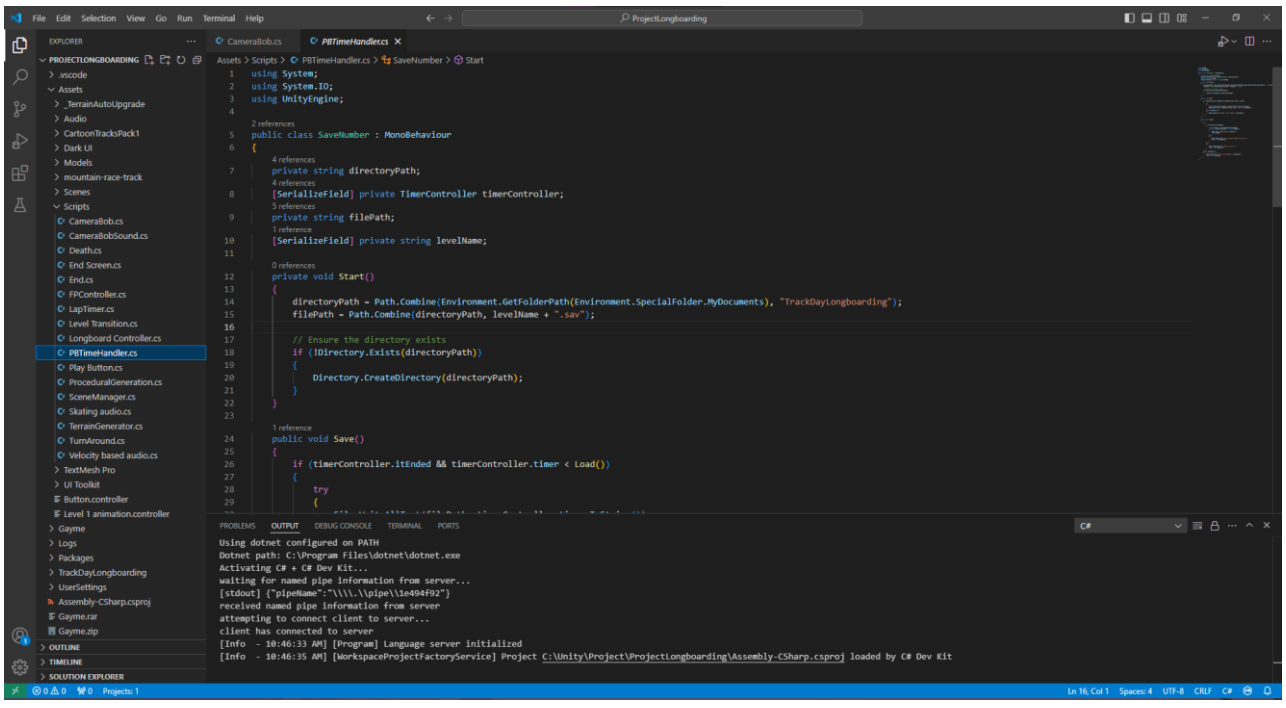

Рисунок 3.1 – Загальний вигляд інтерфейсного вікна програмного продукту VS Code

Основні переваги Visual Studio Code:

Розширюваність: Visual Studio Code підтримує великий набір розширень, які дозволяють розробникам додавати функціонал за потребою. Від підтримки різних мов програмування до інтеграції з системами контролю версій, як-от Git, VS Code легко адаптується під будь-які завдання.

Вбудований термінал: VS Code має вбудований термінал, що дозволяє розробникам виконувати команди безпосередньо з IDE, що значно полегшує робочий процес.

IntelliSense: Ця функція забезпечує інтелектуальну автодоповнення коду, що включає завершення символів, підказки функцій та рефакторинг, що значно підвищує продуктивність.

Debugging: VS Code має вбудовані інструменти для налагодження коду, що дозволяє ефективно знаходити та виправляти помилки.

Unity Hub – це офіційний менеджер проектів від Unity Technologies, який використовується для організації, управління та запуску проектів в Unity. [11] Вікно редактору Unity Hub показано на рисунку 3.2.

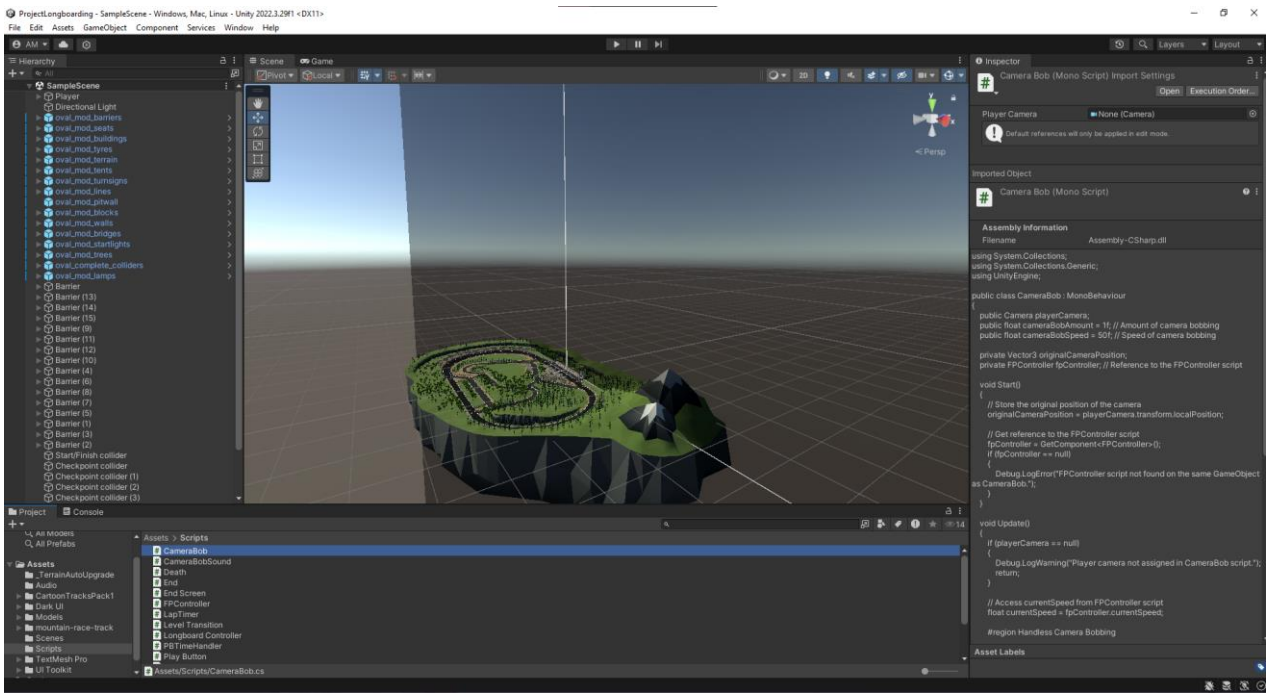

Рисунок 3.2 – Загальний вигляд інтерфейсного вікна
програмного продукту Unity Hub

Основні переваги Unity Hub:

Управління проектами: Unity Hub дозволяє легко створювати, відкривати та організовувати проекти. Це зручно для розробників, які працюють над кількома проектами одночасно.

Інсталяція версій Unity: Unity Hub дозволяє встановлювати та керувати кількома версіями Unity, що важливо для сумісності різних проектів з різними версіями рушія.

Доступ до документації та навчальних матеріалів: Unity Hub забезпечує швидкий доступ до документації, туторіалів та навчальних матеріалів, що полегшує процес навчання та освоєння нових функцій Unity.

Командна робота: Unity Hub підтримує інструменти для колаборації, такі як Unity Teams, що дозволяють розробникам спільно працювати над проектами, синхронізувати зміни та зберігати версії проектів у хмарі.

Інтеграція з Visual Studio: Unity Hub інтегрується з Visual Studio та Visual Studio Code, що дозволяє використовувати їх як основні редактори коду для проектів Unity, забезпечуючи єдиний робочий процес.

3.2 Розробка 3D моделей

На рівні з алгоритмами та механіками застосунку, графіка є основним компонентом взаємодії користувача з продуктом. Для того, щоб гравець був зацікавлений в продовженні гри, моделі і текстури повинні бути візуально привабливі.

У процесі розробки гри було прийнято рішення використовувати безкоштовні моделі з сайтів sketchfab.com та assetstore.unity.com.

На рисунку 3.3 наведено головний набір асетів, а саме – «Cartoon Race Track – Oval» [11].

Рисунок 3.3 - Cartoon Race Track – Oval

Також було використано такі моделі: Arcadia Longboard [12] та Longboard texturing practice [13].

Для створення різних рівнів було прийнято рішення використовувати той самий трек, обмежуючи пересування гравців за допомогою стін з покришок, що створює різні карти з одного гігантського треку. Такий підхід дозволяє розробникам економити ресурси та час на створення нових треків, одночасно забезпечуючи гравцям відчуття новизни та різноманіття. Використання покришок як обмежень дозволяє легко змінювати конфігурацію трас, створюючи нові виклики для гравців на кожному рівні. Вигляд повного треку зверху зображено на рисунку 3.4, а вигляд першого рівня з обмеженнями у вигляді чорних та жовтих покришок зображено на рисунку 3.5. Це рішення також додає грі елемент пазла, оскільки гравцям потрібно знайти найшвидший шлях через обмежену трасу, що додатково стимулює інтерес і залученість у процес гри.

Подібний дизайн дозволяє гравцям знайомитися з базовим треком і поступово досліджувати його різні варіації, що сприяє розвитку навичок та стратегічного мислення. Введення нових обмежень на кожному рівні зберігає гру свіжою та цікавою, змушуючи гравців адаптуватися до нових умов та випробовувати свої здібності у нових ситуаціях.

Рисунок 3.4 - Зображення повного треку.

Рисунок 3.5 – Зображення 1-го рівня

3.3 Розробка користувацького інтерфейсу

Базуючись на схемах користувацького інтерфейсу, які були спроектовані у розділі 2, було створено ігрові меню за допомогою вбудованих інструментів

ігрового рушія Unity. На рисунку 3.6 зображено розроблене головне меню комп’ютерної гри [14].

Рисунок 3.6 – Ігрове меню розроблене на основі спроектованих моделей

На рисунку 3.7 зображене меню вибору рівнів, створене на основі схем користувацького інтерфейсу. Кожен рівень також відображено зі схемою траси, щоб надати користувачеві більше інформації та спростити навігацію по рівню. Це рішення дозволяє гравцям краще зрозуміти структуру треку перед початком гри, що допомагає ефективніше планувати свої дії. Крім того, меню вибору рівнів може містити додаткові інформаційні елементи, такі як складність рівня, рекомендований час проходження і досягнення, які можна отримати на цьому рівні. Така детальна інформація допомагає гравцям оцінити свої можливості та обрати рівень, який відповідає їхнім навичкам і очікуванням.

Наявність візуальних схем трас у меню вибору рівнів робить процес вибору інтуїтивно зрозумілим і зручним, а також підвищує загальний користувацький досвід.

Рисунок 3.7 – Меню вибору рівнів

На рисунку 3.8 зображено екран перемоги, розроблений на основі спроектованих у 2-му розділі схем.

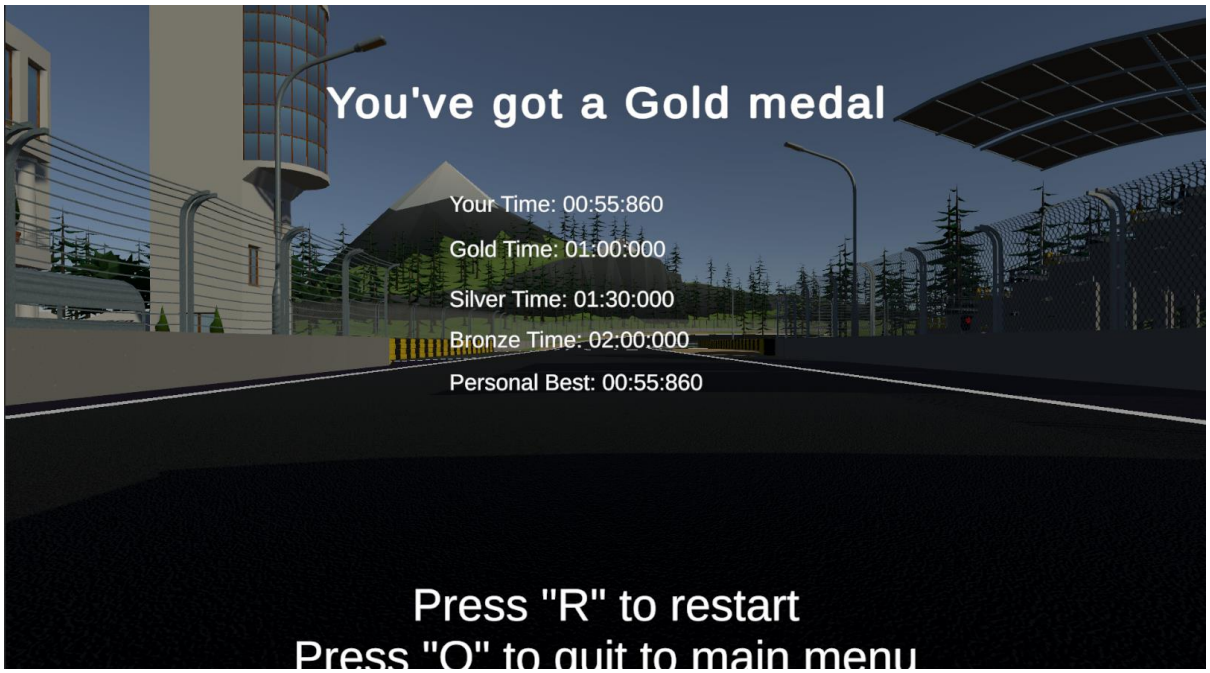

Рисунок 3.8 – Екран перемоги, розроблений на основі спроектованих у 2 розділі моделей.

На рисунку 3.9 зображено екран поразки, розроблене на основі спроектованих у 2 розділі схем.

Рисунок 3.9 – Екран поразки, розроблене на основі спроектованих у 2 розділі моделей.

3.4 Розробка основних модулів додатку

Основним модулем комп’ютерної гри «лонгбординг» на базі ігрового рушія Unity є клас FPController, який відповідає за керування персонажем, та його взаємодію з ігровим світом.

Рух персонажа відбувається за рахунок кнопок на клавіатурі. Реалізація руху відбувається у методі Update. Різні частини функціоналу руху розділені регіонами у коді. Частини коду, що реалізовують рух гравця наведено на рисунках 3.10, 3.11 та 3.12.

```
#region Handles Movement
Vector3 forward = transform.forward;

if (canMove && !timerController.itEnded)
{
    if (Input.GetAxis("Vertical") > 0)
        currentSpeed = Mathf.Min(currentSpeed + Time.deltaTime * acceleration, maxSpeed);
    else if (Input.GetAxis("Vertical") < 0)
        currentSpeed = Mathf.Max(currentSpeed - Time.deltaTime * acceleration, 0f);
}
else if (!timerController.itEnded && currentSpeed < 0.01)
{
    currentSpeed = 0f;
}

movementDirectionY -= gravity * Time.deltaTime;
moveDirection = forward * currentSpeed + new Vector3(0, movementDirectionY, 0);
if (currentSpeed >= 0.01)
{
    currentSpeed = currentSpeed - 0.01f;
}
characterController.Move(moveDirection * Time.deltaTime);
#endregion
```

Рисунок 3.10 – Регіон Handles Movement, який відповідає за переміщення гравця у просторі

```
#region Handles Camera Bobbing
if (currentSpeed > 0 && Input.GetAxis("Vertical") != 0 && currentSpeed < maxSpeed && canMove && !timerController.itEnded)
{
    bobbingOffset = -Mathf.Sin(Time.time * cameraBobSpeed) * cameraBobAmount;
    if (bobbingOffset < 0)
    {
        bobbingOffset *= 6f;
        isDescending = true;
    }
    else
    {
        isDescending = false;
        lowestBobbingOffset = float.MaxValue;
    }
    Vector3 targetCameraPosition = originalCameraPosition + new Vector3(0, bobbingOffset, 0);
    playerCamera.transform.localPosition = targetCameraPosition;

    if (isDescending)
    {
        lowestBobbingOffset = Mathf.Min(lowestBobbingOffset, bobbingOffset);
    }

    if (isDescending && bobbingOffset <= lowestBobbingOffset)
    {
        audioSource.Play();
    }
}
else
{
    playerCamera.transform.localPosition = Vector3.Lerp(playerCamera.transform.localPosition, originalCameraPosition, Time.deltaTime * 5f);
    lowestBobbingOffset = float.MaxValue;
    isDescending = false;
}
#endregion
```

Рисунок 3.11 – Регіон Handles Camera Bobbing, який відповідає за покачування камери для симуляції відштовхування від землі

```
#region Handles Rotation and Tilt
if (canMove && !timerController.itEnded)
{
    // Handle vertical rotation
    rotationX += -Input.GetAxis("Mouse Y") * lookSpeed;
    rotationX = Mathf.Clamp(rotationX, -lookXLimit, lookXLimit);

    // Gradual turning
    float turnInput = Input.GetAxis("Horizontal");
    if (turnInput != 0)
    {
        transform.rotation *= Quaternion.Euler(0, turnInput * turnSpeed * Time.deltaTime, 0);
    }

    // Camera tilt
    float targetTiltAngle = maxTiltAngle * -turnInput;
    Quaternion targetTiltRotation = Quaternion.Euler(0, 0, targetTiltAngle);

    // Combine tilt with vertical look rotation
    Quaternion currentCameraRotation = Quaternion.Euler(rotationX, playerCamera.transform.localRotation.eulerAngles.y, 0);
    playerCamera.transform.localRotation = Quaternion.Slerp(playerCamera.transform.localRotation, currentCameraRotation * targetTiltRotation, Time.deltaTime * tiltSpeed);
}
#endregion
```

Рисунок 3.12 – Регіон Handles Rotation and Tilt, який відповідає за повороти та нахил камери при повороті

У регіоні Handles Movement рух прописано так, що змінна currentSpeed – це швидкість, якої намагається увесь час досягнути моделька гравця. При затисканні клавіші “W” цільова швидкість збільшується, при затисканні клавіші “S” цільова швидкість навпаки зменшується. Щоб уникнути зворотнього руху, мінімальне значення швидкості гравця дорівнює нулю. Також, щоб уникнути нереалістичних швидкостей, максимальне значення дорівнює maxSpeed, яке легко змінити в ігровому редакторі і було визначене експереминтальним шляхом. Також, для симуляції тертя і вповільнення в реальному житті, цільова швидкість дуже повільно, але поступово зменшується до нуля.

З початком розробки, стало зрозуміло, що необхідно симулювати відштовхування при розгоні і зупинці, яке відбувається і у реальному житті, щоб зробити гру реалістичнішою. Регіон на рисунку 3.11 відповідає за погойдування камери, яке, власне, і є симуляцією цих відштовхувань. Камера більш різко опускається донизу і більш плавно підіймається догори. Також, щоб довершити симуляцію відштовхування, у цьому регіоні було додано програш звукового ефекту кроку тоді, коли камера досягає найнижчої позиції.

Регіон “Handles Rotation and Tilt” відповідає за поворот гравця. Код було написано таким чином, щоб з прискоренням гравця радіус повороту збільшувався, а камера нахилялась відповідно до куту повороту, щоб симулювати реалістичний поворот на лонгборді.

В ході розробки було вирішено, що створювати окремі колайдери для програшу гравця витратить занадто багато часу. В результаті, було розроблено інший метод визначення програшу гравця. За це відповідає клас SpeedMonitor, Update метод якого зображено на рисунку 3.13. Щоразу, коли гравець стикається з перешкодою, його реальна швидкість падає, тоді як currentSpeed з класу FPController залишається незмінною. Тож, в даному класі було додано змінну actualSpeed, яка отримує дані з ігрової сцени про швидкість гравця у напрямку руху. Якщо реальна швидкість гравця менша на певне значення, ніж цільова швидкість, гравця буде зупинено і показано екран поразки.

```
0 references
void Update()
{
    if (fpController != null && characterController != null && !timerController.itEnded)
    {
        // Calculate the actual speed of the player using the CharacterController's velocity
        Vector3 horizontalVelocity = new Vector3(characterController.velocity.x, 0, characterController.velocity.z);
        actualSpeed = horizontalVelocity.magnitude;

        // Calculate the difference between the actual speed and the current speed
        float speedDifference = Mathf.Abs(actualSpeed - fpController.currentSpeed);

        // Check if the difference exceeds the threshold
        if (speedDifference > speedDifferenceThreshold)
        {
            // Stop the player by disabling movement
            fpController.canMove = false;
            fpController.currentSpeed = 0;
            deathSreen.SetActive(true);
            Debug.Log("Player stopped due to large speed difference. Actual speed: " + actualSpeed + ", Current speed: " + fpController.currentSpeed);
        }
    }
}
```

Рисунок 3.13 – Метод Update класу SpeedMonitor

Клас Skatingaudio відповідає за звук лонгборду. При первинній розробці даний клас програвав звук коліс лонгборду під час руху, але в ході тестування було прийнято рішення вдосконалити цей процес, впевнившись в тому, що швидкість звуку, що програється відповідає швидкості гравця. За допомогою функцій мапингу, змінюється швидкість звуку відповідно до швидкості гравця. Метод Update класу Skatingaudio (рис. 3.14) відповідає саме за це.

```
void Update()
{
    if (speedMonitor.actualSpeed > 0.01f)
    {
    float normalizedSpeed = Mathf.Clamp01(speedMonitor.actualSpeed / fPController.maxSpeed);
    float pitch = Mathf.Lerp(0.5f, 1.0f, normalizedSpeed);
    audioSource.pitch = pitch;
    }
    else
    {
        audioSource.pitch = 0;
    }
}
```

Рисунок 3.14 – Метод Update класу Skatingaudio

Клас TimerController відповідає за таймер після старту і до фінішу. Одразу було очевидно, що без чекпоінтів гравці просто перетинатимуть старт/фініш двічі за рахунок розвороту, що викликало б моментальну зупинку таймера і фінальний екран. Було прийнято рішення додати чекпоінти по усьому треку, щоб впевнитись у вірному проходженні рівня гравцем. Також було розроблено систему, яка відслідковує послідовність отримання чекпоінтів, щоб запобігти «обману» гри шляхом перетинання одного чекпоінта кілька разів. Клас TimerController має такі методи:

1. Update (рис. 3.15) відповідає за таймер та його відображення на екрані гравця.
2. OnTriggerEnter (рис. 3.16) відповідає за відслідковування колізії гравця з чекпоінтами або старт/фінішною лінією.
3. StartTimer і StopTimer (рис. 3.17) відповідають за зміну статусу таймеру.
4. GetNextCheckpointTransform (рис. 3.18) відповідає за знаходження наступного чекпоінту.
5. FormatTime (рис. 3.19) відповідає за форматування часу.
6. UpdateTimerDisplay (рис. 3.20) відповідає за оновлення таймеру на екрані.

```
0 references
void Update()
{
    if (timerRunning)
    {
        timer += Time.deltaTime;
        overlay.SetActive(true);
        UpdateTimerDisplay();
    }

    if (!timerRunning)
    {
        overlay.SetActive(false);
    }
}
```

Рисунок 3.15 – Метод Update класу TimerController

```
void OnTriggerEnter(Collider other)
{
    // Check if the player collides with a checkpoint
    if (checkpoints.Contains(other))
    {
        int checkpointIndex = checkpoints.IndexOf(other);
        if (checkpointIndex == currentCheckpointIndex + 1)
        {
            currentCheckpointIndex = checkpointIndex;
            Debug.Log("Checkpoint " + (currentCheckpointIndex + 1) + " reached!");
        }
        else if (checkpointIndex != currentCheckpointIndex + 1)
        {
            Debug.Log("Cheating detected! Player skipped checkpoints.");
            cheated = true;
        }
        // Else, player revisited previous checkpoint, no action needed
    }
    // Check if the player collides with the start/finish line
    else if (other == startFinishCollider)
    {
        if (!timerRunning)
        {
            StartTimer();
        }
        else if (!cheated && currentCheckpointIndex > 1)
        {
            StopTimer();
        }
    }
}
```

Рисунок 3.16 – Метод OnTriggerEnter класу TimerController

```
1 reference
void StartTimer()
{
    timerRunning = true;
    Debug.Log("Timer started!");
}

1 reference
void StopTimer()
{
    Debug.Log("Timer stopped!");
    itEnded = true;
    saveNumber.Save();
    timerRunning = false;
}
```

Рисунок 3.17 – Методи StartTimer і StopTimer класу TimerController

```
1 reference
public Transform GetNextCheckpointTransform()
{
    if (currentCheckpointIndex < checkpoints.Count - 1)
    {
        return checkpoints[currentCheckpointIndex + 1].transform;
    }
    else
    {
        return null; // If there are no more checkpoints, return null
    }
}
```

Рисунок 3.18 – Метод GetNextCheckpoinTransform класу TimerController

```
private string FormatTime(float timeInSeconds)
{
    int minutes = Mathf.FloorToInt(timeInSeconds / 60);
    int seconds = Mathf.FloorToInt(timeInSeconds % 60);
    int milliseconds = Mathf.FloorToInt((timeInSeconds - Mathf.Floor(timeInSeconds)) * 1000);
    return string.Format("{0:00}:{1:00}:{2:000}", minutes, seconds, milliseconds);
}
```

Рисунок 3.19 – Метод FormatTime класу TimerController

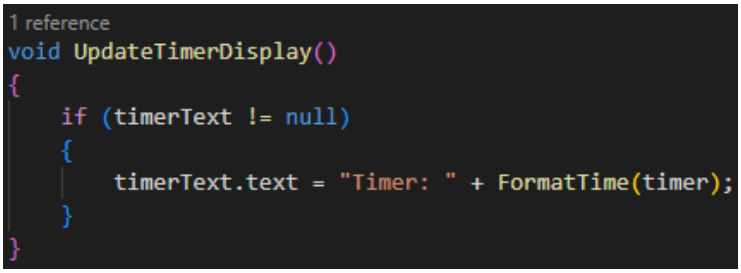

Рисунок 3.20 – Метод UpdateTimerDisplay класу TimerController

Клас EndScreenController, частина якого зображена на рисунку 3.21, відповідає за екран перемоги гравця. Він відображає час гравця, час медалей і найкращий час гравця за всі спроби. Крім того, екран перемоги може включати додаткові елементи, такі як повідомлення про досягнення, графічні елементи для підвищення візуальної привабливості та кнопки для переходу до головного меню або початку нової гри.

Цей клас функціонує в тандемі з класом SaveNumber, який відповідає за збереження найкращого результату гравця у спеціальний файл з розширенням «.sav». SaveNumber обирає найкращий час гравця шляхом порівняння останнього часу і часу записаного в файлі. Якщо файлу не існує, він створюється і туди записується останній час гравця. Такий підхід гарантує, що прогрес гравця завжди буде збережений, навіть якщо гра буде перезапущена. Це забезпечує додаткову мотивацію для гравців намагатися покращити свої результати, знаючи, що їхні досягнення будуть збережені.

Методи Start, Save і Load класу SaveNumber зображені на рисунках 3.22, 3.23 і 3.24 відповідно. Метод Start відповідає за ініціалізацію необхідних параметрів на початку, метод Save забезпечує збереження найкращого часу у файл, а метод Load відповідає за завантаження збережених даних при запуску гри. Всі ці методи разом забезпечують стабільну і надійну роботу механізму збереження і завантаження прогресу гравця, що є важливою складовою для забезпечення позитивного користувацького досвіду.

Крім того, клас EndScreenController може містити методи для аналізу продуктивності гравця, такі як обчислення середнього часу проходження рівнів або підрахунок кількості спроб, необхідних для досягнення певного результату. Ці дані можуть відображатися на екрані перемоги, надаючи гравцям детальнішу зворотну інформацію про їхню гру. Такий функціонал сприяє підвищенню зацікавленості гравців у поліпшенні своїх навичок і надає додатковий стимул для подальших спроб.

```
0 references
void Update()
{
    if (timerController.itEnded && !timerController.timerRunning)
    {
        playerTimeText.text = "Your Time: " + FormatTime(timerController.timer);
        goldTimeText.text = "Gold Time: " + FormatTime(goldTime);
        silverTimeText.text = "Silver Time: " + FormatTime(silverTime);
        bronzeTimeText.text = "Bronze Time: " + FormatTime(bronzeTime);
        if (saveNumber.Load() != float.MaxValue)
        {
            personalBestText.text = "Personal Best: " + FormatTime(saveNumber.Load());
        }
        endScreenCanvas.SetActive(true);
    }
}

5 references
private string FormatTime(float timeInSeconds)
{
    int minutes = Mathf.FloorToInt(timeInSeconds / 60);
    int seconds = Mathf.FloorToInt(timeInSeconds % 60);
    int milliseconds = Mathf.FloorToInt((timeInSeconds - Mathf.Floor(timeInSeconds)) * 1000);
    return string.Format("{0:00}:{1:00}:{2:000}", minutes, seconds, milliseconds);
}
}
```

Рисунок 3.21 – Частина класу EndScreenController

```
0 references
private void Start()
{
    directoryPath = Path.Combine(Environment.GetFolderPath(Environment.SpecialFolder.MyDocuments), "TrackDayLongboarding");
    filePath = Path.Combine(directoryPath, levelName + ".sav");

    // Ensure the directory exists
    if (!Directory.Exists(directoryPath))
    {
        Directory.CreateDirectory(directoryPath);
    }
}
```

Рисунок 3.22 – Метод Start класу SaveNumber

```
1 reference
public void Save()
{
    if (timerController.itEnded && timerController.timer < Load())
    {
        try
        {
            File.WriteAllText(filePath, timerController.timer.ToString());
            Debug.Log($"Number {timerController.timer} saved to {filePath}");
        }
        catch (Exception e)
        {
            Debug.LogError($"Failed to save number: {e.Message}");
        }
    }
}
```

Рисунок 3.23 – Метод Save класу SaveNumber

```
3 references
public float Load()
{
    try
    {
        if (File.Exists(filePath))
        {
            string content = File.ReadAllText(filePath);
            if (float.TryParse(content, out float number))
            {
                Debug.Log($"Loaded number: {number}");
                return number;
            }
            else
            {
                Debug.LogWarning("Failed to parse number from file.");
                return float.MaxValue;
            }
        }
        else
        {
            Debug.LogWarning("File does not exist.");
            return float.MaxValue;
        }
    }
    catch (Exception e)
    {
        Debug.LogError($"Failed to load number: {e.Message}");
        return float.MaxValue;
    }
}
```

Рисунок 3.24 – Метод Load класу SaveNumber

Класи SceneTransition та MainMenu максимально схожі між собою. Функціонал класів полягає в плавному переміщенні камери всередині меню та

між рівнями кривою з визначеним кутом за допомогою кривої Безьє. Різниця між ними полягає в тому, що SceneTransition також відповідає за перехід на рівень в той час, як MainMenu лише переміщає камеру. Методи OnPlayButtonClicked та CalculateBezierPoint однакові в обох класах та зображені на рисунку 3.25 і 3.26 відповідно, метод SmoothTransitionToLevelSelection класу MainMenu (рис. 3.27) має на одну стрічку менше за такий самий метод класу SceneTransition (рис. 3.28). Ця стрічка відповідає за завантаження сцени рівня.

Окрім плавних переходів, класи SceneTransition та MainMenu можуть включати додаткові функції для покращення користувацького досвіду. Наприклад, клас SceneTransition може мати анімаційні ефекти, які з'являються при завантаженні рівня.

```
1 reference
private void OnPlayButtonClicked()
{
    StartCoroutine(SmoothTransitionToLevelSelection());
}
```

Рисунок 3.25 – Метод OnPlayButtonClicked класів SceneTransition та MainMenu

```
private Vector3 CalculateBezierPoint(Vector3 p0, Vector3 p1, Vector3 p2, float t)
{
    float u = 1f - t;
    float tt = t * t;
    float uu = u * u;

    Vector3 point = uu * p0;
    point += 2f * u * t * p1;
    point += tt * p2;

    return point;
}
```

Рисунок 3.26 – Метод CalculateBezierPoint класів SceneTransition та MainMenu

```
private IEnumerator SmoothTransitionToLevelSelection()
{
    float elapsedTime = 0f;

    Vector3 startTransitionPosition = mainMenuCamera.transform.position;
    Vector3 endTransitionPosition = levelSelectionCamera.transform.position;

    // Calculate control point for the Bezier curve
    Vector3 controlPoint = (startTransitionPosition + endTransitionPosition) / 2f;
    controlPoint += Vector3.up * arcHeight;

    transitionCamera.gameObject.SetActive(true);
    transitionCamera.transform.position = startTransitionPosition;
    transitionCamera.transform.rotation = mainMenuCamera.transform.rotation;

    mainMenuCamera.gameObject.SetActive(false);

    while (elapsedTime < transitionDuration)
    {
        elapsedTime += Time.deltaTime;
        float t = elapsedTime / transitionDuration;

        // Calculate position along the Bezier curve
        Vector3 bezierPosition = CalculateBezierPoint(startTransitionPosition, controlPoint, endTransitionPosition, t);

        // Smoothly interpolate the position
        transitionCamera.transform.position = bezierPosition;

        // Smoothly interpolate the rotation
        transitionCamera.transform.rotation = Quaternion.Slerp(mainMenuCamera.transform.rotation, levelSelectionCamera.transform.rotation, t);

        yield return null;
    }

    // Ensure the transition camera ends at the exact position and rotation of the level selection camera
    transitionCamera.transform.position = endTransitionPosition;
    transitionCamera.transform.rotation = levelSelectionCamera.transform.rotation;

    transitionCamera.gameObject.SetActive(false);
    levelSelectionCamera.gameObject.SetActive(true);
}
```

Рисунок 3.27 – Метод SmoothTransitionToLevelSelection класу MainMenu

```
1 reference
private IEnumerator SmoothTransitionToLevelSelection()
{
    float elapsedTime = 0f;

    Vector3 startTransitionPosition = mainMenuCamera.transform.position;
    Vector3 endTransitionPosition = levelSelectionCamera.transform.position;

    // Calculate control point for the Bezier curve
    Vector3 controlPoint = (startTransitionPosition + endTransitionPosition) / 2f;
    controlPoint += Vector3.up * arcHeight;

    transitionCamera.gameObject.SetActive(true);
    transitionCamera.transform.position = startTransitionPosition;
    transitionCamera.transform.rotation = mainMenuCamera.transform.rotation;

    mainMenuCamera.gameObject.SetActive(false);

    while (elapsedTime < transitionDuration)
    {
        elapsedTime += Time.deltaTime;
        float t = elapsedTime / transitionDuration;

        // Calculate position along the Bezier curve
        Vector3 bezierPosition = CalculateBezierPoint(startTransitionPosition, controlPoint, endTransitionPosition, t);

        // Smoothly interpolate the position
        transitionCamera.transform.position = bezierPosition;

        // Smoothly interpolate the rotation
        transitionCamera.transform.rotation = Quaternion.Slerp(mainMenuCamera.transform.rotation, levelSelectionCamera.transform.rotation, t);

        yield return null;
    }

    // Ensure the transition camera ends at the exact position and rotation of the level selection camera
    transitionCamera.transform.position = endTransitionPosition;
    transitionCamera.transform.rotation = levelSelectionCamera.transform.rotation;

    transitionCamera.gameObject.SetActive(false);
    levelSelectionCamera.gameObject.SetActive(true);
    SceneManager.LoadScene(levelSceneName);
}
```

Рисунок 3.28 – Метод SmoothTransitionToLevelSelection класу SceneTransition

3.5 Результати тестування програмного модуля комп’ютерної гри

Першим кроком тестування комп’ютерної гри «Лонгбординг» на базі ігрового рушія Unity є її запуск. Після запуску гравця зустрічає головне меню з кнопками «Почати гру» та «Вихід». Результат запуску зображено на рисунку 3.29. На рисунку чітко видно всі елементи головного меню, що дозволяє оцінити його функціональність та естетичний вигляд. Також можна перевірити коректність відображення всіх кнопок і елементів інтерфейсу, що є важливою частиною процесу тестування, щоб забезпечити зручність і комфорт для гравців.

Рисунок 3.29 – Загальний вигляд головного меню комп’ютерної гри

Після натискання кнопки «Почати гру» відбувається процес переносу камери до меню вибору рівнів. Результат цього перенесення зображено на рисунку 3.30.

Рисунок 3.30 – Загальний вигляд меню вибору рівнів комп’ютерної гри

Після переходу на рівень, розпочинається тестування геймплею. Для початку потрібно затиснути “W”, щоб перетнути старт/фініш, результатом чого буде меню ігрового процесу, яке включає в себе таймер. Важливо перевірити, чи правильно працює таймер, чи він запускається вчасно, коли гравець перетинає стартову лінію. Також необхідно перевірити, чи коректно відображається час на таймері під час гри, щоб переконатися в його точності. Результат початку гри зображено на рисунку 3.29. На цьому рисунку можна побачити, як виглядає ігровий інтерфейс під час активного геймплею, що дозволяє оцінити його зручність та функціональність. Це також дає можливість виявити потенційні помилки або недоліки, які можуть вплинути на ігровий досвід, і внести необхідні корективи для їх усунення.

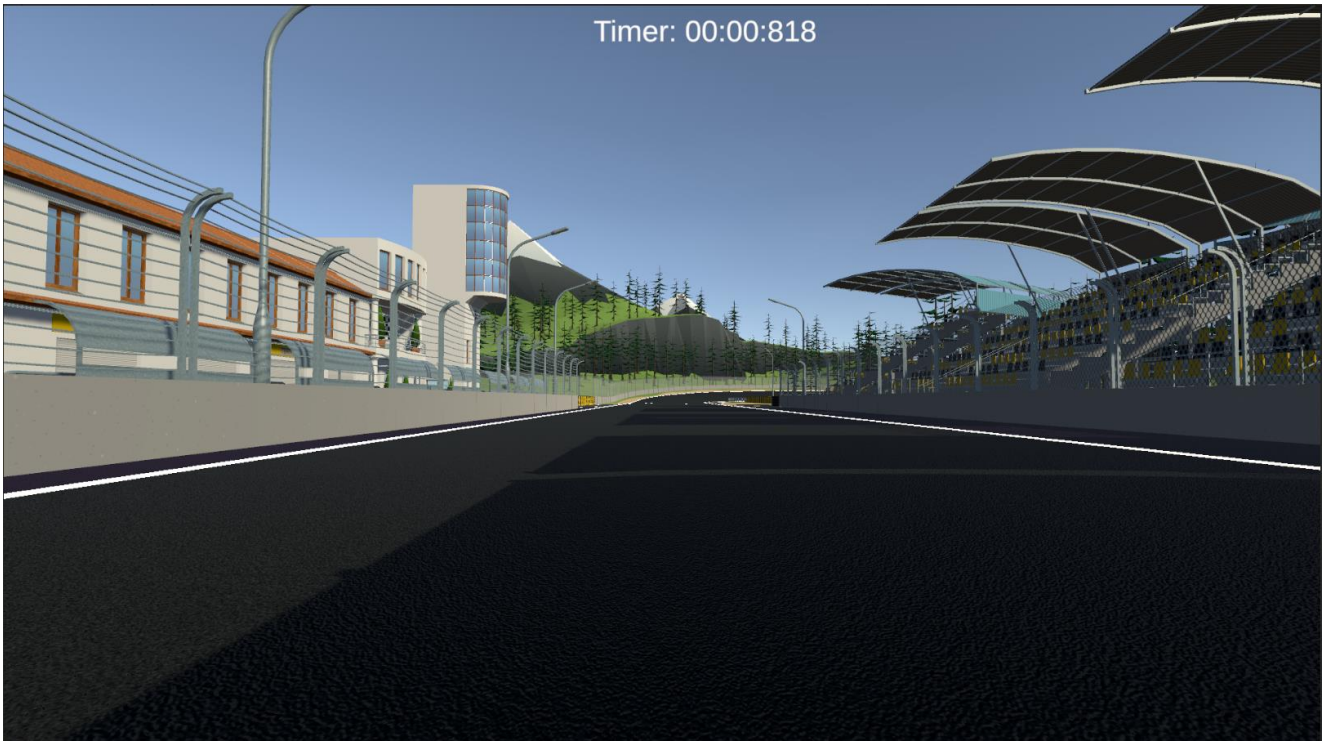

Рисунок 3.31 – Загальний вигляд інтерфейсу початку гри

Подальшим кроком буде тестування повноцінного руху персонажа. Керування лонгбордом здійснюється за допомогою клавіш WASD, що є стандартним для багатьох ігор, забезпечуючи зручність та інтуїтивне розуміння для гравців. Під час тестування необхідно перевірити, чи персонаж коректно реагує на натискання кожної з цих клавіш. Наприклад, при натисканні “W” персонаж повинен рухатись вперед, при натисканні “S” - назад, “A” і “D” відповідають за рух вліво і вправо відповідно.

Важливо перевірити плавність і чутливість керування, щоб переконатися, що персонаж рухається без затримок і різких ривків. Це включає тестування різних сценаріїв, таких як різка зміна напрямку, рух по діагоналі, а також перевірка того, як персонаж реагує на команди при різних швидкостях. Плавність анімацій і відповідність рухів реалістичній фізиці також мають велике значення для забезпечення захоплюючого і реалістичного ігрового досвіду.

Особливу увагу варто приділити тестуванню анімації руху персонажа, щоб переконатися, що всі рухи виконуються плавно і природно. Це включає синхронізацію рухів тіла з рухом лонгборду, а також перевірку правильності

відображення тіні і взаємодії персонажа з поверхнею. Необхідно також переконатися, що анімація відповідає швидкості руху та змінюється відповідно до різних дій, таких як прискорення або гальмування.

Результат натискання на кнопки руху персонажа зображено на рисунках 3.32 та 3.33. На рисунках можна побачити, як персонаж змінює своє положення у просторі у відповідь на натискання відповідних клавіш. Це дозволяє оцінити, наскільки точно і коректно працює система управління. Крім того, на рисунках можна проаналізувати, як відбувається взаємодія персонажа з оточенням: чи проходить він крізь об'єкти, чи правильно реагує на колізії, чи немає будь-яких графічних артефактів або глюків.

Тестування також включає перевірку того, як персонаж рухається по різним поверхням і ухилам. Наприклад, необхідно переконатися, що персонаж може правильно підніматись на схили та спускатися з них, що його швидкість і контроль над лонгбордом реалістично змінюються залежно від нахилу поверхні.

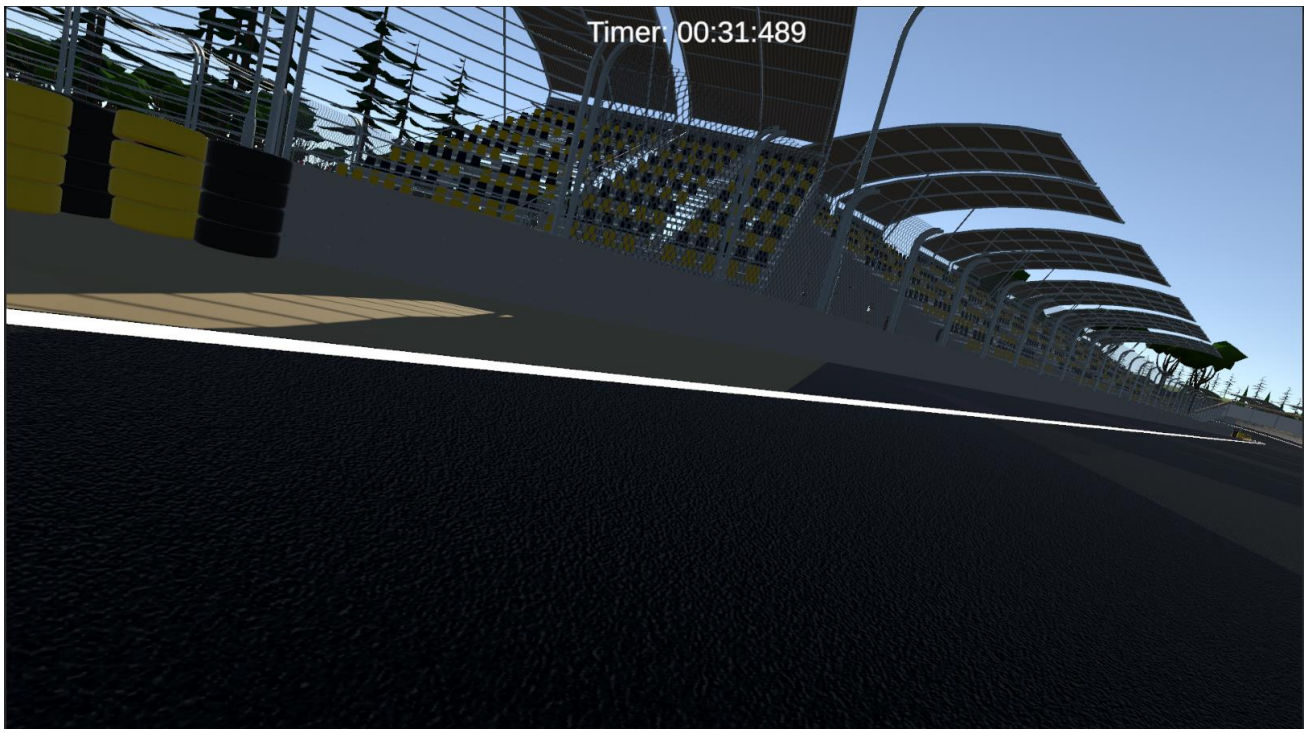

Рисунок 3.32 – Загальний вигляд інтерфейсу при повороті наліво

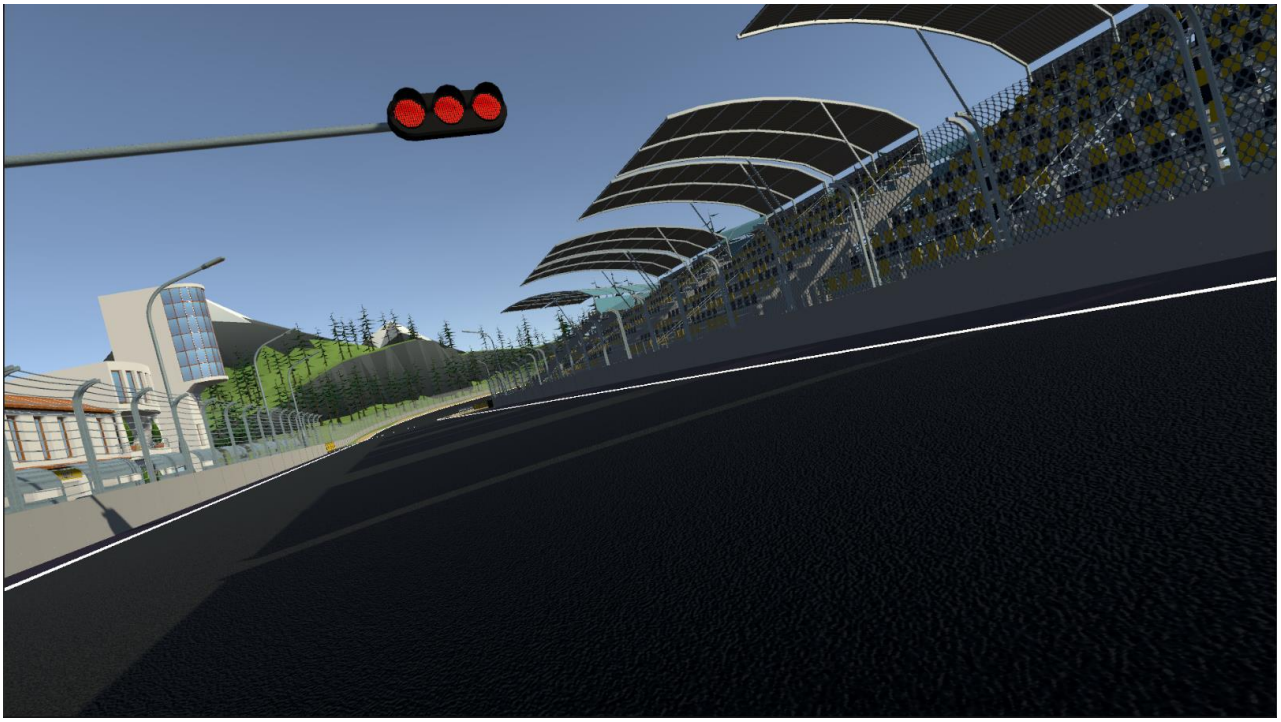

Рисунок 3.33 – Загальний вигляд інтерфейсу при повороті направо

Гравець має змогу взаємодіяти з перешкодами та стінами рівня, що є важливим елементом геймплею, оскільки додає викликів та напруження. Якщо гравець не вчасно оминає їх, то це призводить до поразки гравця. Перешкоди можуть бути різного типу, включаючи нерухомі об'єкти, такі як стіни і бар'єри. Це робить гру більш динамічною і непередбачуваною, вимагаючи від гравця швидких реакцій та стратегічного мислення.

На рисунку 3.34 зображено момент поразки гравця. У цьому моменті можна побачити, як персонаж стикається з перешкодою і зазнає поразки. Це може бути виражено через різні візуальні та звукові ефекти, такі як спалахи, вибухи або зміна кольору екрану, які сигналізують про те, що гравець не зміг уникнути небезпеки. Такий зворотний зв'язок важливий для гравця, оскільки дозволяє чітко зрозуміти причину поразки і спонукає його до покращення своїх навичок.

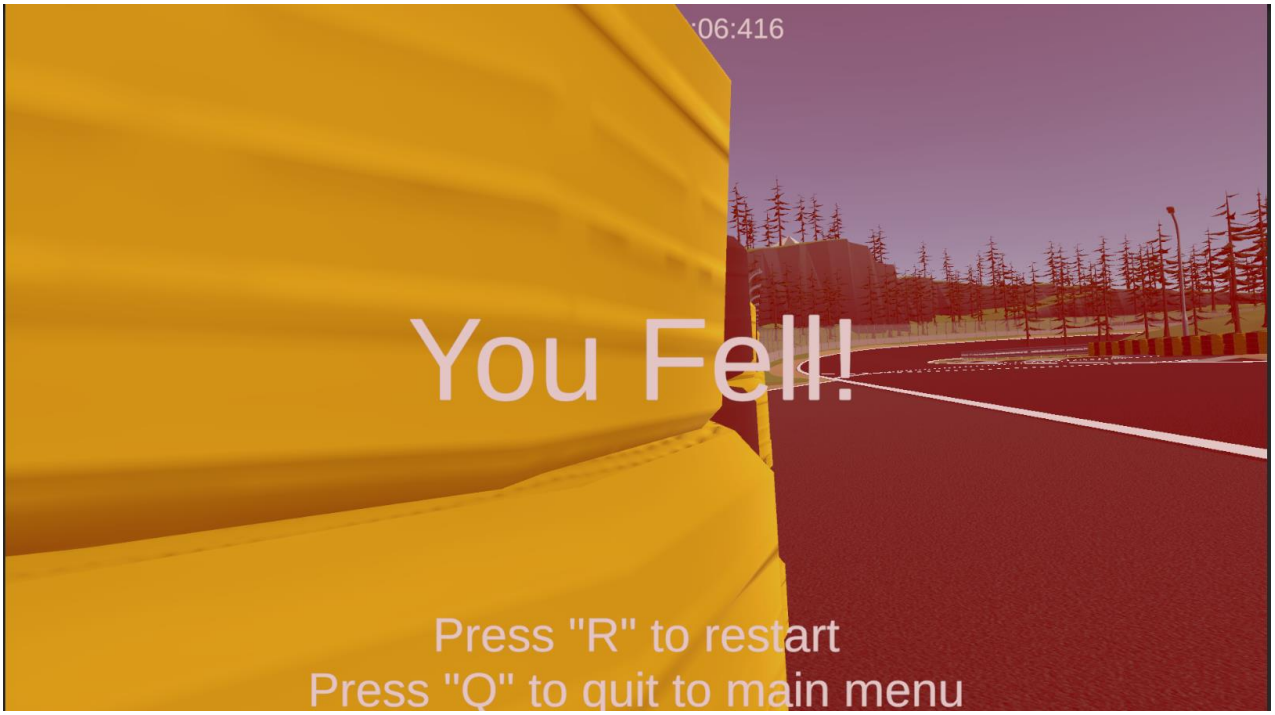

Рисунок 3.34 – Загальний вигляд інтерфейсу після поразки гравця

Останньою частиною гри є екран перемоги гравця, який є важливим етапом для відображення досягнень гравця і визначення його успішності в поточному раунді чи рівні. Цей екран з'являється після успішного завершення кола чи досягнення певних цілей гравцем і має на меті підсумувати його досягнення.

На екрані перемоги коректно відображаються наступні елементи:

1. Найкращий час гравця: Показується час, який гравець зміг досягти в даному раунді чи рівні. Це важливий показник успішності і може слугувати для порівняння результатів між різними спробами гравця.
2. Медалі: В залежності від досягнутого часу або інших критеріїв гра може вручити гравцеві медалі, які відображають його результати. Зазвичай це може бути золота, срібна або бронзова медалі, які відповідають за досягнутий рівень успішності.
3. Час поточної спроби: Показується час, який було витрачено на поточну спробу чи раунд. Це дозволяє гравцеві зрозуміти, наскільки близько він був до свого найкращого результату або цілей.

Екран перемоги гравця є важливим моментом в ігровому процесі, оскільки він надає гравцю відчуття задоволення від досягнення успіху та мотивує його до подальших досягнень. Правильне відображення інформації на цьому екрані дозволяє гравцеві чітко оцінити свої навички та виконання, а також стимулює до покращення результатів у майбутніх гральних сесіях.

На рисунку 3.35 зображено момент перемоги гравця. Цей рисунок ілюструє, як виглядає екран з показниками найкращого часу, вручені медалі та час поточної спроби. Чітке і привабливе візуальне представлення дозволяє гравцю відчути себе в центрі уваги і оцінити свої досягнення в грі.

Рисунок 3.35 – Загальний вигляд інтерфейсу після перемоги гравця

За результатами тестування комп’ютерної гри «Лонгбординг» на базі ігрового рушія Unity підтверджено працездатність усіх функцій.

3.6 Висновок

У третьому розділі дипломної роботи детально розглядається процес створення та тестування 3D гри у жанрі лонгбордингу. Цей розділ охоплює

основні етапи, такі як концептуалізація, проектування, реалізація та тестування, що дозволило створити кінцевий продукт високої якості.

На етапі проектування ретельно підбиралися інструменти та технології, необхідні для розробки гри. Було обрано ігровий рушій, який найкраще підходив для реалізації 3D графіки та забезпечував легку інтеграцію з іншими програмними засобами. Вибір рушія базувався на його функціональних можливостях, гнучкості та зручності у використанні. Також було обрано мову програмування, яка забезпечувала оптимальну продуктивність та легкість у навчанні і використанні.

Реалізація гри включала створення основних ігрових механік. Було розроблено систему руху персонажа, яка дозволяла гравцям легко керувати своїм персонажем, симулюючи катання на лонгборді. Також було впроваджено систему взаємодії з об'єктами, яка включала зіткнення з перешкодами. Крім того, система медалей та рівнів забезпечувала мотивацію гравців до проходження гри.

Ретельне тестування гарантувало високий рівень стабільності гри та її готовність до випуску. Завдяки систематичному підходу до тестування були виявлені та виправлені більшість помилок на ранніх етапах, що дозволило уникнути серйозних проблем на фінальному етапі розробки. Це забезпечило високий рівень готовності продукту до комерційного використання.

ВИСНОВКИ

Поставлені перед бакалаврською кваліфікаційною роботою задачі виконано в повному обсязі, а саме:

- Обгрунтувано доцільність створення комп’ютерної гри «Лонгбординг»
- Проаналізовано існуючі рішення: Проведено аналіз існуючих 2D і 3D ігор у жанрі лонгбординг для визначення найкращих практик та виявлення можливостей для інновацій.
- Спроектовано програмний модуль комп’ютерної гри.
- Реалізовано комп’ютерну гру “лонгбординг” на базі ігрового рушія Unity.
- Протестовано комп’ютерної гри, щоб впевнетись у відсутності збоїв у грі.
- Створено інструкцію користувача.

У ході виконання бакалаврської роботи були проаналізовані методи та принципи розробки комп'ютерних ігор жанру лонгбординг. Розглянуто сучасні програмні аналоги ігор цього жанру. Відповідно до завдання бакалаврської роботи були розроблені структурна схема програмного модуля, методи обробки та управління базами даних гравців та алгоритм роботи основних ігрових механік.

Також було проведено тестування розробленої гри.

Мета дослідження - розширення функціональних можливостей комп’ютерних ігор жанру лонгбординг та впровадженні ефективних методів проектування, розробки та тестування 3D гри у жанрі лонгбординг, що забезпечить високу якість користувацького досвіду, стабільність роботи гри та її конкурентоспроможність на ринку комп'ютерних ігор.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Тези доповіді [Електронний ресурс] – Режим доступу до ресурсу: https://sci-conf.com.ua/ix-mizhnarodna-naukovo-praktichna-konferentsiya-innovative-development-of-science-technology-and-education-6-8-06-2024-vankuver-kanada-arhiv/
2. Nicoll, B., Keogh, B., Nicoll, B., & Keogh, B. (2019). The Unity game engine and the circuits of cultural software (pp. 1-21). Springer International Publishing.
3. Tanuki Sunset. [Електронний ресурс]. Режим доступу: https://rewindgames.itch.io/tanuki-sunset
4. Turn your Longboard into a game controller. [Електронний ресурс]. Режим доступу: https://sk8-bit.itch.io/longboard
5. Skater-girl (2018). [Електронний ресурс]. Режим доступу: https://emotiontheory.itch.io/skater-girl-2018
6. Johnson, D., & Wiles, J. (2003). Effective affective user interface design in games. Ergonomics, 46(13-14), 1332-1345.
7. Moizer, J., Lean, J., Towler, M., Smith, G., & Encinas-Arquero, P. (2007). Using causal loop diagramming to represent the factors affecting learning for students using a business simulation game. Esic Market, 123, 161-177.
8. Okita, A. (2019). Learning C# programming with Unity 3D. AK Peters/CRC Press.
9. Hocking, J. (2022). Unity in action: multiplatform game development in C. Simon and Schuster.
10. Amann, S., Proksch, S., Nadi, S., & Mezini, M. (2016, March). A study of visual studio usage in practice. In 2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER) (Vol. 1, pp. 124-134). IEEE.
11. Cartoon Race Track - Oval [Електронний ресурс]. Режим доступу: https://assetstore.unity.com/packages/3d/environments/roadways/cartoon-race-track-oval-175061
12. Arcadia Longboard [Електронний ресурс]. Режим доступу:

https://sketchfab.com/3d-models/arcadia-longboard-f14f4545c27b49cfab16965c8509b79f

13. Longboard texturing practice [Електронний ресурс]. Режим доступу: https://sketchfab.com/3d-models/longboard-texturing-practice-1ba2620fab2c4433b5184a0514d46bb0
14. Findlater, L., & McGrenere, J. (2004, April). A comparison of static, adaptive, and adaptable menus. In Proceedings of the SIGCHI conference on Human factors in computing systems (pp. 89-96).

ДОДАТОК А (обов'язковий)
ПРОТОКОЛ ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ НА НАЯВНІСТЬ ТЕКСТОВИХ ЗАПОЗИЧЕНЬ

Назва роботи: Комп'ютерна гра Лонгбординг на базі ігрового рушія Unity

Тип роботи: бакалаврська дипломна робота
(БДР, МКР)

Підрозділ кафедра комп'ютерних наук, ФІТА
(кафедра, факультет)

Показники звіту подібності Unicheck

Оригінальність 90,12% Схожість 9,88%

Аналіз звіту подібності (відмітити потрібне):

- ✓ Запозичення, виявлені у роботі, оформлені коректно і не містять ознак плагіату.
- □ Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її виконання автором. Роботу направити на розгляд експертної комісії кафедри.
- □ Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

Особа, відповідальна за перевірку Озеранський В.С.

Ознайомлені з повним звітом подібності, який був згенерований системою Unicheck щодо роботи.

Автор роботи Матвієнко А.І.

Керівник роботи Малініч І.П.

ДОДАТОК Б (ОБОВ'ЯЗКОВИЙ)

Акт впровадження

УЗГОДЖЕНО	ЗАТВЕРДЖУЮ
Директор ЦДО ВНТУ	Завідувач кафедри КН
к.т.н., доц. каф. БМІОЕС	д.т.н., професор
Тужанський С. Є.	Яровий А.А.
_____	_____
«____» _____ 2024 року	«____» _____ 2024 року

АКТ ВПРОВАДЖЕННЯ № 2 / 31.05.2024
результатів бакалаврської дипломної роботи

Замовник: Центр діджиталізації освітнього процесу Вінницького національного технічного університету
(найменування організації)

Цим актом підтверджується, що результати роботи – «Комп'ютерна гра Лонгбординг на базі ігрового рушія Unity»
(найменування теми)

що виконав студент гр. 3КН–20б, Матвієнко А. І., на громадських засадах
(виконавець)

відповідно до планів робіт Центру діджиталізації освітнього процесу ВНТУ на 2023-2024 р. та впроваджено у Вінницькому національному технічному університеті
(строки виконання) (найменування організації, де здійснювалося впровадження)

1. Вид впроваджених результатів: комп'ютерна гра
(експлуатація виробу, роботи, технології)

2. Характеристика масштабу впровадження: одиничне
(унікальне, одиничне, партія, масове, серійне)

3. Форма впровадження: комп'ютерна гра для навчання катання на лонгборді

4. Новизна результатів роботи: ігрові механіки катання на лонгбордах
(піонерські, принципово нові, якісно нові, модифікації, модернізація старих розробок)

5. Впроваджені: у гейміфікації навчального процесу фізкультури

6. Соціальний та науково-технічний ефект: студентам буде легше навчитись катанню на лонгбордах, оскільки вони зможуть краще відчути механіку катання на лонгборді. Гарно відпрацьована техніка катання дозволить уникнути фізичних травм при катанні на спортивному майданчику, а також на вулицях міста

(охорона навколишнього середовища, поліпшення й оздоровлення умов праці, удосконалення структури керування, науково-технічних напрямків, спеціальне призначення)

Від виконавця:
студент групи 3КН–20б

_____ Матвієнко А.І.

Керівник: асистент кафедри КН

Від Центру діджиталізації
освітнього процесу ВНТУ, директор,
к.т.н., доц. каф. БМІОЕС:

_____ Тужанський С.Є.

_____ Малініч І.П.

ДОДАТОК В
ІНСТРУКЦІЯ КОРИСТУВАЧА КОМП'ЮТЕРНОЇ ГРИ ЛОНГБОРДИНГ НА БАЗІ ІГРОВОГО РУШІЯ UNITY

1. Відкриваємо папку з проектом у провіднику операційної системи
2. Запускаємо файл ProjectLongboarding.exe

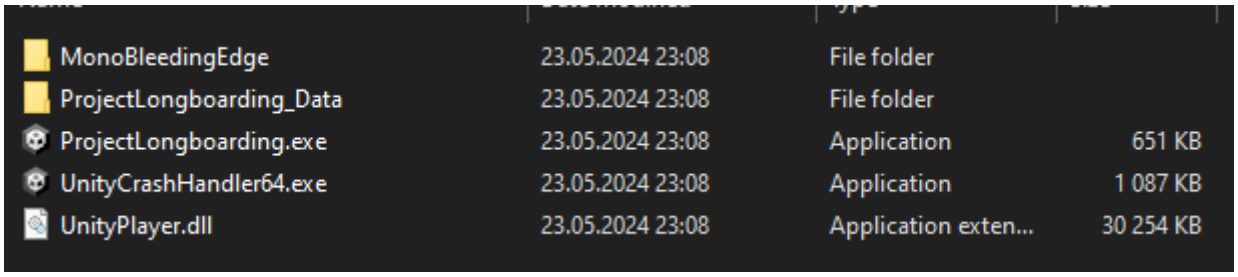

Рисунок Б.1 – Зображення файла запуску

3. Переглядаємо головне меню гри, переходимо на екран вибору рівнів

Рисунок Б.2 – Загальний вигляд меню програмного модуля

4. Після вибору рівня, почніть гру, керуючи за допомогою клавіш WASD

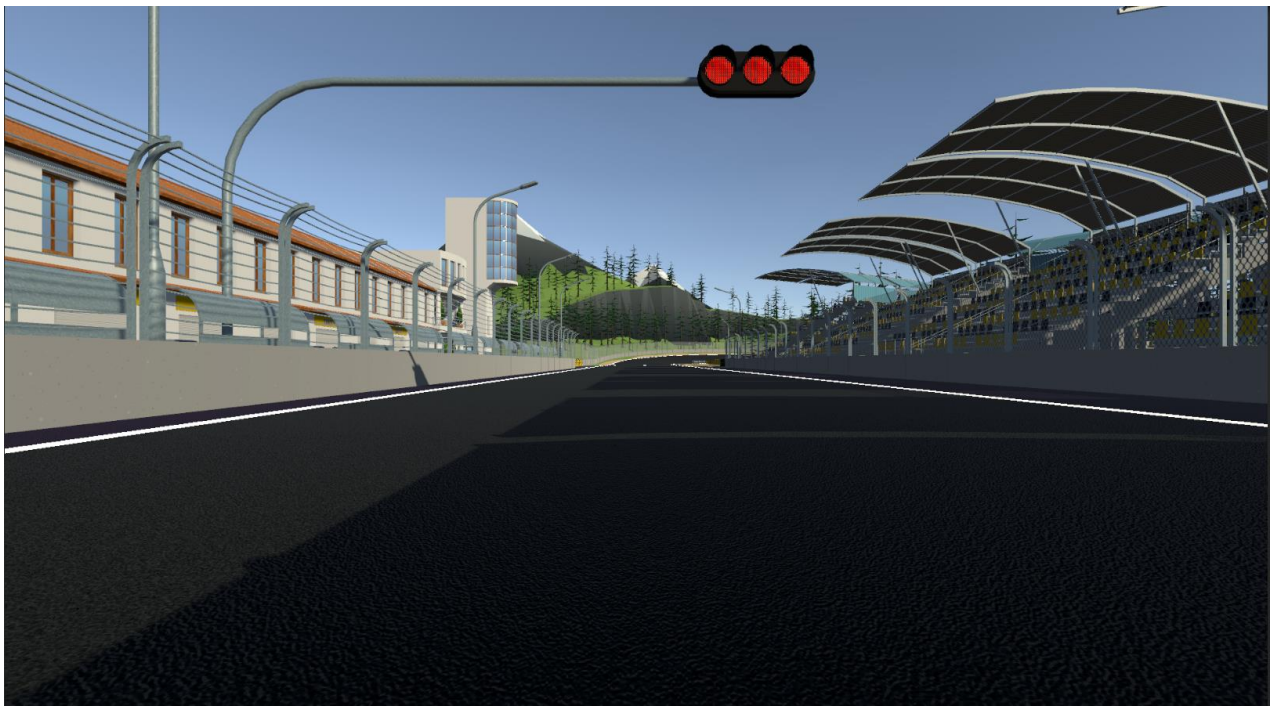

Рисунок Б.3 – Загальний вигляд програмного модуля

ДОДАТОК Г (обов'язковий)

ЛІСТИНГ ПРОГРАМИ

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

[RequireComponent(typeof(CharacterController))]
public class FPController : MonoBehaviour
{
    public Camera playerCamera;
    public float acceleration = 10f;
    public float maxSpeed = 1200f;
    public float gravity = 10f;
    public float cameraBobAmount = 1f;
    float bobbingOffset = 0f;
    float lowestBobbingOffset = float.MaxValue;
    bool isDescending = false;
    public AudioClip bobSound;
    private AudioSource audioSource;
    public float cameraBobSpeed = 50f;

    public float lookSpeed = 2f;
    public float lookXLimit = 45f;

    public float turnSpeed = 50f; // Variable to control the turning speed
    public float maxTiltAngle = 15f; // Maximum tilt angle for the camera
    public float tiltSpeed = 5f; // Speed at which the camera tilts

    Vector3 moveDirection = Vector3.zero;
    public float currentSpeed = 0f;
    float movementDirectionY = 0f;
    float rotationX = 0f;
    Vector3 originalCameraPosition;

    public bool canMove = true;

    CharacterController characterController;
    [SerializeField] TimerController timerController;

    void Start()
    {
        characterController = GetComponent<CharacterController>();
        Cursor.lockState = CursorLockMode.Locked;
        Cursor.visible = false;

        originalCameraPosition = playerCamera.transform.localPosition;
        audioSource = GetComponent<AudioSource>();
        audioSource.clip = bobSound;
```

```
    }

    void Update()
    {
        #region Handles Movement
        Vector3 forward = transform.forward;

        if (canMove && !timerController.itEnded)
        {
            if (Input.GetAxis("Vertical") > 0)
                currentSpeed = Mathf.Min(currentSpeed + Time.deltaTime *
acceleration, maxSpeed);
            else if (Input.GetAxis("Vertical") < 0)
                currentSpeed = Mathf.Max(currentSpeed - Time.deltaTime *
acceleration, 0f);
        }
        else if (!timerController.itEnded && currentSpeed < 0.01)
        {
            currentSpeed = 0f;
        }

        movementDirectionY -= gravity * Time.deltaTime;
        moveDirection = forward * currentSpeed + new Vector3(0, movementDirectionY,
0);
        if (currentSpeed >= 0.01)
        {
            currentSpeed = currentSpeed - 0.01f;
        }
        characterController.Move(moveDirection * Time.deltaTime);
        #endregion

        #region Handles Camera Bobbing
        if (currentSpeed > 0 && Input.GetAxis("Vertical") != 0 && currentSpeed <
maxSpeed && canMove && !timerController.itEnded)
        {
            bobbingOffset = -Mathf.Sin(Time.time * cameraBobSpeed) * cameraBobAmount;
            if (bobbingOffset < 0)
            {
                bobbingOffset *= 6f;
                isDescending = true;
            }
            else
            {
                isDescending = false;
                lowestBobbingOffset = float.MaxValue;
            }
            Vector3 targetCameraPosition = originalCameraPosition + new Vector3(0,
bobbingOffset, 0);
            playerCamera.transform.localPosition = targetCameraPosition;
```

```
            if (isDescending)
            {
                lowestBobbingOffset = Mathf.Min(lowestBobbingOffset, bobbingOffset);
            }

            if (isDescending && bobbingOffset <= lowestBobbingOffset)
            {
                audioSource.Play();
            }
        }
        else
        {
            playerCamera.transform.localPosition =
Vector3.Lerp(playerCamera.transform.localPosition, originalCameraPosition,
Time.deltaTime * 5f);
            lowestBobbingOffset = float.MaxValue;
            isDescending = false;
        }
        #endregion

        #region Handles Rotation and Tilt
        if (canMove && !timerController.itEnded)
        {
            // Handle vertical rotation
            rotationX += -Input.GetAxis("Mouse Y") * lookSpeed;
            rotationX = Mathf.Clamp(rotationX, -lookXLimit, lookXLimit);

            // Gradual turning
            float turnInput = Input.GetAxis("Horizontal");
            if (turnInput != 0)
            {
                transform.rotation *= Quaternion.Euler(0, turnInput * turnSpeed *
Time.deltaTime, 0);
            }

            // Camera tilt
            float targetTiltAngle = maxTiltAngle * -turnInput;
            Quaternion targetTiltRotation = Quaternion.Euler(0, 0, targetTiltAngle);

            // Combine tilt with vertical look rotation
            Quaternion currentCameraRotation = Quaternion.Euler(rotationX,
playerCamera.transform.localRotation.eulerAngles.y, 0);
            playerCamera.transform.localRotation =
Quaternion.Slerp(playerCamera.transform.localRotation, currentCameraRotation *
targetTiltRotation, Time.deltaTime * tiltSpeed);
        }
        #endregion
    }
}
```

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

public class TimerController : MonoBehaviour
{
    [SerializeField] private Collider startFinishCollider; // Collider for
start/finish line
    [SerializeField] private List<Collider> checkpoints; // List of checkpoint
triggers
    [SerializeField] private TMP_Text timerText; // Reference to the TMP Text
component for displaying the timer
    [SerializeField] private GameObject overlay;
    [SerializeField] private SaveNumber saveNumber;
    public bool itEnded = false;
    public bool timerRunning = false;
    public float timer = 0f;
    private int currentCheckpointIndex = -1; // Initialize to -1 to indicate player
hasn't reached any checkpoints yet
    private bool cheated = false;

    void Start()
    {
        // Make sure the start/finish collider is set to be a trigger
        startFinishCollider.isTrigger = true;

        // Make sure all checkpoints are set to be triggers
        foreach (Collider checkpoint in checkpoints)
        {
            checkpoint.isTrigger = true;
        }
    }

    void OnTriggerEnter(Collider other)
    {
        // Check if the player collides with a checkpoint
        if (checkpoints.Contains(other))
        {
            int checkpointIndex = checkpoints.IndexOf(other);
            if (checkpointIndex == currentCheckpointIndex + 1)
            {
                currentCheckpointIndex = checkpointIndex;
                Debug.Log("Checkpoint " + (currentCheckpointIndex + 1) + "
reached!");
            }
            else if (checkpointIndex != currentCheckpointIndex + 1)
            {
                Debug.Log("Cheating detected! Player skipped checkpoints.");
                cheated = true;
            }
```

```
        // Else, player revisited previous checkpoint, no action needed
    }
    // Check if the player collides with the start/finish line
    else if (other == startFinishCollider)
    {
        if (!timerRunning)
        {
            StartTimer();
        }
        else if (!cheated && currentCheckpointIndex > 1)
        {
            StopTimer();
        }
    }
}

void Update()
{
    if (timerRunning)
    {
        timer += Time.deltaTime;
        overlay.SetActive(true);
        UpdateTimerDisplay();
    }

    if (!timerRunning)
    {
        overlay.SetActive(false);
    }
}

void StartTimer()
{
    timerRunning = true;
    Debug.Log("Timer started!");
}

void StopTimer()
{
    Debug.Log("Timer stopped!");
    itEnded = true;
    saveNumber.Save();
    timerRunning = false;
}

void UpdateTimerDisplay()
{
    if (timerText != null)
    {
```

```
            timerText.text = "Timer: " + FormatTime(timer); // Format timer value to
two decimal places
        }
    }

    public Transform GetNextCheckpointTransform()
    {
        if (currentCheckpointIndex < checkpoints.Count - 1)
        {
            return checkpoints[currentCheckpointIndex + 1].transform;
        }
        else
        {
            return null; // If there are no more checkpoints, return null
        }
    }

    private string FormatTime(float timeInSeconds)
    {
        int minutes = Mathf.FloorToInt(timeInSeconds / 60);
        int seconds = Mathf.FloorToInt(timeInSeconds % 60);
        int milliseconds = Mathf.FloorToInt((timeInSeconds -
Mathf.Floor(timeInSeconds)) * 1000);
        return string.Format("{0:00}:{1:00}:{2:000}", minutes, seconds,
milliseconds);
    }
}



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

public class SceneTransition : MonoBehaviour
{
    [SerializeField] private Camera mainMenuCamera;
    [SerializeField] private Camera levelSelectionCamera;
    [SerializeField] private Camera transitionCamera;
    [SerializeField] private Button playButton;
    [SerializeField] private float transitionDuration = 1.0f;
    [SerializeField] private float arcHeight = 2.0f; // Adjust the arc height as
needed
    [SerializeField] private string levelSceneName = "Level1";

    private void Start()
    {
        playButton.onClick.AddListener(OnPlayButtonClicked);
    }
```

```
    private void OnPlayButtonClicked()
    {
        StartCoroutine(SmoothTransitionToLevelSelection());
    }

    private IEnumerator SmoothTransitionToLevelSelection()
    {
        float elapsedTime = 0f;

        Vector3 startTransitionPosition = mainMenuCamera.transform.position;
        Vector3 endTransitionPosition = levelSelectionCamera.transform.position;

        // Calculate control point for the Bezier curve
        Vector3 controlPoint = (startTransitionPosition + endTransitionPosition) /
2f;
        controlPoint += Vector3.up * arcHeight;

        transitionCamera.gameObject.SetActive(true);
        transitionCamera.transform.position = startTransitionPosition;
        transitionCamera.transform.rotation = mainMenuCamera.transform.rotation;

        mainMenuCamera.gameObject.SetActive(false);

        while (elapsedTime < transitionDuration)
        {
            elapsedTime += Time.deltaTime;
            float t = elapsedTime / transitionDuration;

            // Calculate position along the Bezier curve
            Vector3 bezierPosition = CalculateBezierPoint(startTransitionPosition,
controlPoint, endTransitionPosition, t);

            // Smoothly interpolate the position
            transitionCamera.transform.position = bezierPosition;

            // Smoothly interpolate the rotation
            transitionCamera.transform.rotation =
Quaternion.Slerp(mainMenuCamera.transform.rotation,
levelSelectionCamera.transform.rotation, t);

            yield return null;
        }

        // Ensure the transition camera ends at the exact position and rotation of
the level selection camera
        transitionCamera.transform.position = endTransitionPosition;
        transitionCamera.transform.rotation =
levelSelectionCamera.transform.rotation;
```

```
        transitionCamera.gameObject.SetActive(false);
        levelSelectionCamera.gameObject.SetActive(true);
        SceneManager.LoadScene(levelSceneName);
    }

    // Calculate position along a Bezier curve
    private Vector3 CalculateBezierPoint(Vector3 p0, Vector3 p1, Vector3 p2, float t)
    {
        float u = 1f - t;
        float tt = t * t;
        float uu = u * u;

        Vector3 point = uu * p0;
        point += 2f * u * t * p1;
        point += tt * p2;

        return point;
    }
}

using System;
using System.IO;
using UnityEngine;

public class SaveNumber : MonoBehaviour
{
    private string directoryPath;
    [SerializeField] private TimerController timerController;
    private string filePath;
    [SerializeField] private string levelName;

    private void Start()
    {
        directoryPath =
Path.Combine(Environment.GetFolderPath(Environment.SpecialFolder.MyDocuments),
"TrackDayLongboarding");
        filePath = Path.Combine(directoryPath, levelName + ".sav");

        // Ensure the directory exists
        if (!Directory.Exists(directoryPath))
        {
            Directory.CreateDirectory(directoryPath);
        }
    }

    public void Save()
    {
        if (timerController.itEnded && timerController.timer < Load())
        {
            try
```

```
            {
                File.WriteAllText(filePath, timerController.timer.ToString());
                Debug.Log($"Number {timerController.timer} saved to {filePath}");
            }
            catch (Exception e)
            {
                Debug.LogError($"Failed to save number: {e.Message}");
            }
        }
    }

    public float Load()
    {
        try
        {
            if (File.Exists(filePath))
            {
                string content = File.ReadAllText(filePath);
                if (float.TryParse(content, out float number))
                {
                    Debug.Log($"Loaded number: {number}");
                    return number;
                }
                else
                {
                    Debug.LogWarning("Failed to parse number from file.");
                    return float.MaxValue;
                }
            }
            else
            {
                Debug.LogWarning("File does not exist.");
                return float.MaxValue;
            }
        }
        catch (Exception e)
        {
            Debug.LogError($"Failed to load number: {e.Message}");
            return float.MaxValue;
        }
    }
}



using System;
using System.IO;
using UnityEngine;

public class SaveNumber : MonoBehaviour
```

```
{
    private string directoryPath;
    [SerializeField] private TimerController timerController;
    private string filePath;
    [SerializeField] private string levelName;

    private void Start()
    {
        directoryPath =
Path.Combine(Environment.GetFolderPath(Environment.SpecialFolder.MyDocuments),
"TrackDayLongboarding");
        filePath = Path.Combine(directoryPath, levelName + ".sav");

        // Ensure the directory exists
        if (!Directory.Exists(directoryPath))
        {
            Directory.CreateDirectory(directoryPath);
        }
    }

    public void Save()
    {
        if (timerController.itEnded && timerController.timer < Load())
        {
            try
            {
                File.WriteAllText(filePath, timerController.timer.ToString());
                Debug.Log($"Number {timerController.timer} saved to {filePath}");
            }
            catch (Exception e)
            {
                Debug.LogError($"Failed to save number: {e.Message}");
            }
        }
    }

    public float Load()
    {
        try
        {
            if (File.Exists(filePath))
            {
                string content = File.ReadAllText(filePath);
                if (float.TryParse(content, out float number))
                {
                    Debug.Log($"Loaded number: {number}");
                    return number;
                }
                else
                {
```

```
                    Debug.LogWarning("Failed to parse number from file.");
                    return float.MaxValue;
                }
            }
            else
            {
                Debug.LogWarning("File does not exist.");
                return float.MaxValue;
            }
        }
        catch (Exception e)
        {
            Debug.LogError($"Failed to load number: {e.Message}");
            return float.MaxValue;
        }
    }
}

using UnityEngine;
using UnityEngine.UI;
using System.Collections;

public class MainMenu : MonoBehaviour
{
    [SerializeField] private Camera mainMenuCamera;
    [SerializeField] private Camera levelSelectionCamera;
    [SerializeField] private Camera transitionCamera;
    [SerializeField] private Button playButton;
    [SerializeField] private float transitionDuration = 1.0f;
    [SerializeField] private float arcHeight = 2.0f; // Adjust the arc height as
needed

    private void Start()
    {
        Cursor.lockState = CursorLockMode.Confined;
        Cursor.visible = true;
        playButton.onClick.AddListener(OnPlayButtonClicked);
        mainMenuCamera.gameObject.SetActive(true);
        levelSelectionCamera.gameObject.SetActive(false);
        transitionCamera.gameObject.SetActive(false);
    }

    private void OnPlayButtonClicked()
    {
        StartCoroutine(SmoothTransitionToLevelSelection());
    }

    private IEnumerator SmoothTransitionToLevelSelection()
    {
        float elapsedTime = 0f;
```

```
        Vector3 startTransitionPosition = mainMenuCamera.transform.position;
        Vector3 endTransitionPosition = levelSelectionCamera.transform.position;

        // Calculate control point for the Bezier curve
        Vector3 controlPoint = (startTransitionPosition + endTransitionPosition) /
2f;
        controlPoint += Vector3.up * arcHeight;

        transitionCamera.gameObject.SetActive(true);
        transitionCamera.transform.position = startTransitionPosition;
        transitionCamera.transform.rotation = mainMenuCamera.transform.rotation;

        mainMenuCamera.gameObject.SetActive(false);

        while (elapsedTime < transitionDuration)
        {
            elapsedTime += Time.deltaTime;
            float t = elapsedTime / transitionDuration;

            // Calculate position along the Bezier curve
            Vector3 bezierPosition = CalculateBezierPoint(startTransitionPosition,
controlPoint, endTransitionPosition, t);

            // Smoothly interpolate the position
            transitionCamera.transform.position = bezierPosition;

            // Smoothly interpolate the rotation
            transitionCamera.transform.rotation =
Quaternion.Slerp(mainMenuCamera.transform.rotation,
levelSelectionCamera.transform.rotation, t);

            yield return null;
        }

        // Ensure the transition camera ends at the exact position and rotation of
the level selection camera
        transitionCamera.transform.position = endTransitionPosition;
        transitionCamera.transform.rotation =
levelSelectionCamera.transform.rotation;

        transitionCamera.gameObject.SetActive(false);
        levelSelectionCamera.gameObject.SetActive(true);
    }

    // Calculate position along a Bezier curve
    private Vector3 CalculateBezierPoint(Vector3 p0, Vector3 p1, Vector3 p2, float t)
    {
        float u = 1f - t;
        float tt = t * t;
```

```
        float uu = u * u;

        Vector3 point = uu * p0;
        point += 2f * u * t * p1;
        point += tt * p2;

        return point;
    }
}

using UnityEngine;
using UnityEngine.SceneManagement;

public class SceneController : MonoBehaviour
{
    public string restartSceneName;
    public string nextSceneName;

    private void Update()
    {
        if (Input.GetKeyDown(KeyCode.R))
        {
            RestartScene();
        }
        else if (Input.GetKeyDown(KeyCode.Q))
        {
            ChangeScene();
        }
    }

    private void RestartScene()
    {
        SceneManager.LoadScene(restartSceneName);
    }

    private void ChangeScene()
    {
        SceneManager.LoadScene(nextSceneName);
    }
}

using System.Collections;
using System.Collections.Generic;
using System.Runtime.CompilerServices;
using Unity.VisualScripting;
using UnityEngine;

public class Skatingaudio : MonoBehaviour
{
    [SerializeField] FPController fPController;
```

```
    [SerializeField] SpeedMonitor speedMonitor;
    private AudioSource audioSource;
    // Start is called before the first frame update
    void Start()
    {
        audioSource = GetComponent<AudioSource>();
        audioSource.loop = true;
    }

    // Update is called once per frame
    void Update()
    {
        if (speedMonitor.actualSpeed > 0.01f)
        {
        float normalizedSpeed = Mathf.Clamp01(speedMonitor.actualSpeed /
fPController.maxSpeed);
        float pitch = Mathf.Lerp(0.5f, 1.0f, normalizedSpeed);
        audioSource.pitch = pitch;
        }
        else
        {
            audioSource.pitch = 0;
        }
    }
}

using UnityEngine;

public class SpeedMonitor : MonoBehaviour
{
    public FPController fpController; // Reference to the FPController script
    public float speedDifferenceThreshold = 5f; // Threshold for the difference
between actual speed and current speed
    public float actualSpeed = 0;
    private CharacterController characterController;
    [SerializeField] TimerController timerController;
    [SerializeField] private GameObject deathSreen;

    void Start()
    {
        deathSreen.SetActive(false);
        // Ensure the FPController reference is set
        if (fpController == null)
        {
            fpController = GetComponent<FPController>();
        }

        // Get the CharacterController component
        characterController = GetComponent<CharacterController>();
    }
```

```
    void Update()
    {
        if (fpController != null && characterController != null &&
!timerController.itEnded)
        {
            // Calculate the actual speed of the player using the
CharacterController's velocity
            Vector3 horizontalVelocity = new Vector3(characterController.velocity.x,
0, characterController.velocity.z);
            actualSpeed = horizontalVelocity.magnitude;

            // Calculate the difference between the actual speed and the current
speed
            float speedDifference = Mathf.Abs(actualSpeed -
fpController.currentSpeed);

            // Check if the difference exceeds the threshold
            if (speedDifference > speedDifferenceThreshold)
            {
                // Stop the player by disabling movement
                fpController.canMove = false;
                fpController.currentSpeed = 0;
                deathSreen.SetActive(true);
                Debug.Log("Player stopped due to large speed difference. Actual
speed: " + actualSpeed + ", Current speed: " + fpController.currentSpeed);
            }
        }
    }
}

using UnityEngine;
using TMPro;

public class EndScreenController : MonoBehaviour
{
    [SerializeField] private TMP_Text playerTimeText;
    [SerializeField] private TMP_Text goldTimeText;
    [SerializeField] private TMP_Text silverTimeText;
    [SerializeField] private TMP_Text bronzeTimeText;
    [SerializeField] private GameObject endScreenCanvas;
    [SerializeField] private TimerController timerController;
    [SerializeField] private TMP_Text personalBestText;
    [SerializeField] private SaveNumber saveNumber;
    private float goldTime = 60f;
    private float silverTime = 90f;
    private float bronzeTime = 120f;

    void Start()
    {
```

```
        endScreenCanvas.SetActive(false);
    }

    void Update()
    {
        if (timerController.itEnded && !timerController.timerRunning)
        {
            playerTimeText.text = "Your Time: " + FormatTime(timerController.timer);
            goldTimeText.text = "Gold Time: " + FormatTime(goldTime);
            silverTimeText.text = "Silver Time: " + FormatTime(silverTime);
            bronzeTimeText.text = "Bronze Time: " + FormatTime(bronzeTime);
            if (saveNumber.Load() != float.MaxValue)
            {
                personalBestText.text = "Personal Best: " +
FormatTime(saveNumber.Load());
            }
            endScreenCanvas.SetActive(true);
        }
    }

    private string FormatTime(float timeInSeconds)
    {
        int minutes = Mathf.FloorToInt(timeInSeconds / 60);
        int seconds = Mathf.FloorToInt(timeInSeconds % 60);
        int milliseconds = Mathf.FloorToInt((timeInSeconds -
Mathf.Floor(timeInSeconds)) * 1000);
        return string.Format("{0:00}:{1:00}:{2:000}", minutes, seconds,
milliseconds);
    }
}
```

ДОДАТОК Д (обов'язковий)

ІЛЮСТРАТИВНА ЧАСТИНА

«КОМП'ЮТРЕНА ГРА ЛОНГБОРДИНГ НА БАЗІ ІГРОВОГО РУШІЯ UNITY»

Виконав: студент 4 курсу, групи 3КН-20б
спеціальності 122 – Комп'ютерні науки
(шифр і назва напряму підготовки, спеціальності)
Матвієнко А. І.
(прізвище та ініціали)
Керівник: асистент кафедри КН
Малініч І. П.
(прізвище та ініціали)
«07» 06 2024 р.

Вінниця ВНТУ – 2024 рік

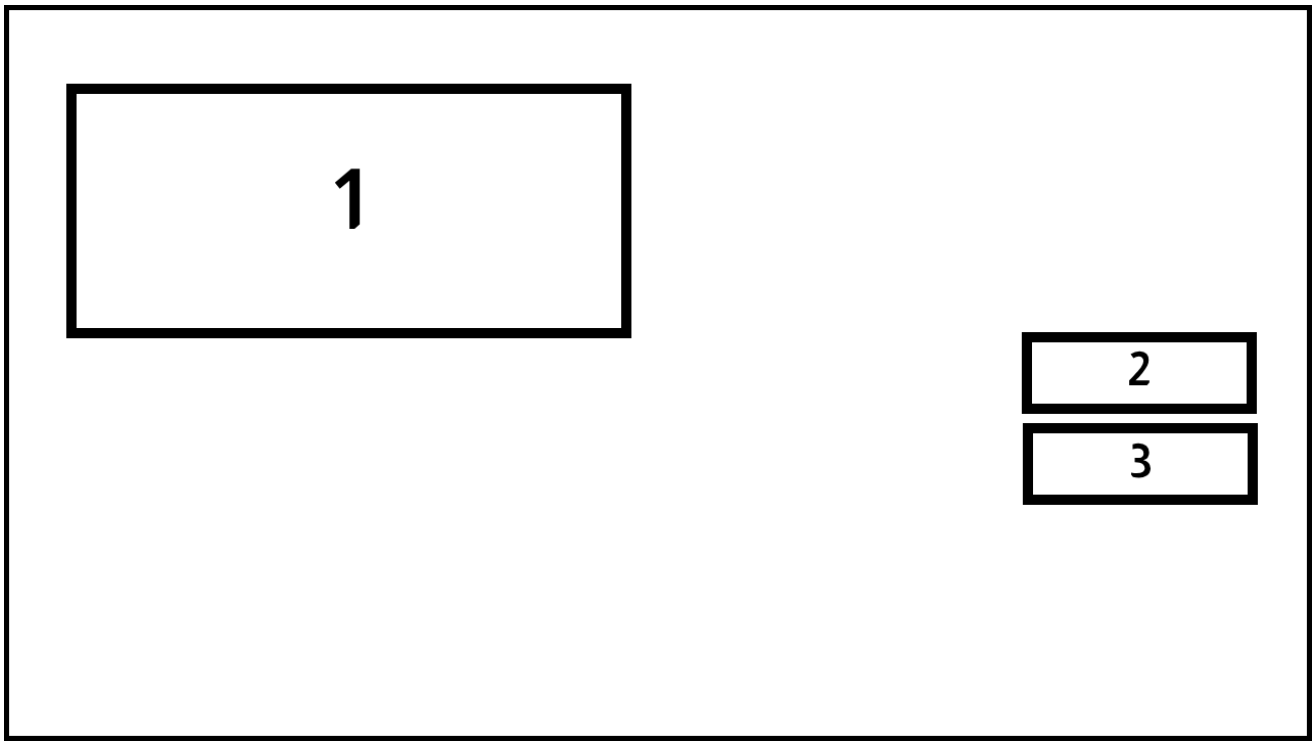

Рисунок Д.1 – Схема головного меню

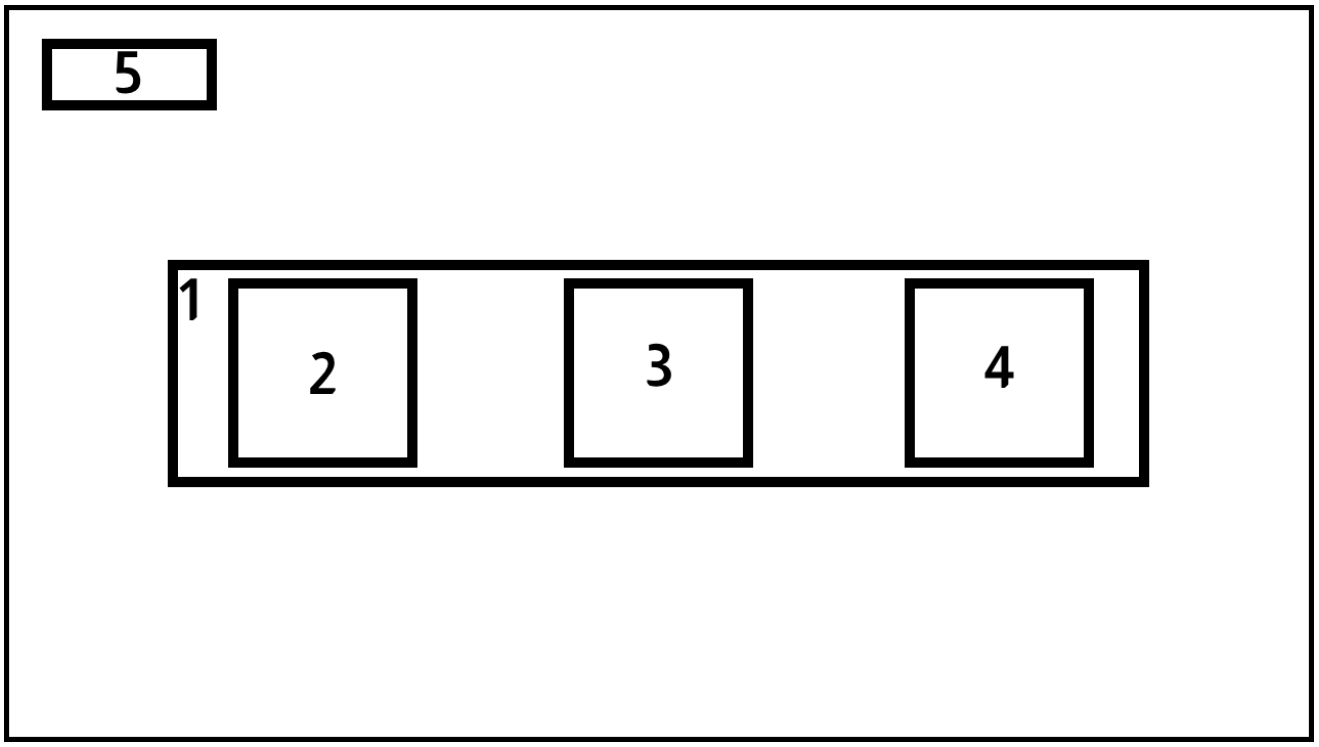

Рисунок Д.2 – Схема меню вибору рівнів

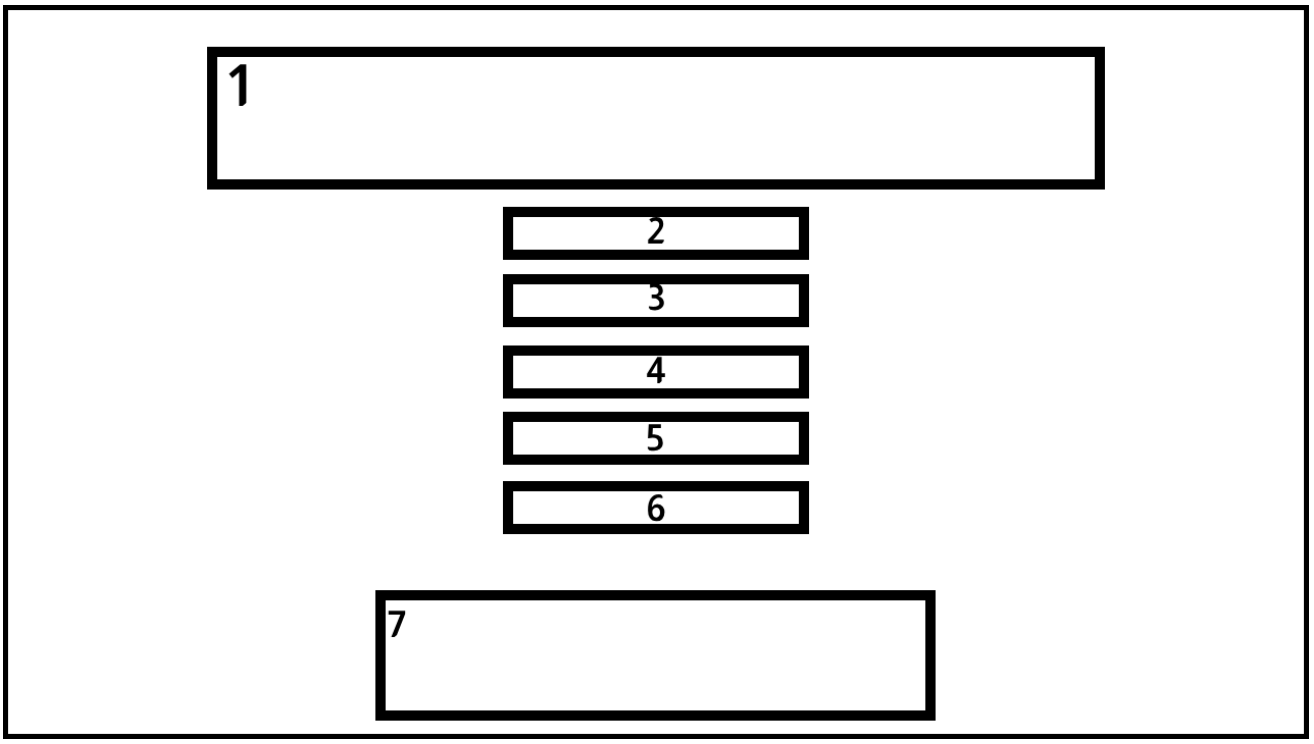

Рисунок Д.3 – Схема екрану перемоги

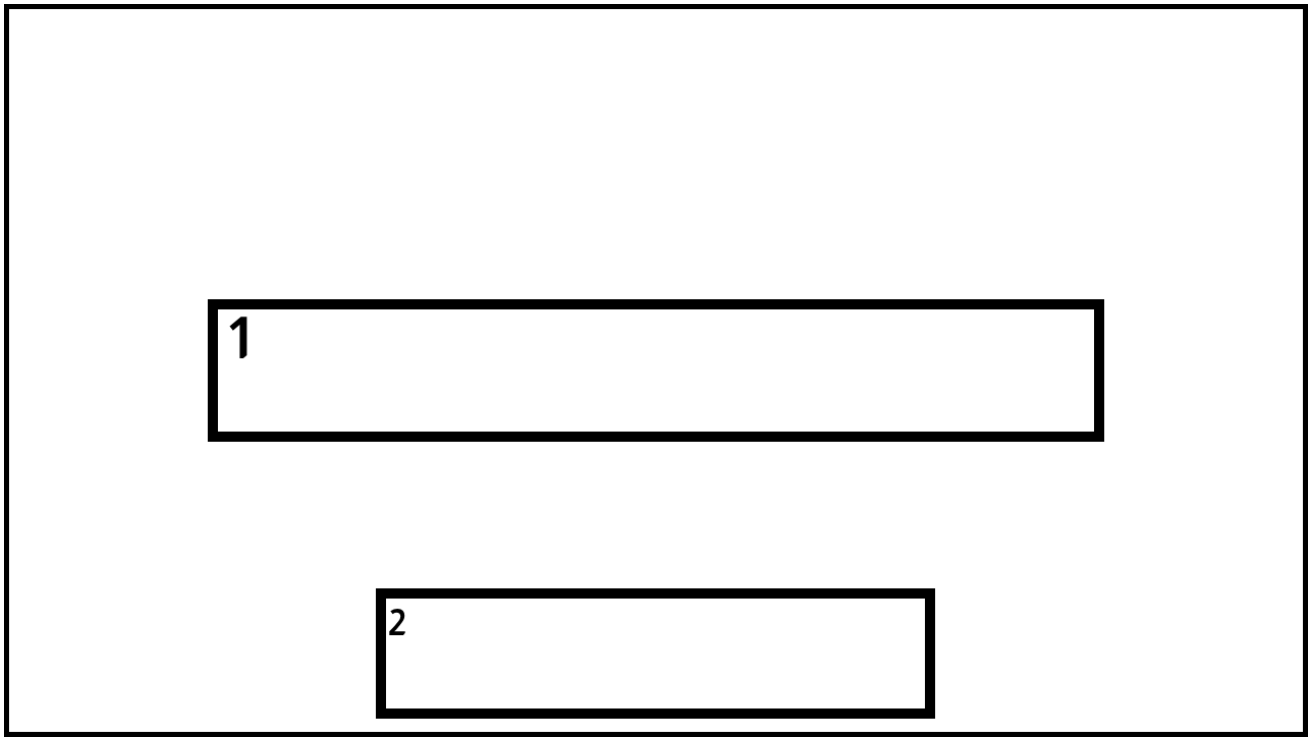

Рисунок Д.4 – Схема екрану поразки

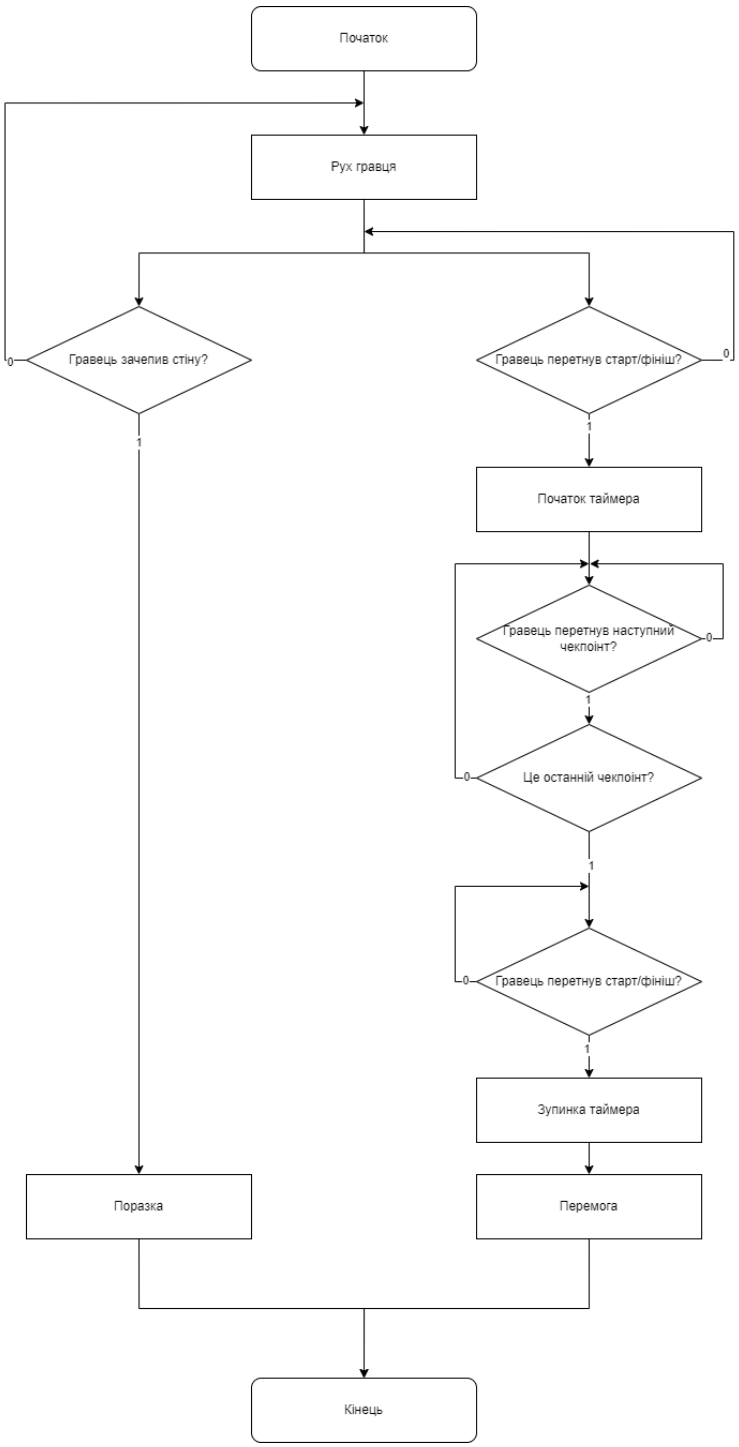

Рисунок Д.5 - Блок-схема алгоритму роботи 3D відеогри жанру “лонгбординг”

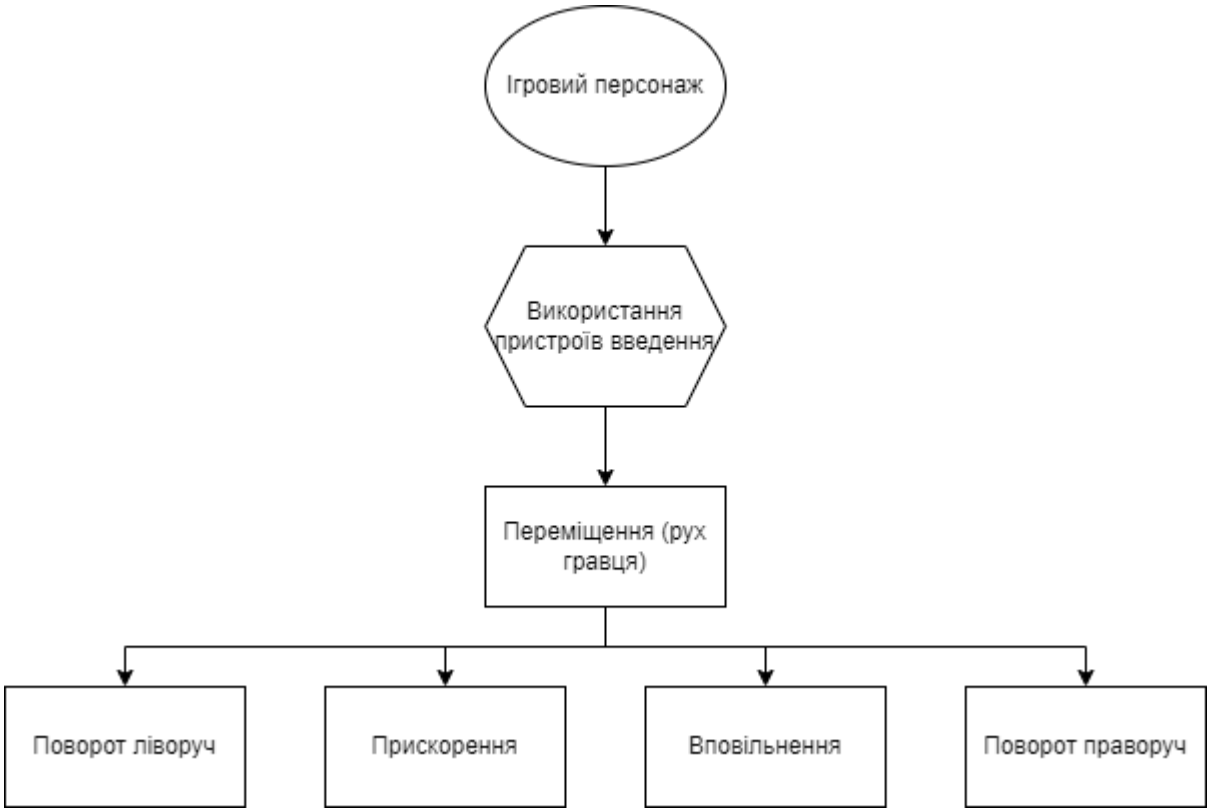

Рисунок Д.6 – Функціональна модель персонажа

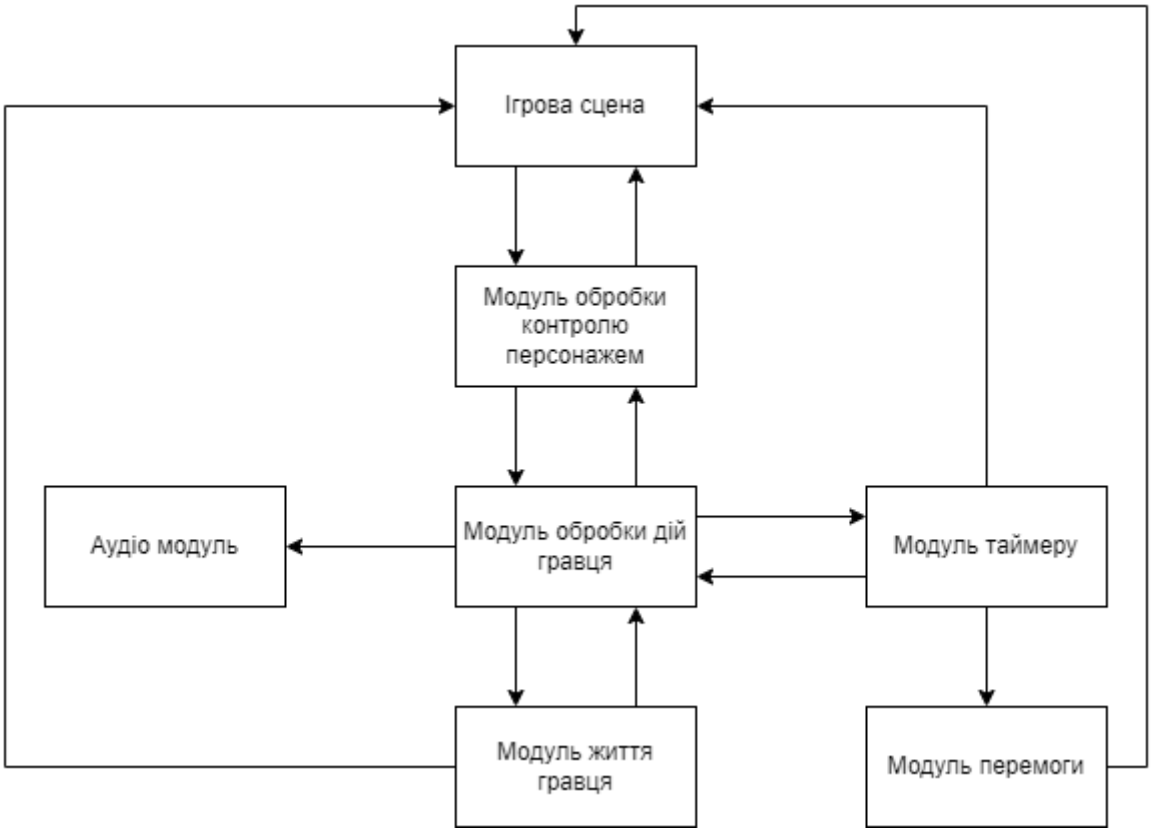

Рисунок Д.7 – Модель реалізації ігрової системи

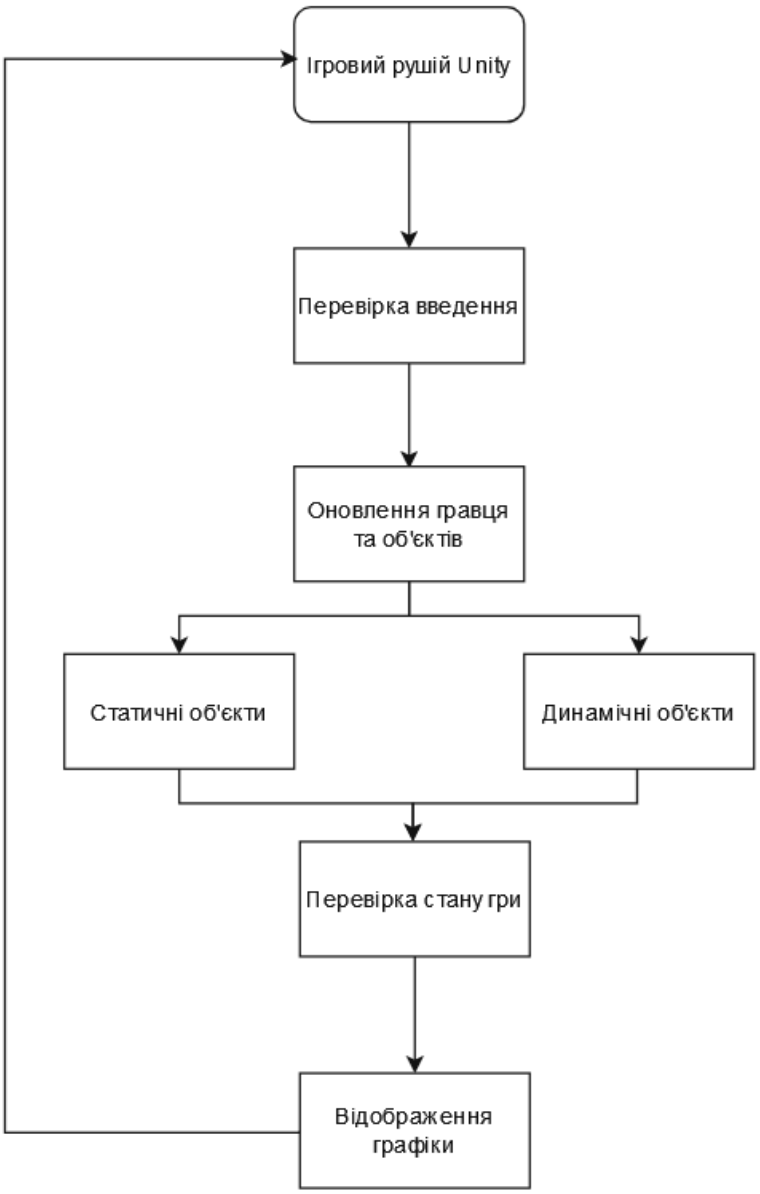

Рисунок Д.8 – Діаграма ігрового циклу

Рисунок Д.9 – Загальний вигляд головного меню комп’ютерної гри

Рисунок Д.10 – Загальний вигляд меню вибору рівнів комп’ютерної гри

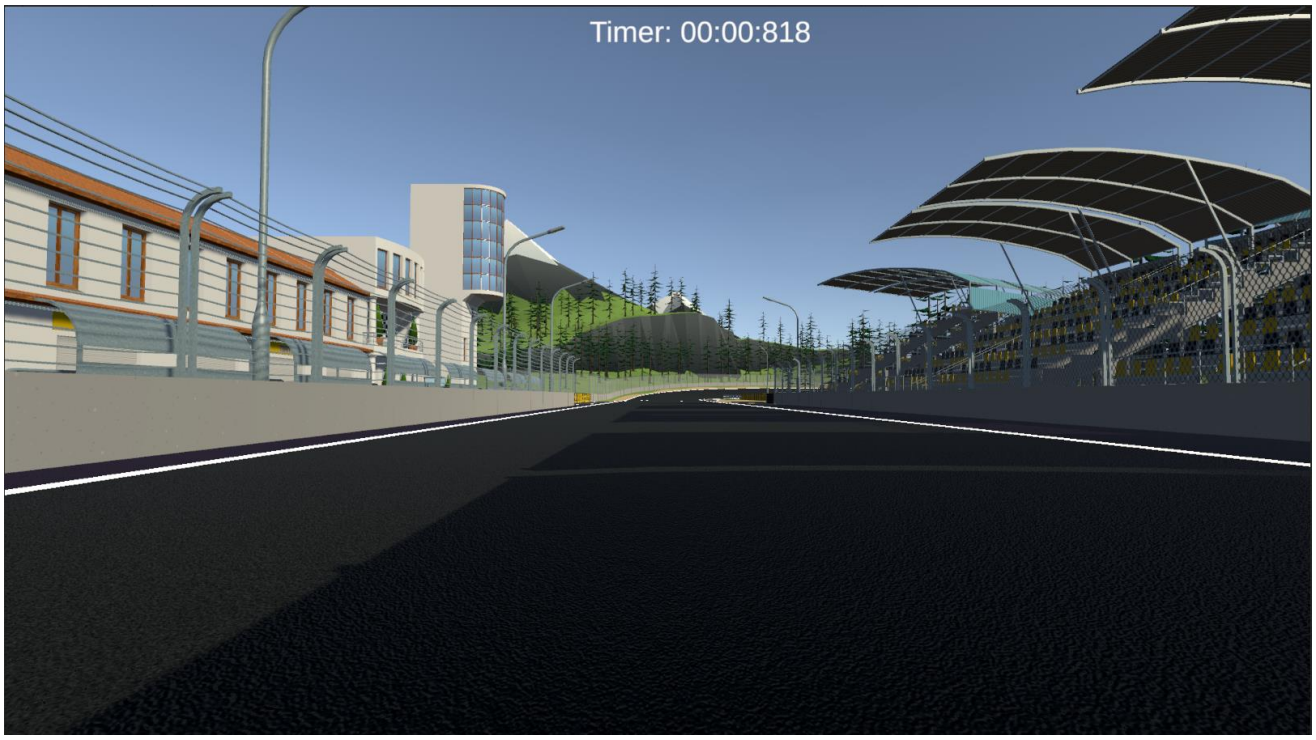

Рисунок Д.11 – Загальний вигляд інтерфейсу початку гри

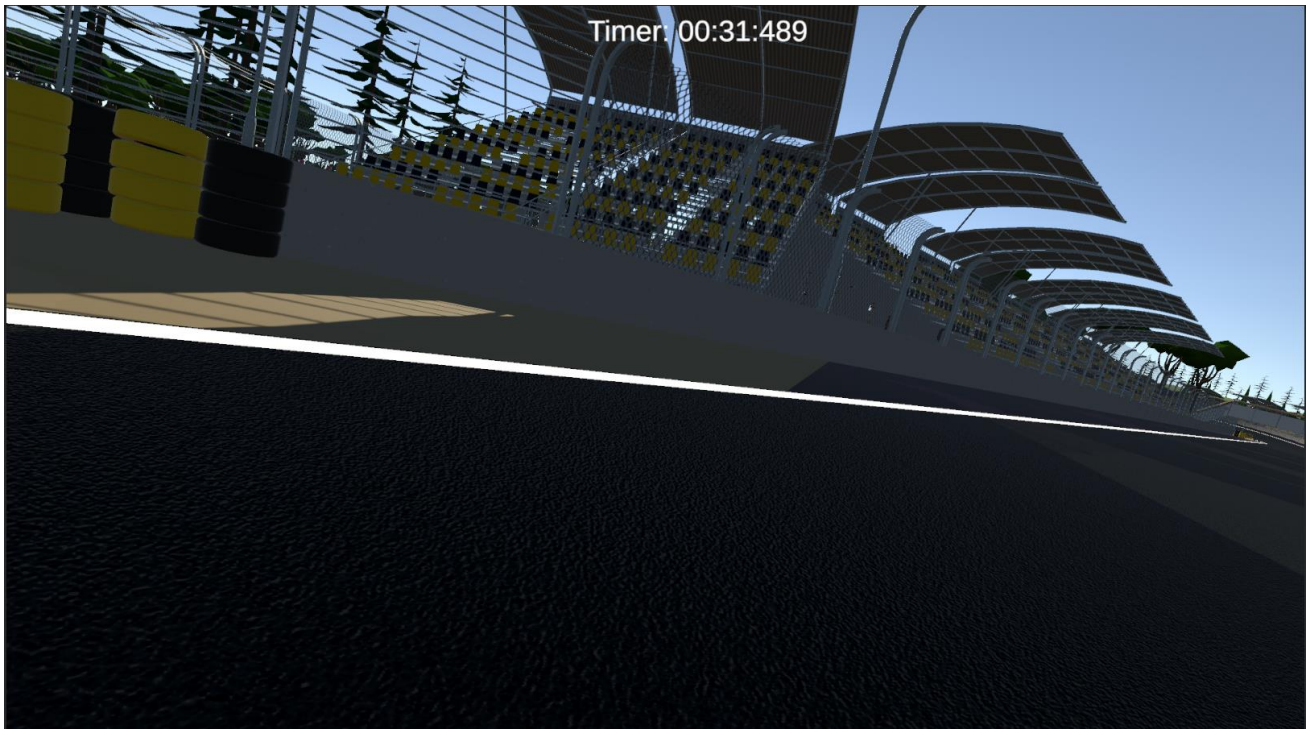

Рисунок Д.12 – Загальний вигляд інтерфейсу при повороті наліво

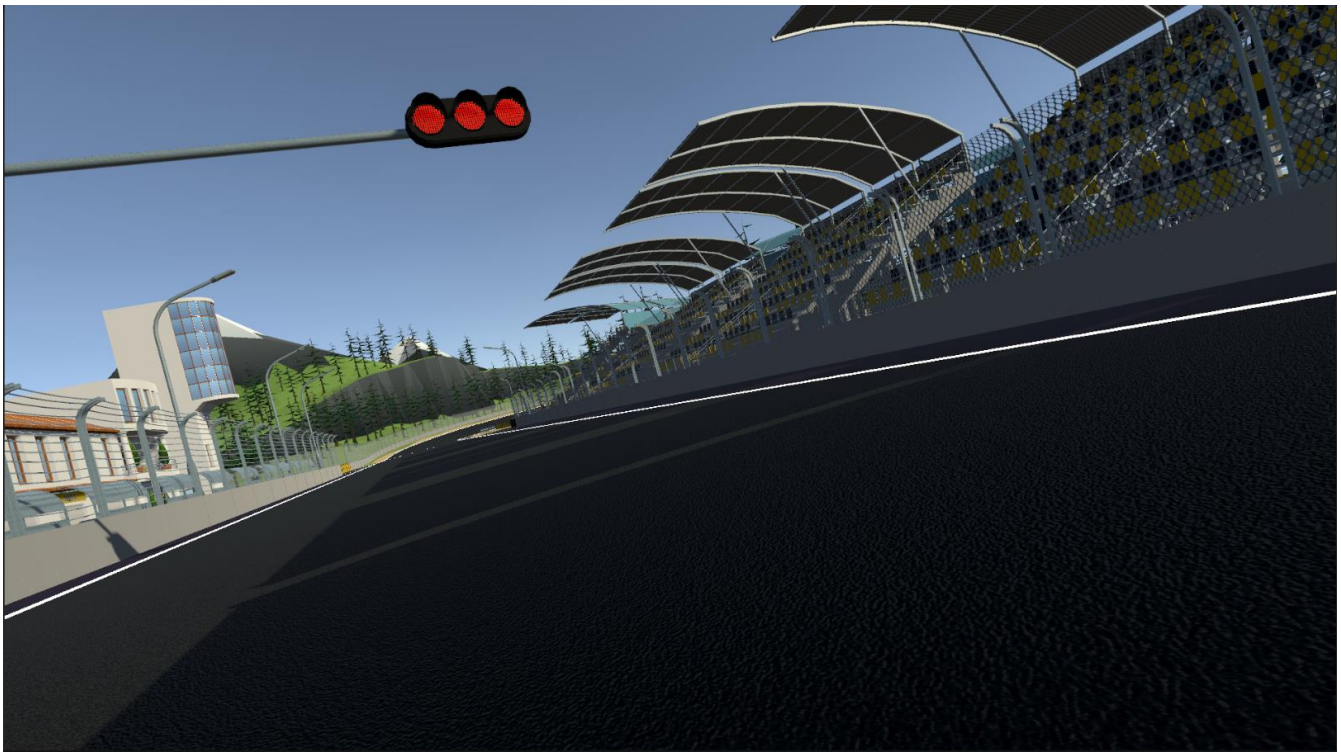

Рисунок Д.13 – Загальний вигляд інтерфейсу при повороті направо

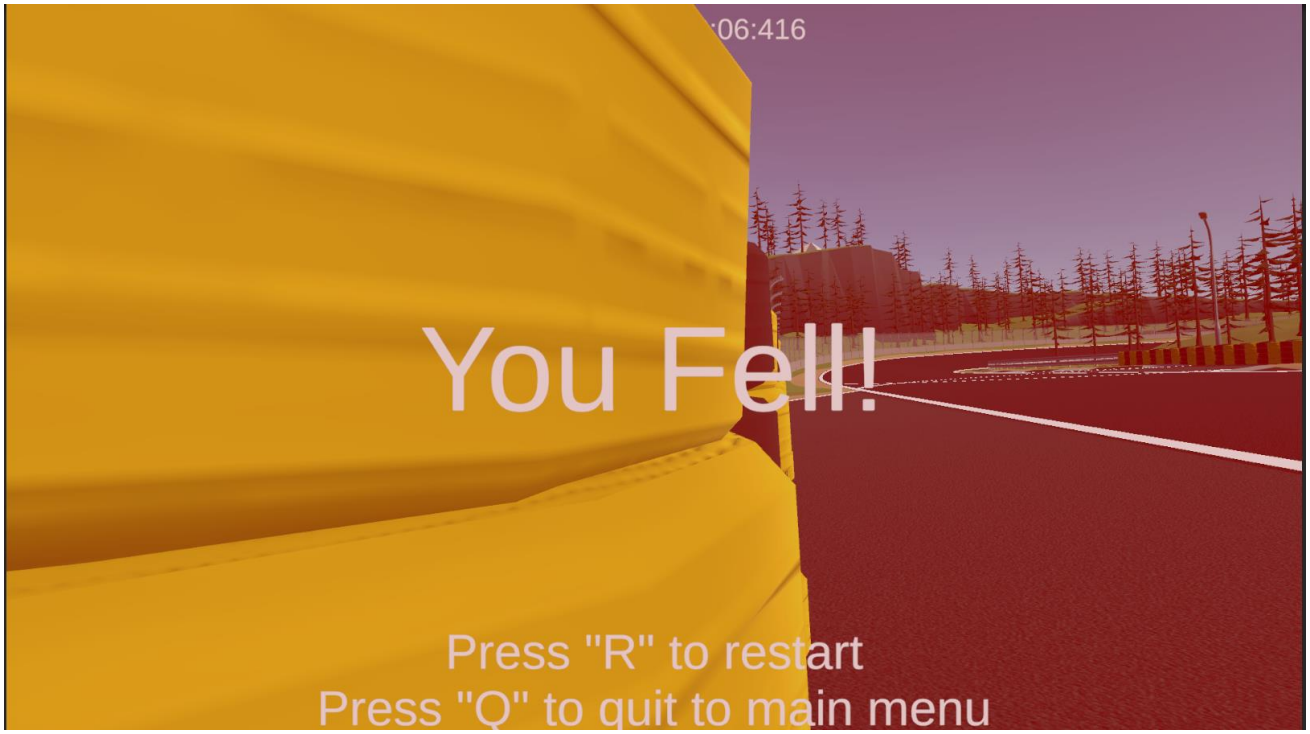

Рисунок Д.14 – Загальний вигляд інтерфейсу після поразки гравця

Рисунок Д.15 – Загальний вигляд інтерфейсу після перемоги гравця