МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Чорноморський національний університет імені Петра Могили

Факультет комп'ютерних наук

Кафедра інженерії програмного забезпечення

ДОПУЩЕНО ДО ЗАХИСТУ
Завідувач кафедри_____ Є. О. Давиденко
підпис
«___» _____2024 р.

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

Розробка ігрового застосунку в жанрі клікер на рушії Unity

Спеціальність «Інженерія програмного забезпечення»

121 – КРБ.1 – 408.22010816

Здобувач _____М. А. Скиба
підпис
«__» _____ 2024 р.

Керівник PhD, ст. викладач _____ І. О. Кандиба
підпис
«__» _____ 2024 р.

Консультант канд. техн. наук, доцент _____ А. О. Алексєєва
підпис
«__» _____ 2024 р.

Миколаїв – 2024 рік

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Чорноморський національний університет імені Петра Могили

Факультет комп'ютерних наук

Кафедра інженерії програмного забезпечення

ЗАТВЕРДЖУЮ

Зав. кафедри Інженерії

Програмного Забезпечення

_____ Є. О. Давиденко

« 22» ___грудня___ 2023 р.

ЗАВДАННЯ

на виконання кваліфікаційної роботи бакалавра

Видано здобувачу групи 408 факультету комп'ютерних наук

_____Скибі Миколі Андрійовичу_____

(прізвище, ім'я, по батькові здобувача)

1. Тема кваліфікаційної роботи

_____Розробка ігрового застосунку в жанрі клікер на рушії Unity _____

Затверджена наказом по ЧНУ від «22» грудня 2023 р. № 269

2. Строк представлення кваліфікаційної роботи «____» _____ 20__ р.

3. Очікуваний результат роботи та початкові дані, якщо такі потрібні

_____Очікуваним результатом є ігровий застосунок у жанрі клікер на основі рушія Unity2D_____

4. Перелік питань, що підлягають розробці

_____Аналіз предметної галузі, аналіз механік до ігрового застосунку, виконати проектування ігрового застосунку, реалізувати ігровий застосунок, протестувати розроблений ігровий застосунок_____

5. Перелік графічних матеріалів

Презентація

6. Завдання до спеціальної частини

_____

_____

_____

7. Консультанти:

Консультант	Кафедра (організація) Частина роботи	Кафедра (організація) Частина роботи
А. О.Алексєєва	Кафедра екології	Спеціальна частина з охорони праці

Керівник роботи ___PhD, ст. викладач Кандиба Ігор Олександрович_____

(посада, прізвище, ім'я, по батькові)

_____

(підпис)

Завдання прийнято до виконання

_____Скиба Микола Андрійович_____

(прізвище, ім'я, по батькові здобувача)

_____

(підпис)

Дата видачі завдання «22» ____грудня____ 2023 р.

КАЛЕНДАРНИЙ ПЛАН

виконання кваліфікаційної роботи

Тема: «Розробка ігрового застосунку в жанрі клікер на рушії Unity»

№	Найменування роботи	Початок	Закінчення	Примітки
1.	Розробка та затвердження завдання на виконання КРБ	21.12.23	22.12.23	Виконано
2.	Огляд літератури за темою роботи	04.01.23	04.01.23	Виконано
3.	Складання календарного плану КРБ	15.01.23	16.01.23	Виконано
4.	Аналіз предметної області	17.01.24	18.01.24	Виконано
5.	Розробка проєктних рішень	22.01.24	30.01.24	Виконано
6.	Моделювання та конструювання ПЗ	22.01.24	30.01.24	Виконано
7.	Кодування, тестування та апробація розробленого ПЗ	22.01.24	30.02.24	Виконано
8.	Розробка спеціальної частини з охорони праці	26.04.24	26.04.24	Виконано
9.	Оформлення КРБ та презентації	04.04.24	16.04.24	Виконано
10.	Відгук керівника КРБ	08.05.24	08.05.24	Виконано
11.	Попередній захист	03.06.24	05.06.24	Виконано
12.	Рецензування	15.06.24	18.06.24	Виконано
13.	Захист кваліфікаційної роботи	26.06.24	26.06.24	Виконано

Розробив здобувач Скиба Микола Андрійович _____

(прізвище, ім'я, по батькові) *(підпис)*

«15» січня 2024 р.

Керівник роботи PhD, ст. викладач Кандиба Ігор Олександрович _____

(посада, прізвище, ім'я, по батькові) *(підпис)*

«16» січня 2024 р.

АНОТАЦІЯ

до кваліфікаційної роботи бакалавра

«Розробка ігрового застосунку в жанрі клікер на рушії Unity»

Здобувач 408 гр.: Скиба Микола Андрійович

Керівник: PhD, ст. викладач Кандиба Ігор Олександрович

Кваліфікаційна робота присвячена розробці ігрового застосунку в жанрі клікер на рушії Unity.

Актуальність кваліфікаційної роботи полягає в тому, що на сьогоднішній день гравці стикаються зі значною стагнацією на ринку, де переважають перевидання, ремастери та продовження вже існуючих франшиз. Повна ігрова графа заповнена бета-версіями, які часто виходять до завершення полірування геймплею. Багато ігрових застосунків подібних жанрів і всесвітів, що призводить до загального втомлення гравців від монотонності та недосконалості.

Об'єктом кваліфікаційної роботи є ігровий застосунок у жанрі клікер на базі рушія Unity 2D.

Предметом кваліфікаційної роботи є інструментарій для розробки ігрового застосунку у жанрі клікер на базі рушія Unity 2D.

Метою кваліфікаційної роботи є розробка ігрового застосунку у жанрі клікер на рушії Unity 2D, для демонстрації навичок розробки програмного забезпечення.

Для досягнення цієї мети необхідно вирішити наступні завдання:

1) дослідити особливості жанру клікер та провести аналіз сучасних мобільних ігор цього жанру;

2) проаналізувати можливості рушія Unity для розробки мобільних ігор та використання його функцій для реалізації особливостей гри в жанрі клікер;

3) спроєкутвати гру в жанрі клікер;

4) розробка дизайну гри, включаючи геймплей, рівні, персонажів та ефекти;

5) реалізація гри на рушії Unity та тестування її функцій та ефективності.

У першому розділі було розглянуто різні жанри ігрових застосунків, їх особливості та переваги та недоліки. На завершення цього розділу була складена специфікація вимог для мобільної гри у жанрі клікер на рушії Unity.

У другому розділі було проведено створення та аналіз діаграм, які стали важливим інструментом для аналізу та моделювання взаємодії між системою та користувачами.

У третьому розділі був проведений аналіз ключових інструментів та технологій для розробки ігор, зокрема .NET (з мовою програмування C#), Visual Studio та Unity.

У четвертому розділі було розроблено 2D ігрового застосунку у жанрі клікер. Він включав аналіз існуючих рішень, дизайн програмного забезпечення, створення інтерфейсу користувача, розробку ігрових елементів, вибір рушія Unity 2D. Це дозволило досягти поставленої мети проекту та створити функціональну та привабливу гру.

КРБ викладена на 85 сторінки, вона містить 4 розділи, 53 ілюстрацій, 4 таблиці, 23 джерел в переліку посилань.

Ключові слова: ігровий застосунок, клікер, звукове супроводження, гравці, unity.

ABSTRACT

to the bachelor's qualification work

«Development of a game application in the genre of clicker on the Unity engine»

Student of group 408: Skyba Mykola Andriyovych

Supervisor: PhD, senior lecturer Kandyba Igor Oleksandrovych

Qualification work is devoted to the development of a game application in the genre of clicker on the Unity engine.

The **relevance** of the qualification work lies in the fact that today players are facing significant stagnation in a market dominated by re-releases, remasters, and sequels to existing franchises. The full game graph is filled with beta versions that are often released before the gameplay is polished. Many gaming applications are of similar genres and universes, which leads to the general fatigue of players from monotony and imperfection.

The **object** of the qualification work is a clicker game application based on the Unity 2D engine.

The **subject** of the qualification work is a toolkit for developing a clicker game application based on the Unity 2D engine.

The **purpose** of the qualification work is to develop a clicker game application based on the Unity 2D engine to demonstrate software development skills.

To achieve this goal, the following tasks need to be solved:

1) to study the features of the clicker genre and analyze modern mobile games of this genre;

2) to analyze the capabilities of the Unity engine for developing mobile games and using its functions to implement the features of the clicker game;

3) to design a game in the clicker genre;

4) develop game design, including gameplay, levels, characters, and effects;

5) implement the game on the Unity engine and test its functions and efficiency.

The first chapter discussed different genres of gaming applications, their features, advantages, and disadvantages. At the end of this chapter, a specification of requirements for a clicker mobile game on the Unity engine.

In the second section, we created and analyzed diagrams, which became an important tool for analyzing and modeling the interaction between the system and users.

The third chapter analyzed key tools and technologies for game development, including .NET (with C# programming language), Visual Studio, and Unity.

In the fourth chapter, we developed a 2D clicker game application. It included analysis of existing solutions, software design, creation of the user interface, development of game elements, and selection of the Unity 2D engine. This allowed us to achieve the project goal and create a functional and attractive game.

The bachelor’s qualification work is set out on 85 pages, it contains 4 sections, 53 illustrations, 4 tables, 23 sources in the list of references.

Keywords: game application, clicker, sound, players, unity.

ЗМІСТ

ВСТУП 4
1 АНАЛІЗ СУЧАСНИХ МОБІЛЬНИХ ІГОР 5
1.1 Жанри мобільних ігор 5
1.2 Аналіз аналогів 14
1.3 Специфікація вимог до мобільної гри в жанрі Clicker 18
Висновки до розділу 1 19
2 МОДЕЛЮВАННЯ ТА ПРОЄКТУВАННЯ МОБІЛЬНОЇ ГРИ В ЖАНРІ КЛІКЕР 20
2.1 Створення діаграм прецедентів 20
2.2 Побудова діаграм класів 21
2.3 Алгоритм роботи програмного забезпечення 23
2.4 Діаграми станів та переходів 24
2.5 Сценарії використання 26
2.6 Діаграми діяльності 28
Висновки до розділу 2 31
3 ІНСТРУМЕНТАЛЬНІ ЗАСОБИ РОЗРОБКИ КОМП'ЮТЕРНОЇ ГРИ 32
3.1 Платформа .NET та мова програмування C# 32
3.2 Середовище розробки Visual Studio 34
3.3 Рушій для розробки відеоігор Unity 35
3.4 Обґрунтування вибору платформи 37
3.5 Ознайомлення із базовим функціоналом Unity 3D 38
3.6 Ознайомлення із базовим функціоналом Unity 3D 39
3.7 Тестування та методика тестування ігор 41
Висновки до розділу 3 43
4 ПРОГРАМНА РЕАЛІЗАЦІЯ ІГРОВОГО ЗАСТОСУНКУ 44
4.1 Реалізація головного меню 44
4.2 Реалізація привітальних панелей 48
4.3 Реалізація ворогів 51
4.4 Реалізація рівнів та босів 53
4.5 Реалізація магазину покращень 54

4.6 Реалізація паузи 56
4.7 Реалізація івентів та метод WordNotation 58
4.8 Реалізація збереження в грі 59
4.9 Реалізація анімацій 60
Висновки до розділу 4 62
ВИСНОВКИ 63
ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ 64
ДОДАТОК А 66

ВСТУП

Актуальність кваліфікаційної роботи полягає в тому, що на сьогоднішній день гравці стикаються зі значною стагнацією на ринку, де переважають перевидання, ремастери та продовження вже існуючих франшиз. Повна ігрова графа заповнена бета-версіями, які часто виходять до завершення полірування геймплею. Багато ігрових застосунків подібних жанрів і всесвітів, що призводить до загального втомлення гравців від монотонності та недосконалості.

Завдяки потужності Unity, розробники можуть створити ігровий досвід, який поєднує в собі простоту управління з глибокою системою прогресу та візуальною привабливістю.

Об’єктом кваліфікаційної роботи є ігровий застосунок у жанрі клікер на базі рушія Unity 2D.

Предметом кваліфікаційної роботи є інструментарій для розробки ігрового застосунку у жанрі клікер на базі рушія Unity 2D.

Метою кваліфікаційної роботи є розробка ігрового застосунку у жанрі клікер на рушії Unity 2D, щоб продемонструвати навички розробки програмного забезпечення.

Для досягнення цієї мети необхідно вирішити наступні завдання:

1) дослідити особливості жанру клікер та провести аналіз сучасних мобільних ігор цього жанру;

2) проаналізувати можливості рушія Unity для розробки мобільних ігор та використання його функцій для реалізації особливостей гри в жанрі клікер;

3) спроєкутвати гру в жанрі клікер;

4) розробка дизайну гри, включаючи геймплей, рівні, персонажів та ефекти;

5) реалізація гри на рушії Unity та тестування її функцій та ефективності.

1 АНАЛІЗ СУЧАСНИХ МОБІЛЬНИХ ІГОР

1.1 Жанри мобільних ігор

Сучасні мобільні ігри відіграють важливу роль як у економічному, так і в культурному контекстах. Вони стали необхідною складовою індустрії розваг та креативного сектора економіки. Однак, естетика та технології, що застосовуються у мобільних іграх, знаходять застосування в різних сферах життя. Вони допомагають у процесі навчання дітей, молоді та дорослих як у навчальних закладах, так і на робочому місці, забезпечуючи цікавий спосіб освоєння складного матеріалу. Крім того, мобільні ігри можуть бути використані для мотивації пацієнтів у медичних та терапевтичних закладах, створюючи сприятливе середовище для досягнення позитивних результатів. Таким чином, мобільні ігри виходять за межі простого розважального середовища й знаходять широке застосування у різних аспектах життя.

У світі мобільних ігор існує безліч різноманітних жанрів, які відображають різноманітні стилі, геймплей та ігрові механіки. Від захоплюючих рольових пригод і епічних стратегій до швидких аркад і складних головоломок - мобільні ігри пропонують безліч можливостей для розваг та відпочинку. Залежно від настрою, користувач може відчути адреналін у гонках, вдосконалити свою логіку в головоломках або створити власний світ у стратегічних іграх. До основних жанрів мобільних ігор можна віднести рольові ігри, головоломки і логічні ігри, аркади, гонки і спортивні ігри, стратегії, а також клікери та ігри в жанрі екшен[1-3].

Жанр рольових ігор (RPG) транспортує гравців у захопливий світ фантазії та пригод. У цих іграх вони можуть відчути себе у ролі героїв, що ведуть епічні кампанії, розвиваються та змагаються з різноманітними ворогами. RPG надають можливість вибирати персонажа, його характеристики, зовнішній вигляд та навички. У цьому жанрі кожне рішення гравця має значення, він може взаємодіяти з іншими персонажами та приймати

рішення, які впливають на сюжет та розвиток історії. Захопливі квести, завдання та діалоги розкриваються по ходу гри, дозволяючи гравцю повністю погрузитися у іммерсивну атмосферу ігрового світу. У RPG важливо розвивати персонажа, збираючи досвід та виконуючи завдання, що дозволяє покращити навички, отримати нове спорядження та розкривати нові можливості. Приклади мобільних ігор у жанрі RPG: «The Elder Scrolls: Blades» (рис 1.1), «Final Fantasy XV Pocket Edition», «Summoners War», «Star Wars: Knights of the Old Republic».

Рисунок 1.1 – Інтерфейс відеогри The Elder Scrolls: Blades

Жанр головоломок та логічних ігор на мобільних пристроях дозволяє гравцям стати справжніми майстрами розумових викликів. Ці ігри переносять їх у захоплюючі лабіринти складних загадок, головоломок та завдань, які вимагають логічного мислення, кмітливості та творчого підходу. У цьому жанрі представлено широкий спектр ігор, від класичних головоломок, таких як «тетріс» та «судоку», до більш складних ігор з унікальними геймплейними механіками. Гравець повинен розв’язувати логічні загадки, збирати пазли, шукати сховані об’єкти або розв’язувати криптографічні головоломки, щоб просуватися в грі. Ці ігри не лише цікаві, але й корисні, оскільки вони

розвивають креативність, аналітичні здібності, логічне мислення та стратегічне планування. Приклади мобільних ігор у цьому жанрі: «The Room», «Monument Valley», «The Witness», «Cut the Rope», «Lara Croft GO» (рис 1.2).

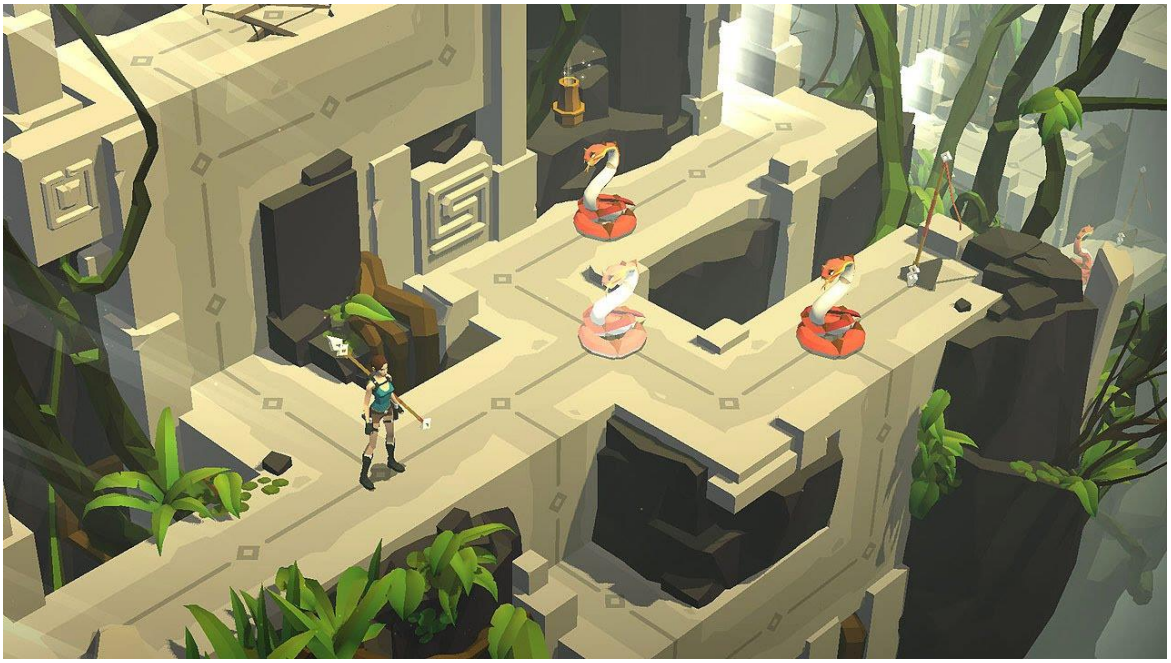


Рисунок 1.2 – Інтерфейс відеогри Lara Croft GO

Аркадні ігри – це категорія мобільних ігор, які призначені для швидких геймплейних сесій з простими правилами та ігровим процесом. Початково цей жанр зародився у ігрових залах, де гравці змагалися за рекордні результати, насолоджуючись адреналіном від суперечок. Сучасні аркадні ігри дуже популярні серед користувачів мобільних платформ та мають широку аудиторію. Суттєвими рисами цього жанру є швидкість геймплею, простота правил та елементи, які можуть змінюватися в кожній грі. Асортимент аркадних ігор досить різноманітний - від класичних «арканоїдів» та «тетрісів» до сучасних раннерів і файтингів. Деякі приклади мобільних ігор у жанрі аркад: Angry Birds (рис 1.3), Fruit Ninja, Candy Crush, Flappy Bird, Pac-Man.

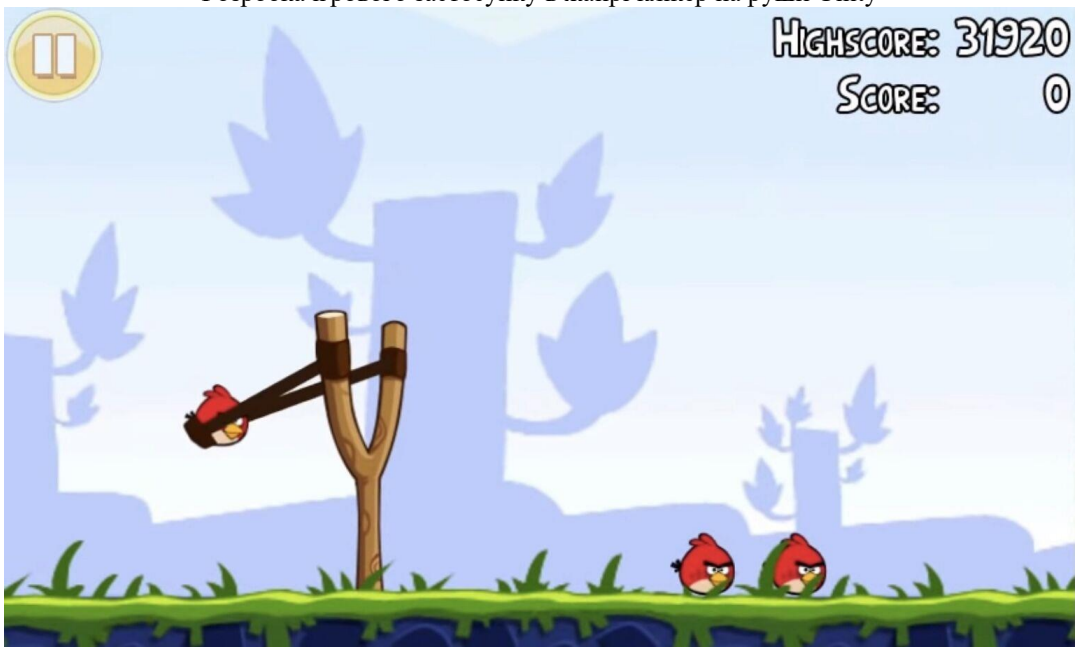


Рисунок 1.3 – Інтерфейс відеогри Angry Birds

Жанр гонок і спорту в мобільних іграх завжди надає піднесення та адреналін. Він дозволяє гравцям відчути швидкість, напруженість змагань і радість перемоги, усе це на їх долонях. У цьому захоплюючому жанрі гравець може перевірити свої навички, ставши гонщиком на найшвидших трасах, керуючи автомобілями, мотоциклами або навіть велосипедами. Жвава графіка та реалістична фізика дозволяють гравцеві відчути кожну криву, кожен скрутний поворот і швидкісний розгін. Аркадні гонки пропонують гравцям шалений швидкісний досвід, використовуючи різноманітні механіки для змагань за перші місця та побиття рекордів. Гравець може випробувати свою реакцію та швидкість на трасах, що переповнені різноманітними перешкодами. Крім того, жанр гонок і спорту включає ігри, які симулюють різні види спорту, такі як футбол, бейсбол, баскетбол і навіть серфінг. Гравець може стати членом команди, взяти участь у змаганнях і використовувати свої навички для досягнення перемоги. Деякі приклади мобільних ігор у жанрі гонки і спорт: Asphalt 9: Legends (рис 1.4), FIFA Mobile Football, NBA 2K Mobile Basketball.

Рисунок 1.4 – Інтерфейс відеогри Asphalt 9: Legends

Жанр стратегій у мобільних іграх розкриває перед гравцями широкий спектр можливостей та вимагає стратегічного мислення, планування і управління ресурсами. У цьому жанрі гравець може стати володарем власної імперії, будувати та розвивати міста, керувати армією або керувати економікою цілого народу. У світі стратегій гравець має вибір різних сценаріїв від середньовічних воєн і фантастичних світів до сучасних політичних конфліктів. Кожна гра в цьому жанрі має свій унікальний підхід та геймплей. Гравець може відчути себе в ролі великого полководця, який веде свою армію до перемоги, або відправитися у подорож у світ політичних інтриг і дипломатії. Деякі приклади мобільних ігор у жанрі стратегії: Clash of Clans (рис 1.5), Civilization VI, Plague Inc, XCOM: Enemy Within.

Рисунок 1.5 – Інтерфейс відеогри Clash of Clans

Жанр екшн у мобільних іграх пропонує шалений ритм, захоплюючі пригоди та безліч екшну. Це жанр, що ставить перед гравцем виклик у вигляді швидких реакцій, неймовірних відчуттів та боротьби зі світом, повним небезпек і ворогів. У світі ігор екшн гравець може стати справжнім героєм, який вирушає на місію порятунку, протистояти злим силам або відправитися на війну. Гравцеві належить боротися з численними ворогами, використовуючи різноманітну зброю - від ножів і пістолетів до магічних заклять і суперсил. Гравець має пройти безліч рівнів, виконувати завдання і виявити свою вправність у битві. У мобільних іграх екшн гравець може погрузитися у світ небезпечних пригод, використовуючи інтуїцію, стратегічне мислення і швидкі рефлекси. Він може відчути себе справжнім героєм, який протистоїть всім випробуванням і небезпекам, що трапляються на його шляху. Деякі приклади мобільних ігор у жанрі екшн: Shadow Fight 3, PUBG Mobile (рис 1.6), Call of Duty Mobile, Injustice 2.

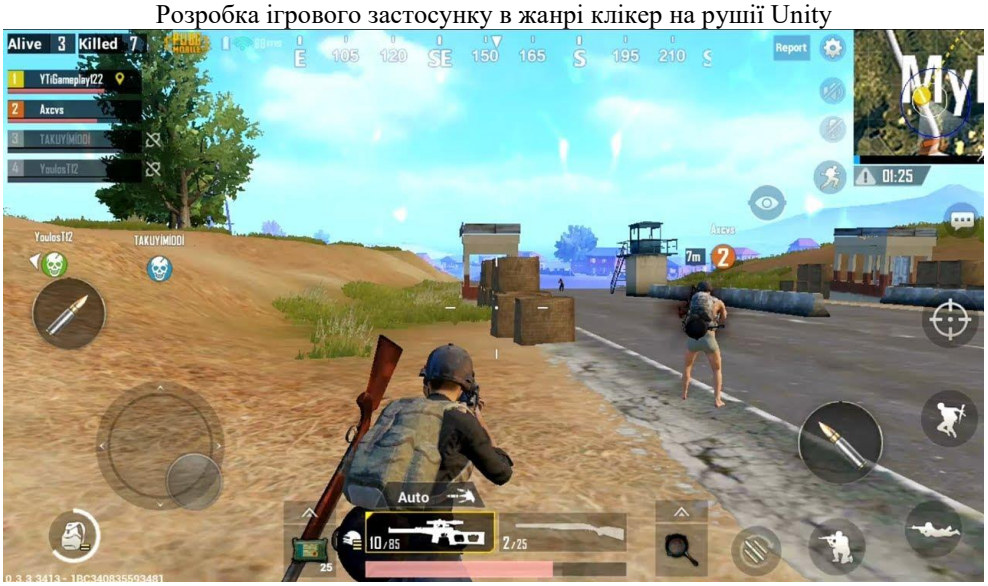


Рисунок 1.6 – Інтерфейс відеогри PUBG Mobile

Окрім цього, існує класифікація відеоігор за віковими категоріями. В залежності від віку гравця визначається відповідний рівень складності гри, її тематика та візуальний стиль. Зазвичай ігри розділяють на категорії, які зібрані в Таблиці 1.1[4]:

Таблиця 1.1 – Розподіл ігор за віковими категоріями

Категорія	Вік	Опис
EC	Для дітей молодшого віку	Гра підходить для дітей від 3 років і не містить матеріалів, які батьки могли б вважати невідповідними. Продукти, що отримали цей рейтинг, спочатку розробляються для дітей і зазвичай є розвиваючі ігри
E	Для всіх	Зміст цілком підходить для будь-якого віку. Такі ігри можуть сподобатися і дорослим. Ігри з цим рейтингом можуть містити мінімальне насильство, переважно мультиплікаційного характеру, без жорстокості

Кінець таблиці 1.1 – Розподіл ігор за віковими категоріями

E10+	Для всіх від 10 років і старше	Проєкти з даним рейтингом можуть містити більше мультиплікаційного або м’якого насильства, або дещо відверті сцени, або мінімальну кількість крові
T	Підліткам	Гра підходить для осіб від 13 років. Проєкти цієї категорії можуть містити насильство, непристойні сцени, грубий гумор, в міру відвертий сексуальний вміст, кров або нечасте використання ненормативної лексики
M	Для дорослих	Матеріали гри не підходять для осіб молодше 17 років. Можуть містити жорстоке насильство, велику кількість крові з розчленуванням, непристойні сексуальні сцени чи грубу ненормативну лексику
AO	Тільки для дорослих	Зміст гри тільки для дорослих старше 18 років. Продукти цієї категорії можуть містити тривалі сцени жорстокого насильства та/або дуже відвертий сексуальний вміст

Важливо також ураховувати різноманітні жанри відеоігор при їх класифікації. Наприклад, у рольових іграх гравець уособлює персонажа, який виконує різноманітні завдання, взаємодіє з іншими персонажами та розвиває свої навички. У шутерах головна мета полягає у знищенні ворогів за допомогою зброї. У пригодницьких іграх гравець розв’язує різноманітні завдання, взаємодіє з персонажами та розв’язує головоломки.

Відеоігри також можна класифікувати за рівнем відкритості світу. У відкритих світах гравець може вільно переміщатися, виконувати завдання та взаємодіяти з іншими персонажами. У лінійних іграх гравець має обмежений вибір дій та можливостей переміщення.

Залежно від способу гри, відеоігри можуть бути класифіковані як одиночні або мультиплеєрні. У одиночних іграх гравець грає сам, зазвичай виконуючи завдання або проходячи історію. У мультиплеєрних іграх гравці можуть грати разом, взаємодіючи та конкуруючи один з одним.

Також варто відзначити класифікацію відеоігор за способом керування. У традиційних відеоіграх керування здійснюється за допомогою геймпаду або клавіатури та миші. З появою розумних технологій, таких як голосове та жестове керування, способи керування відеоіграми стають ще різноманітнішими.

Ці класифікації не є жорсткими категоріями, оскільки багато відеоігор можуть належати до декількох категорій одночасно. Проте вони допомагають краще зрозуміти суть гри та обрати ту, яка найбільше відповідає інтересам гравця.

Для розробників класифікація відеоігор важлива, оскільки допомагає зорієнтуватися в аудиторії гри. Для батьків це інструмент для вибору підходящих ігор для їхніх дітей. Для гравців же вона допомагає знайти гру, яка відповідає їхнім уподобанням. Класифікація відеоігор є корисним інструментом для всіх, хто працює в галузі відеоігор або просто любить грати в них, оскільки допомагає краще розуміти суть гри та відібрати ту, яка найбільше відповідає потребам гравця.

1.2 Аналіз аналогів

Clicker Heroes – це гра, розроблена американською незалежною студією Playsaurus (рис 1.7). Спочатку вона була випущена для браузерів у 2014 році, для мобільних пристроїв у 2015 році та для консолей Xbox One і PlayStation 4 у 2017 році. Гра є спін-оффом попередньої гри Playsaurus Cloudstone, з якої вона використовує багато графічних елементів.[5]

Таблиця 1.2 – Характеристика гри Clicker Heroes

Назва характеристики	Опис
Назва	Clicker Heroes
Розробник	Playsaurus
Архітектура	Unity
Мова реалізації	C#
Перелік функцій, характеристик	– натискання на екран для збору ресурсів; – автоматичне збирання ресурсів з часом; – покращення персонажів та їхніх навичок; – велика кількість персонажів та можливостей для покращення; – різноманітність ворогів та босів.
Аналіз переваг	– простий геймплей, легкий для вивчення; – велика кількість персонажів та можливостей для покращення; – різноманітність ворогів та босів.
Аналіз недоліків	– може швидко стати монотонним; – не дуже глибока стратегічна складова.

Рисунок 1.7 – Інтерфейс відеогри Clicker Heroes

Cookie Clicker – це одна з найпопулярніших ігор жанру «інкрементальних ігор» або «ігор клікера» (рис 1.8). Головна мета гри - натискати на кнопку, щоб виробляти печиво. За кожне печиво гравець отримує гроші, які можна витрачати на покупку різноманітних покращень, які збільшують виробництво печива автоматично або за кліки.[6]

Таблиця 1.3 – Характеристика гри Cookie Clicker

Назва характеристики	Опис
Назва	Cookie Clicker
Розробник	Orteil
Архітектура	HTML5
Мова реалізації	JavaScript
Перелік функцій, характеристик	– натискання на печиво для збільшення кількості печива; – купівля автоматичних систем генерації печива; – розблокування нових видів печива та досягнень; – можливість налаштування та розширення гри за допомогою модів; – графічні ефекти та анімація.

Кінець таблиці1.3 – Характеристика гри Cookie Clicker

Аналіз переваг	– простий, але дуже привабливий геймплей; – необмежена глибина прогресу та розвитку; – можливість налаштування та розширення гри за допомогою модів.
Аналіз недоліків	– може стати монотонним з часом; – графіка та анімація досить прості; – деякі функції можуть виявитися занадто простими.

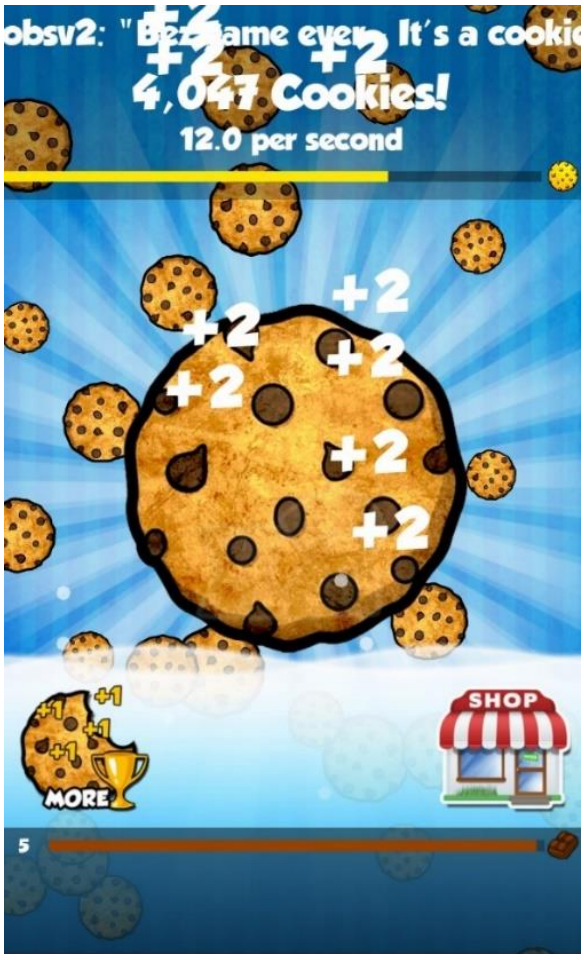


Рисунок 1.8 – Інтерфейс відеогри Cookie Clicker

Spaceplan – це інтерактивна інкрементальна гра, яка відбувається у космічному просторі. Гравець починає з дивана у космічному кораблі, де є тільки комп’ютер, і відправляється в невідомий простір (рис 1.9). Головна мета гри - виживати та розвиватися, використовуючи ресурси та технології, які гравець знаходить у ході гри.[7]

Таблиця 1.4 – Характеристика гри Spaceplan

Назва характеристики	Опис
Назва	Spaceplan
Розробник	Jake Hollands
Архітектура	Unity
Мова реалізації	C#
Перелік функцій, характеристик	– натискання на екран для генерації енергії; – використання зібраної енергії для розвитку космічного корабля; відкриття нових планет та досягнень; – унікальний науково-фантастичний атмосферний світ; – цікавий геймплей та наукові референції.
Аналіз переваг	– унікальний науково-фантастичний атмосферний світ; – цікавий геймплей та наукові референції.
Аналіз недоліків	– гра може бути надто короткою для деяких гравців; – деякі аспекти гри можуть бути важкими для розуміння без попереднього досвіду клікерів.

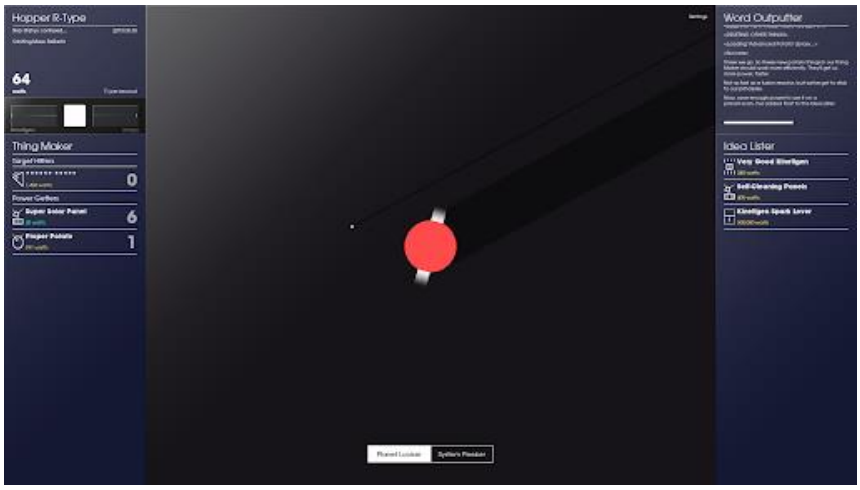


Рисунок 1.9 – Інтерфейс відеогри Spaceplan

Загалом, ігри Clicker Hero, Cookie Clicker і Spaceplan представляють різноманітні аспекти жанру «клікерів» зі своїми унікальними особливостями

та характером. Кожна з них пропонує простий, але привабливий геймплей, хоча має свої переваги і недоліки. Підсумовуючи, вони створюють цікавий ландшафт для шанувальників клікерів, пропонуючи різноманітність відчуттів та можливостей для гравців на різних етапах їхнього розвитку.

1.3 Специфікація вимог до мобільної гри в жанрі Clicker

Призначення системи: створення захоплюючого ігрового середовища, в якому гравці можуть отримувати задоволення від безперервного клікання та покращення своїх можливостей.

Сфера застосування: розваги, відпочинок, розвиток дрібних моторних навичок.

Характеристики користувачів: користувачі будь-якого віку, які мають смартфон або планшет із підтримкою сенсорного введення.

Функції системи:

– система головного меню, що містить список доступних покращень та досягнень;

– система клікання, яка реагує на торкання екрану;

– система автоматичного збору ресурсів;

– система прокачування та покращення виробничих процесів;

– система анімації та візуальних ефектів;

– система магазину для придбання додаткових покращень;

– система збереження прогресу гри.

Вимоги до технічного забезпечення:

– операційна система: Android або iOS;

– процесор: підтримка сенсорних операцій, без особливих вимог до характеристик;

– вільне місце на пристрої: від 100 МБ;

– оперативна пам’ять: від 1 ГБ;

– мінімальна версія ОС: Android 4.4 або iOS 9.0.

Архітектура програмної системи: складається з клієнтської частини.

Системне програмне забезпечення:

– для розробки гри рекомендується використовувати Unity;

– мова програмування: C#;

– відповідність інтерфейсу користувача до стандартів дизайну платформи (Material Design для Android, Human Interface Guidelines для iOS).

Висновки до розділу 1

У розділі 1 було проведено аналіз предметної сфери розробки ігрових застосунків. Було розглянуто різні жанри ігрових застосунків.

Обрано три аналоги гри-застосунку, що були проаналізовані окремо та визначено їх характерні риси, такі як розробник та видавник, мова випуску, перелік функцій та інше. Виокремлено характерні переваги та недоліки ігрових застосунків, що були розглянуті.

У заключній частині розділу 1 сформульовано специфікацію вимог для мобільної гри в жанрі клікер на рушії Unity.

2 МОДЕЛЮВАННЯ ТА ПРОЄКТУВАННЯ МОБІЛЬНОЇ ГРИ В ЖАНРІ КЛІКЕР

2.1 Створення діаграм прецедентів

Діаграма варіантів використання ілюструє способи взаємодії між користувачами або зовнішніми системами та самою системою. Вона демонструє, як користувачі спілкуються з системою, які конкретні дії вони виконують, і як система реагує на їх запити.

Діаграма варіантів використання представлена на рисунку 2.1.

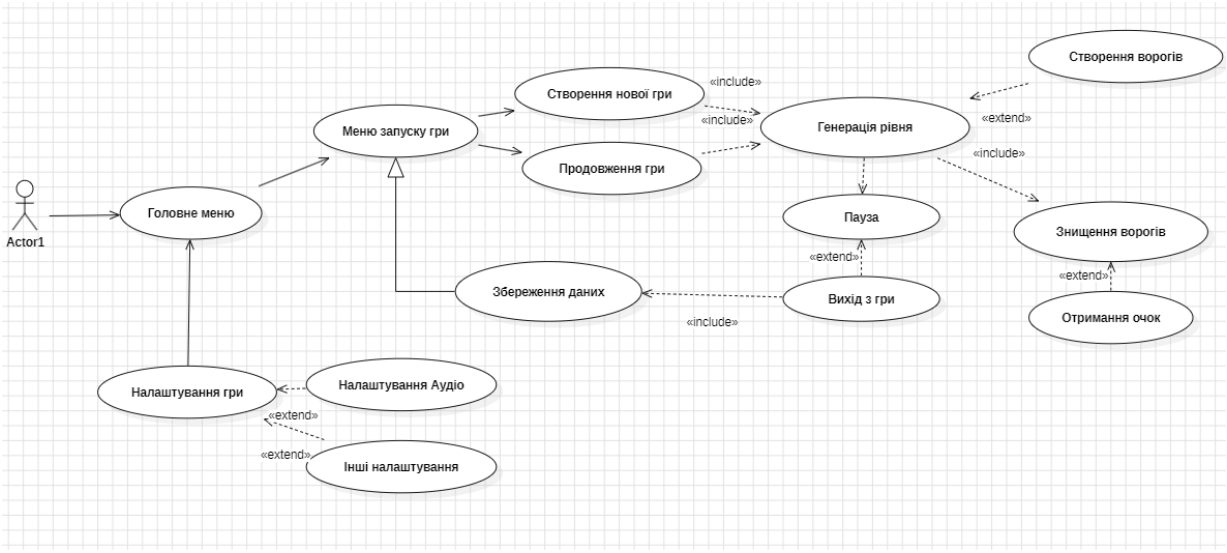


Рисунок 2.1 – Діаграма варіантів використання

Опис елементів моделі на діаграмах варіантів використання:

1) актори – це прості користувачі, які взаємодіють з системою. Користувач може бути тільки людиною;

2) підсистеми – у моделях UML підсистеми існують як тип стереотипних компонентів, поведінкові одиниці в системі [9]. Підсистеми використовуються в діаграмах класів, компонентів і варіантів використання для представлення великомасштабних компонентів у системі яка моделюється;

3) зв’язки та відношення – зв’язки між елементами моделі. Відношення – тип елемента моделі, який додає семантику до моделі, допомогая визначити структуру та поведінку між елементами моделі;

4) випадки використання – описує функцію, яку виконує система для досягнення мети користувача. Випадок використання повинен давати спостережуваний результат, який є цінним для користувача системи.

Гра розпочинається з головного меню, звідки можна перейти в налаштування або у меню створення гри. У меню створення гри, якщо користувач вже грав раніше, він має збережені дані минулих дій і може продовжити грати далі; в іншому випадку йому необхідно створити нову гру. Після запуску гри користувач може клікати на єкран та знищувати ворогів для того, щоб отримувати очки, після чого може придбати в магазині покращення, для більшого отримання очок.

2.2 Побудова діаграм класів

Діаграма класів – це інструмент візуалізації структури та взаємозв’язків між класами у програмному забезпеченні. Вона дозволяє ілюструвати класи разом з їх атрибутами (змінними) та методами, а також зв’язки між класами (рис. 2.2).

Головна мета такої діаграми – спростити розуміння архітектури програми, структури класів та їх залежностей. Вона надає загальний огляд класів та допомагає розробникам зрозуміти, як вони взаємодіють та як вони організовані в системі.

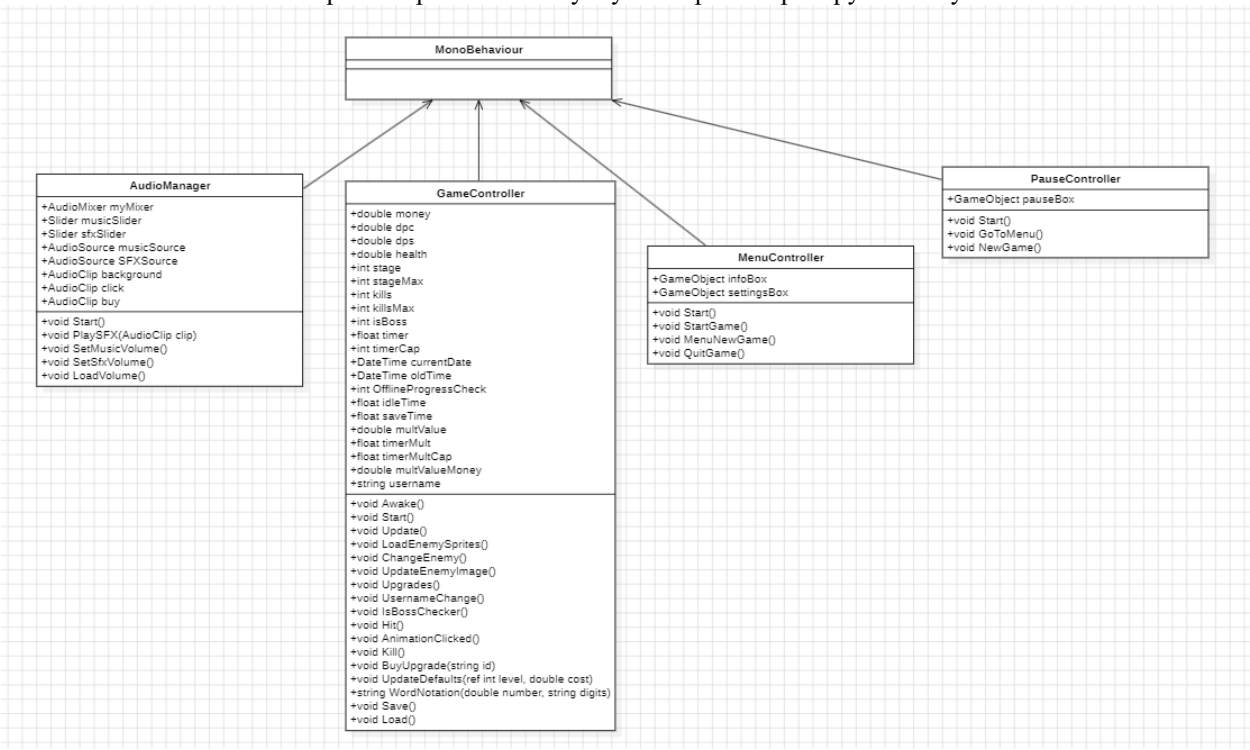


Рисунок 2.2 – Діаграма класів

1) gameController: Цей клас відповідає за керування грою в цілому. Він відстежує стан грошей, здоров’я, рівень інші показники гравця, керує ворогами, апгрейдами, зберігає прогрес гри, обробляє взаємодію з користувачем, відтворює анімацію, контролює звук та багато іншого;

2) audioManager: Цей клас відповідає за управління аудіо в грі. Він відтворює фонову музику, звукові ефекти (наприклад, звук натискання кнопок, звук вбивства ворога), дозволяє гравцю налаштувати гучність музики та звуків;

3) menuController: Цей клас відповідає за керування головним меню гри. Він дозволяє гравцю почати нову гру, завантажити попередні налаштування або вийти з гри. Також він керує відображенням вікон інформації та налаштувань;

4) pauseController: Цей клас відповідає за керування меню паузи. Він забезпечує можливість зупиняти гру, переходити в головне меню або починати нову гру під час паузи.

2.3 Алгоритм роботи програмного забезпечення

Мета створення програмного забезпечення полягає у розробці алгоритму, який описує логіку та послідовність дій, необхідних для досягнення його функціональності та завдань. Головна ціль алгоритму полягає в управлінні та координації операцій та процесів програми, щоб вона могла ефективно функціонувати та виконувати свої завдання.

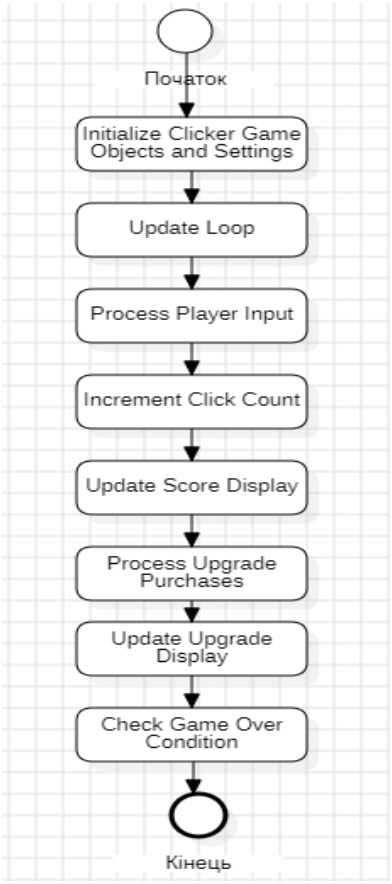


Рисунок 2.3 – Блок-схема роботи програмного забезпечення гри

Ця блок-схема описує основні кроки роботи програми для гри клікера:

1) початок гри: ініціалізація об’єктів та налаштування;

2) оновлення гри: виконання основного циклу оновлення;

3) обробка введення гравця: зчитування кліків гравця;

4) інкрементування лічильника кліків: збільшення кількості кліків гравця;

5) оновлення відображення очків: відображення зміни кількості очків;

6) обробка покупок апгрейдів: дозвіл гравцю придбати апгрейди за накопичені очки;

7) оновлення відображення апгрейдів: відображення доступних апгрейдів;

8) перевірка умови завершення гри: перевірка, чи виконана умова завершення гри;

9) завершення гри: виведення результатів та завершення гри.

Ця блок-схема враховує основні етапи гри клікера, включаючи обробку кліків гравця, підрахунок очків, покупку апгрейдів та перевірку умов завершення гри.

2.4 Діаграми станів та переходів

Діаграма станів та переходів є графічним інструментом, який відображає різні стани, у яких може перебувати система або об’єкт, а також переходи між цими станами. Вона використовується для моделювання та аналізу поведінки системи або об’єкта з точки зору їх станів і переходів між ними.

Основна мета діаграми станів та переходів полягає в тому, щоб візуалізувати та зрозуміти поведінку системи або об’єкта у вигляді послідовності станів та дій, які відбуваються в цих станах. Вона допомагає виявити логіку роботи системи, відобразити можливі стани та переходи між ними, а також виявити можливі проблеми або помилки в логіці системи.

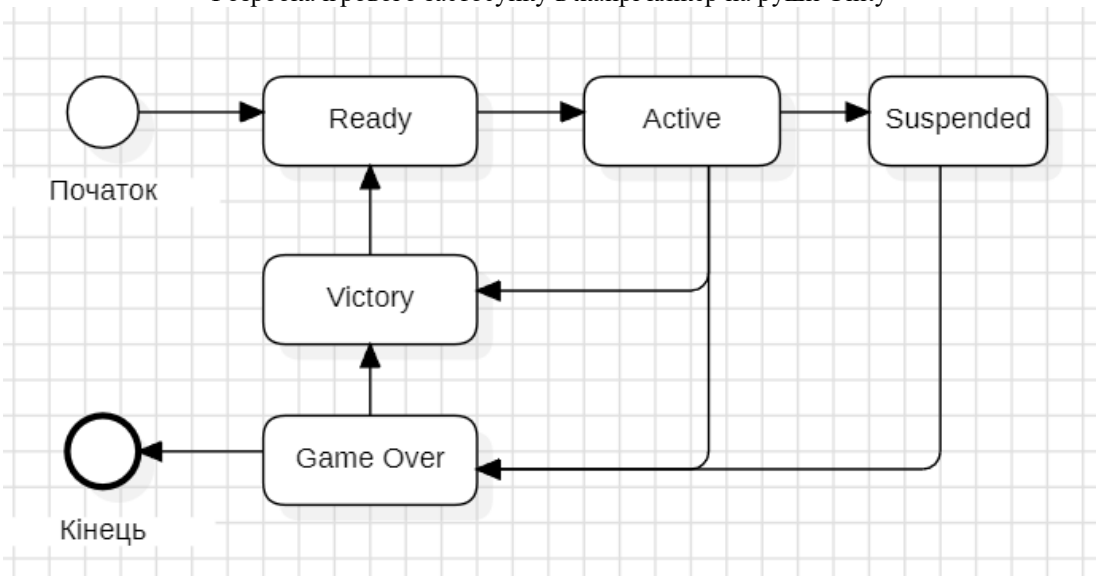


Рисунок 2.4 – Діаграма стану ігрової системи Clicker

1) ready to Active:

Тригер: користувач починає гру, натиснувши кнопку «Ready».

Дія: система активує гру, коли користувач натискає «Ready»;

2) active to Suspended:

Гранична умова: користувач вирішує призупинити гру.

Дія: система призупиняє гру та переходить у стан «Suspended»;

3) active to Victory:

Гранична умова: користувач отримує необхідну кількість балів для перемоги.

Дія: система переходить у стан «Victory»;

4) active to Game Over:

Гранична умова: користувач завершує або виходить з гри.

Дія: система завершує гру та переходить у стан «Game Over».

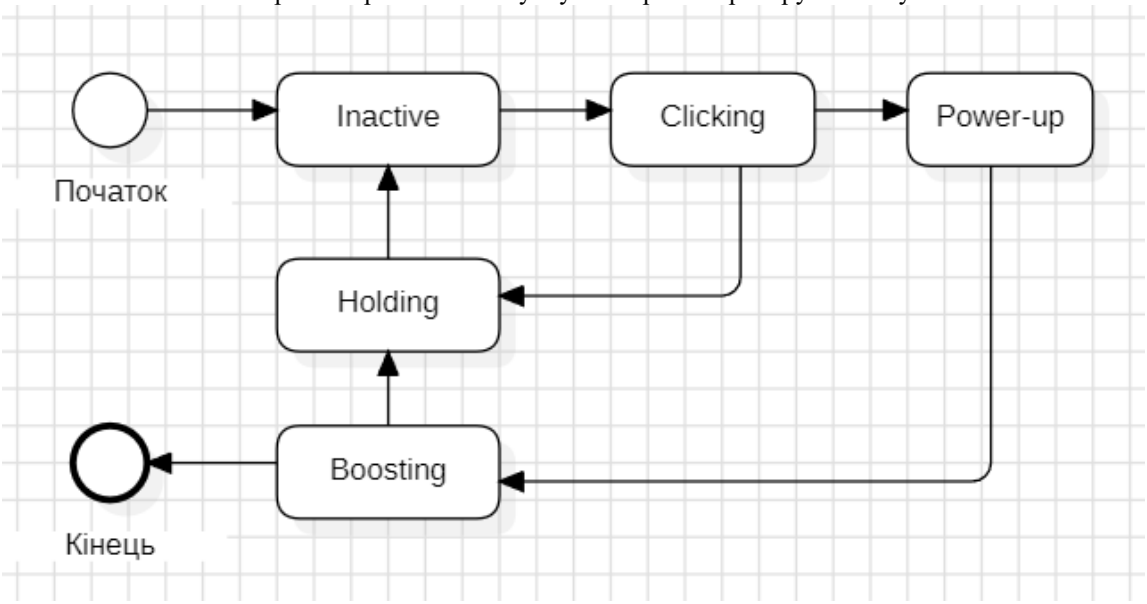


Рисунок 2.5 – Діаграма стану для гравця

1) inactive to Clicking:

Тригер: користувач починає клацати по екрану.

Дія: гравець починає клацати, щоб заробити бали;

2) clicking to Holding:

Гранична умова: користувач припиняє клацати.

Дія: Гравець переходить у стан «Holding»;

3) clicking to Power-up:

Гранична умова: користувач активує підключення.

Дія: Гравець переходить у стан «Power-up»;

4) holding to Clicking:

Тригер: користувач продовжує натискати.

Дія: гравець продовжує натискати, щоб заробити бали;

5) power-up to Clicking:

Тригер: закінчується тривалість увімкнення.

Дія: гравець повертається до звичайного режиму натискання.

2.5 Сценарії використання

Сценарії використання (Use Cases) – це техніка, яка описує, як користувачі взаємодіють з продуктом або системою для досягнення

конкретних цілей. Вони допомагають розробникам та дизайнерам зрозуміти потреби користувачів та спроєктувати інтерфейси, які відповідають цим потребам.

Користувацькі сценарії – це детальні описи, які розповідають про історії користувачів, їхні дії та взаємодію з продуктом. Вони включають образи покупців, їхні мотивації та способи досягнення цілей на сайті чи в застосунку.

Варіанти застосування – це частина методології, яка дозволяє детально описати всі можливі взаємодії між користувачем та системою, включаючи основні та альтернативні потоки дій.

Сценарій 1. Початок гри (рис. 2.6).

Гравець запускає гру та потрапляє в головне меню. Гравець обирає «Нова гра» або «Продовження гри» та починає свою пригоду. Гра пропонує текстовий туторіал, який допомагає зрозуміти основні механіки клікання та заробітку гри.

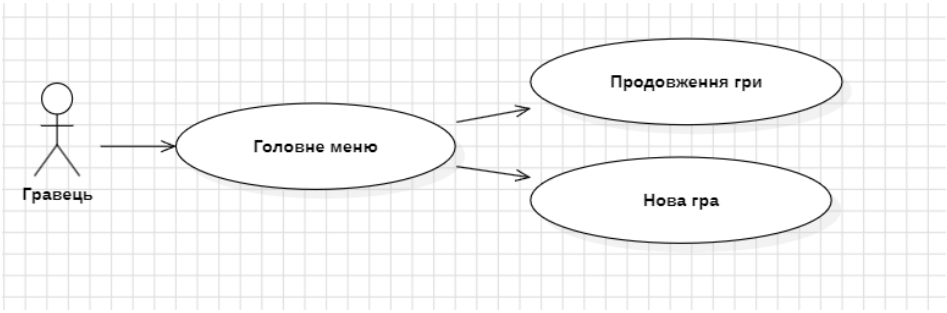


Рисунок 2.6 – Початок гри

Сценарій 2. Заробіток перших монет (рис. 2.7).

Гравець клікає на основний елемент ігрового інтерфейсу, що призводить до заробітку віртуальних монет. Гравець використовує ці монети для покращення своїх можливостей.

Рисунок 2.7 – Заробіток перших монет

2.6 Діаграми діяльності

Діаграми діяльності відтворюють послідовність дій, які відбуваються під час реалізації конкретного сценарію використання або загального функціонування системи. Вони є аналогами блок-схем для будь-якого алгоритму. Як і діаграми станів та переходів, вони представлені у вигляді орієнтованого графа, де вершини представляють дії, а ребра – переходи між діями.

Основні елементи діаграми діяльності включають:

1) дія (action): Відображає окремий крок або дію, яка виконується в процесі. Це може бути специфічна операція, функція, виконання певного блоку коду або будь-яка інша дія, що відбувається в системі;

2) рішення (decision): Показує розгалуження в процесі, залежно від певної умови або вибору. Він має два або більше шляхи виконання, які залежать від заданої умови;

3) старт та кінець (start/end): Вказують на початок та завершення діаграми діяльності або певної послідовності дій. Вони показують стартову та фінальну точки процесу або діяльності;

4) паралельність (concurrent): Вказує на одночасне виконання двох або більше дій або процесів, які не залежать один від одного;

5) флоу (flow): Вказує на послідовність дій або переходи між ними. Використовує стрілки для показу напрямку виконання дій.

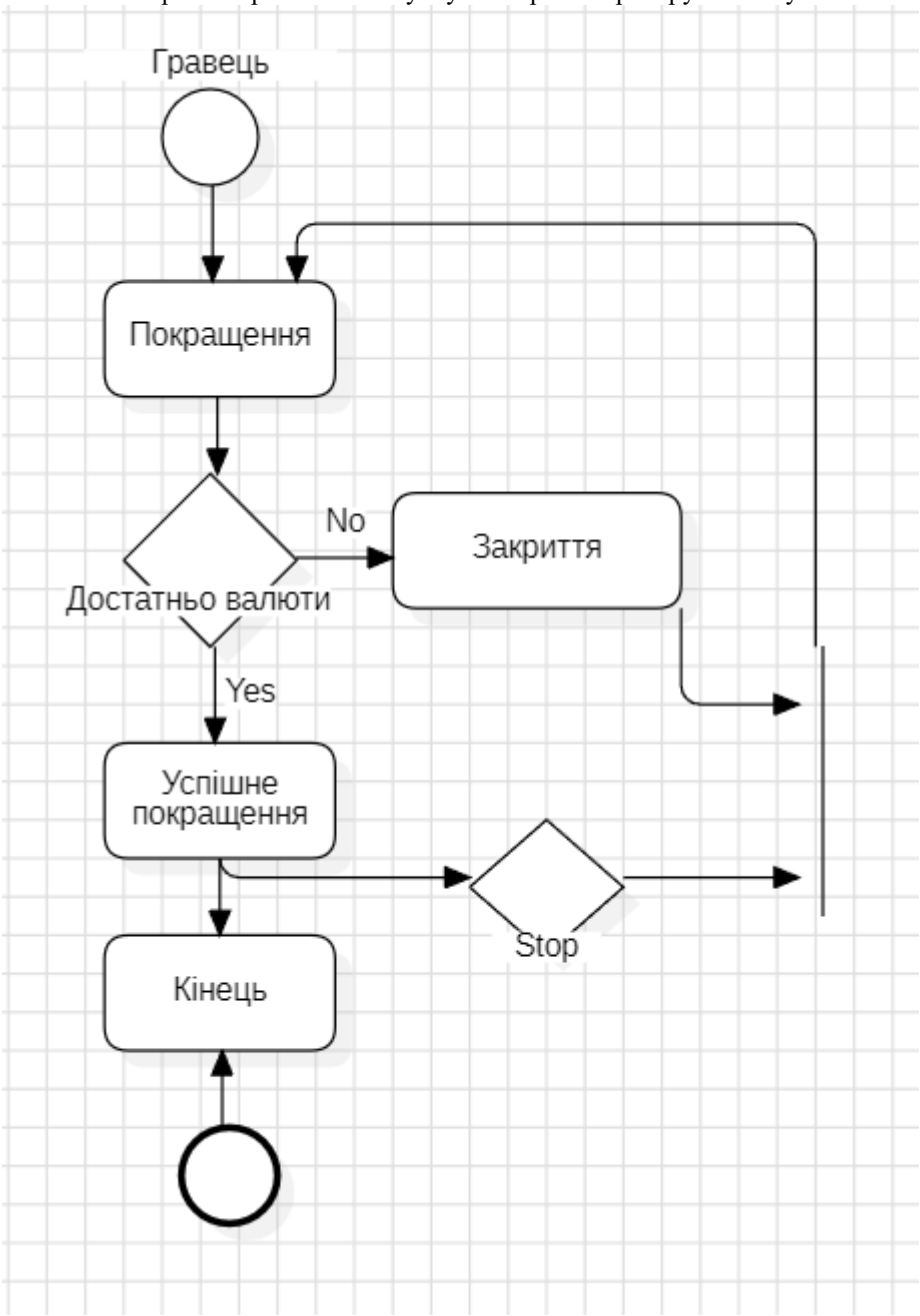


Рисунок 2.8 – Діаграма діяльності покращення

Діаграми діяльності також використовуються для моделювання послідовності дій у бізнес-процесах (рис. 2.9). В цьому контексті вони можуть включати додатковий елемент, відомий як «swimlane».

«Swimlane», що в дослівному перекладі означає «доріжка в плавальному басейні» (через візуальну аналогію), використовується для представлення різних учасників процесу на діаграмі. Swimlanes можуть представляти

окремих осіб, групи людей, відділи компанії або навіть окремі організації. Кожен swimlane відображає дії, які виконуються конкретним учасником.

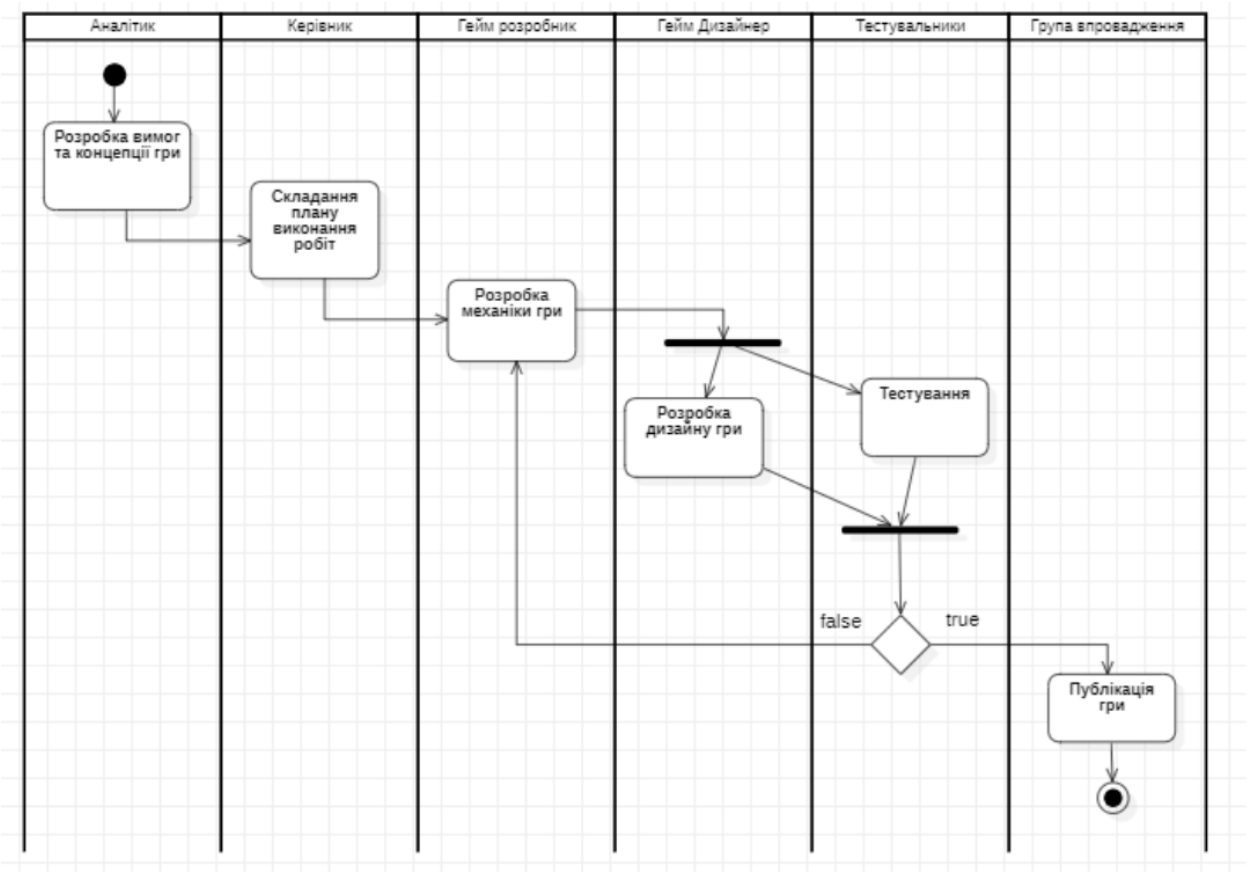


Рисунок 2.9 – Діаграма діяльності для відображення бізнес-процесів

Створено дві діаграми діяльності:

1) покращення. Діаграма перевіряє на наявність ігрової валюти для оновлення, також припинення всього процесу оновлення;

2) діаграма відображення послідовності дій при моделюванні бізнеспроцесів.

Висновки до розділу 2

У розділі 2 проведено проєктування гри засобами мови UML. Створено діаграму варіантів використання. Розроблено алгоритм роботи мобільної гри в жанрі клікер на рушії Unity.

Діаграма класів надає загальний огляд структури програмного забезпечення, відображаючи класи, їх атрибути та методи, а також зв’язки між ними. Це спрощує розуміння архітектури програми та взаємозв’язків між різними компонентами.

Алгоритм роботи програмного забезпечення надає послідовний опис дій, необхідних для досягнення функціональності програми. Цей алгоритм включає ініціалізацію гри, обробку введення гравця, оновлення гри та перевірку умов завершення гри.

Діаграми станів та переходів надають візуальне уявлення про різні стани, у яких може перебувати система або об’єкт, а також переходи між цими станами.

Сценарії використання описують, як користувачі взаємодіють з продуктом або системою для досягнення конкретних цілей.

Діаграми діяльності відтворюють послідовність дій, які відбуваються під час реалізації конкретного сценарію використання або загального функціонування системи.

3 ІНСТРУМЕНТАЛЬНІ ЗАСОБИ РОЗРОБКИ КОМП'ЮТЕРНОЇ ГРИ

3.1 Платформа .NET та мова програмування C#

Платформа .NET та мова програмування C# є потужними інструментами для створення різноманітних програм, включаючи 2D ігри. Їх популярність серед програмістів пояснюється не лише широким розповсюдженням, а й численними перевагами, які вони пропонують[18].

CLR (Common Language Runtime) є ключовою складовою платформи .NET, яка відповідає за виконання коду, управління пам'яттю та безпеку. Щодо C#, вона відзначається простотою використання та статичною типізацією, що полегшує виявлення помилок на етапі компіляції. Ця мова також підтримує об'єктно-орієнтований підхід до програмування.

Широка підтримка та масштабованість .NET дозволяють розробляти як невеликі, так і великі проєкти, включаючи 2D ігри різної складності. Крос-платформеність цієї платформи відкриває можливості для запуску програм на різних операційних системах без необхідності переписування коду.

Для розробки ігор на .NET доступні різноманітні інструменти та бібліотеки, які спрощують процес. Наприклад, MonoGame або Unity надають широкі можливості для роботи з графікою та анімацією. Підтримка різних архітектурних підходів, таких як MVC або MVVM, сприяє чіткому розділенню логіки гри.

Навіть для програмістів з досвідом у інших мовах програмування перехід до C# не складатиме великих труднощів, оскільки вона має схожий синтаксис. Інтегровані середовища розробки, такі як Visual Studio чи Rider, надають зручні інструменти для створення, тестування та відлагодження коду.

Мова програмування C# є однією з лідерів у сфері розробки програмного забезпечення, включаючи ігри. Її популярність пояснюється численними перевагами, які роблять її привабливим інструментом для створення 2D гри.

Однією з ключових переваг C# є його простота та легкість вивчення. Мова була спроектована так, щоб зробити програмування більш зрозумілим та доступним для розробників, особливо для початківців.

Додатковою перевагою C# є його підтримка об’єктно-орієнтованого програмування (ООП). Це дозволяє створювати структуровані та модульні програми, що полегшує управління кодом та забезпечує зручність у розробці та розширенні гри.

Мова C# також відзначається потужною системою типів та автоматичним управлінням пам’яттю, що спрощує процес розробки та допомагає уникнути багатьох типових помилок.

Крім того, інтегровані розробничі середовища, такі як Visual Studio, надають розширений набір інструментів для аналізу коду, налагодження та автоматизованого тестування, що сприяє полегшенню процесу розробки.

Загалом, мова програмування C# є потужним і гнучким інструментом для розробки 2D ігор. Вона поєднує в собі простоту синтаксису, підтримку ООП, широкий вибір фреймворків та бібліотек, що робить її відмінним вибором для розробників, які прагнуть забезпечити ефективну та якісну розробку своїх ігор (рис. 3.1).

Рисунок 3.1 – Логотип Microsoft .NET

3.2 Середовище розробки Visual Studio

Середовище розробки Visual Studio є одним з найпопулярніших та потужних інструментів для створення програмного забезпечення, зокрема і для розробки 2D ігор. Воно пропонує широкий набір функціональних можливостей та інструментів, які роблять його перевагою перед іншими аналогами[14].

По-перше, Visual Studio забезпечує потужну підтримку мови програмування C#, яка є однією з найпоширеніших мов для розробки ігор. Інтегрована підтримка C# у Visual Studio робить створення, редагування, налагодження та відлагодження коду гри легким та зручним. У цьому середовищі можна користуватися всіма функціональними можливостями мови C#, включаючи об’єктно-орієнтоване програмування, делегати, події та інші.

Visual Studio також надає потужні інструменти для управління проєктом та збірки. Воно дозволяє створювати різні типи проєктів, такі як бібліотеки класів, модулі, компоненти тощо, що сприяє організації розробки 2D ігор. Visual Studio автоматично відстежує залежності між файлами, виконує збірку та забезпечує простий спосіб керування вашим проєктом.

Один з великих плюсів Visual Studio - це його потужна система налагодження. Тут можна крок за кроком відстежувати виконання написаної програми, перевіряти значення змінних, використовувати точки зупинки тощо. Це дозволяє ефективно відлагоджувати гру, виявляти та виправляти помилки та забезпечувати якісний розвиток проєкту.

Крім того, Visual Studio має інтеграцію з різними фреймворками та бібліотеками, які широко використовуються для розробки ігор. Наприклад, тут можна легко інтегрувати Unity або інші фреймворки у проєкт Visual Studio. Це дозволяє використовувати потужні функції цих фреймворків, такі як системи фізики, обробка введення, робота з графікою та звуком, для створення 2D ігор.

Не можна не відзначити активну спільноту розробників, яка підтримує Visual Studio. Існує безліч ресурсів, форумів, підручників, блогів та інших джерел, де можна отримати допомогу, відповіді на запитання та поради від досвідчених розробників.

Загалом, використання Visual Studio для розробки 2D ігор має багато переваг. Від широкого набору функціональних можливостей та інструментів до потужної системи налагодження та інтеграції з різними фреймворками - це середовище розробки надає всі необхідні інструменти для створення високоякісних 2D ігор. Не дивно, що Visual Studio залишається популярним вибором серед розробників у галузі геймдеву (рис. 3.2).

Рисунок 3.2 – Логотип Visual Studio

3.3 Рушій для розробки відеоігор Unity

Unity – це інтегроване середовище розробки (IDE) та рушій для створення ігор (рис. 3.3). Рушій Unity, в основному відомий як Unity Engine, є основою для створення різноманітних ігрових проектів, починаючи від ігор для мобільних пристроїв до великих консольних та комп’ютерних ігор [13].

Основні характеристики рушія Unity включають:

1) крос-платформеність: Unity Engine дозволяє розробляти ігри для різних платформ, таких як Windows, macOS, Linux, iOS, Android, консолі Xbox, PlayStation і багато інших;

2) графічний двигун: Unity має потужний графічний двигун, який підтримує високоякісні 2D та 3D графічні об’єкти, ефекти світла та тіней, текстури, анімацію та багато іншого;

3) фізичний двигун: Unity має вбудований фізичний двигун, який дозволяє моделювати реалістичну поведінку об’єктів у грі, таку як гравітація, колізії, рух та взаємодія;

4) аудіо система: Unity має вбудовану систему аудіо, яка дозволяє додавати та керувати звуковими ефектами, музикою та голосовими доріжками у грі;

5) керування сценами та асетами: Unity надає зручні інструменти для управління сценами, об’єктами та ресурсами проекту, що дозволяє легко організовувати та редагувати гру;

6) мови програмування: Рушій Unity підтримує кілька мов програмування, включаючи C#, JavaScript (UnityScript) та Boo. Проте, C# є основною мовою програмування для розробки ігор в Unity;

7) дружній інтерфейс користувача: Unity надає інтуїтивно зрозумілий інтерфейс користувача, що дозволяє легко розуміти та працювати з усіма аспектами розробки гри.

Рисунок 3.3 – Логотип Unity

Unity – це не просто інструмент. Це можливість швидко розпочати розробку, ефективно працювати з усіма аспектами гри і створити щось дійсно захоплююче для гравців.

3.4 Обґрунтування вибору платформи

Сьогодні майже у кожного є смартфон, і багато хто залежить від нього. Хтось грає в ігри, хтось використовує соціальні мережі для заробітку, а для когось телефон – це просто засіб спілкування. Тому створення застосунків для смартфонів є дуже вигідним бізнесом. Однак виникає питання, яку платформу для розробки краще вибрати – Android чи iOS? Ці дві платформи вже давно борються за лідерство, намагаючись випередити одна одну. Неможливо однозначно сказати, яка з них краща, адже обидві мають свої плюси і мінуси.

Користувачів Android більше, оскільки пристрої на цій платформі зазвичай дешевші. Водночас смартфони iOS мають високі стандарти якості, і Apple дуже ретельно перевіряє всі застосунки перед тим, як випустити їх у свій App Store. З одного боку, частіше можна зустріти зіпсований продукт на Android, ніж на iOS, можливо, тому продукти Apple вважаються статусними.

Завдяки ретельній перевірці застосунків, конкуренція між розробниками в App Store не така висока, як на Google Play. Також важливим фактором є ціна розробницького облікового запису: на Google Play це одноразові $25, тоді як на App Store потрібно платити $100 щорічно. Це своєрідна гарантія того, що розробник добросовісний як щодо свого продукту, так і до користувачів. Тому, якщо вибрати будь-яку тему для застосунка, на Google Play буде 100 варіантів, а в App Store – до 20.

Ігрові компанії використовують кілька тактик: випускають ігри одразу на дві платформи, спочатку на iOS, а потім на Android, або лише на одну платформу. У першому випадку потрібні достатні фінансові ресурси для підтримки обох платформ і бажання одразу отримати прибуток від двох гігантів мобільного ринку.

Другий метод використовується з міркувань безпеки ігрового продукту. Наприклад, якщо гра має внутрішні покупки, де можна придбати щось за реальні гроші, то на iOS її вигідніше випустити першою, оскільки відсоток зламаних ігор на цій платформі значно менший, ніж на Android. Це дозволяє розробникам отримати більший прибуток, перш ніж випускати гру на Android. На жаль, будь-яку гру на Android можна зламати, і моди можуть дозволити гравцям купувати предмети безкоштовно, що може завдати значної шкоди успіху компанії.

Останній метод використовується для ексклюзивності або через відсутність фінансів.

Для поточного завдання все ж краще використовувати операційну систему Android.

3.5 Ознайомлення із базовим функціоналом Unity 3D

Інтерфейс Unity дуже простий і зручний у використанні. Якщо ви знаєте основні частини та кнопки, ви зможете працювати з Unity 2D без зовнішньої допомоги. Давайте розглянемо основні компоненти інтерфейсу користувача на прикладі проекту в розробці.

1) scene: Вікно сцени – це місце, де ви можете переглядати та редагувати об’єкти вашого проекту у 2D-просторі. Тут можна переміщувати, обертати і масштабувати об’єкти, а також налаштовувати їхнє розташування;

2) game: Вікно гри показує, як виглядатиме ваша гра під час виконання. Це дозволяє вам переглядати та тестувати ваш проект у режимі реального часу;

3) hierarchy: Панель ієрархії відображає всі об’єкти на вашій сцені в вигляді дерева. Тут можна створювати, видаляти та організовувати об’єкти, встановлюючи батьківські зв’язки між ними;

4) inspector: Панель інспектора дозволяє вам переглядати і змінювати властивості вибраного об’єкта. Ви можете налаштовувати компоненти об’єкта, додавати нові компоненти або видаляти існуючі;

5) project: Панель проекту показує всі файли та ресурси, що використовуються в вашому проекті. Тут можна створювати нові папки, імпортувати файли та організовувати ресурси;

6) console: Консоль відображає повідомлення про помилки, попередження та інші повідомлення, які виникають під час роботи в Unity. Це корисний інструмент для відстеження проблем у вашому коді або проекті;

7) toolbar: Панель інструментів містить кнопки для управління сценою, такими як переміщення, обертання, масштабування об’єктів, запуск і зупинка гри, збереження проекту та інші важливі функції;

8) asset store: Вікно магазину ресурсів дозволяє вам переглядати та завантажувати додаткові ресурси, такі як моделі, текстури, звуки та скрипти, які можна використовувати у вашому проекті.

Розуміння цих основних компонентів інтерфейсу користувача допоможе вам ефективно працювати з Unity 2D і створювати власні проекти без потреби в зовнішній допомозі (рис. 3.4).

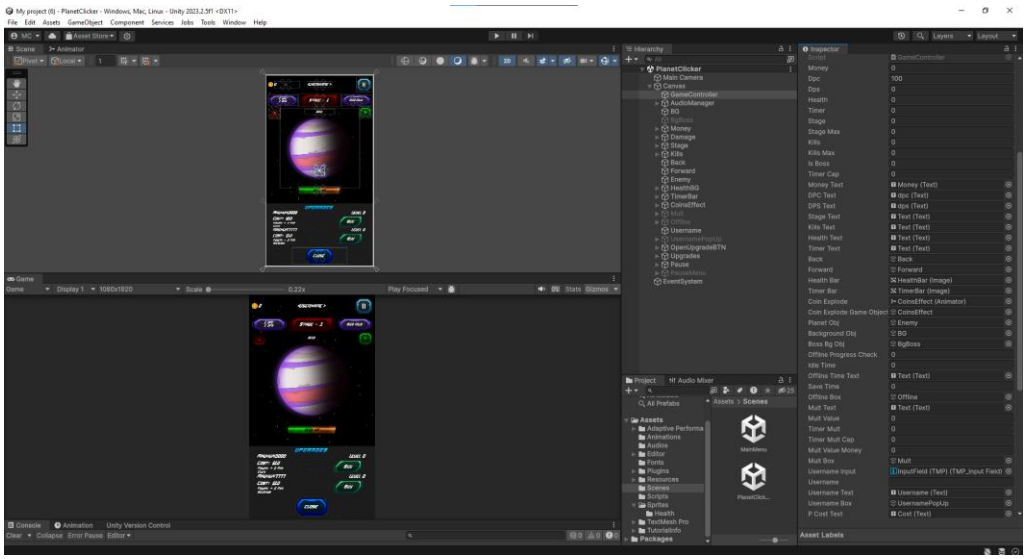


Рисунок 3.4 – Інтерфейс Unity 2D

3.6 Ознайомлення із базовим функціоналом Unity 3D

Створення нової сцени в Unity відбувається через вибір опції «New Scene» у меню «File». Сцена слугує фундаментом для розробки ігор, де можна

розміщувати різноманітні елементи: персонажі, противників, декорації, освітлення, звукові ефекти та інше, що необхідно для створення ігрового світу (рис. 3.5).

Для додавання об’єктів на сцену, розробник може використовувати ручне створення, імпортування з інших джерел або вибір з «Asset Store». Щоб створити новий об’єкт, потрібно перейти до меню «GameObject» та обрати потрібний тип.

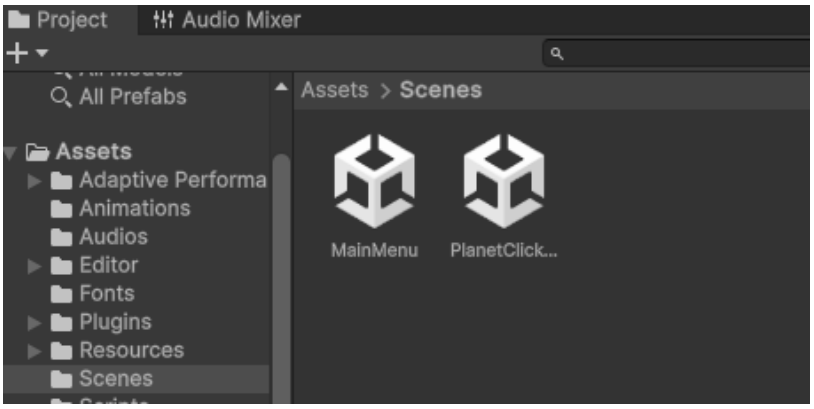


Рисунок 3.5 – Сцени Unity

В Unity кожен об’єкт має набір властивостей, які можна налаштувати через інспектор. Ці властивості охоплюють такі параметри, як положення, розміри, колір, текстури, матеріали, фізичні характеристики тощо.

Ефективна розробка гри передбачає не тільки розміщення об’єктів, але й управління їх поведінкою. Це вимагає написання скриптів, які регулюють взаємодію об’єктів між собою та з ігровим середовищем.

Програмування є ключовим елементом розробки в Unity, де використовується мова C# для написання скриптів, що керують поведінкою об’єктів. C# відома своєю об’єктно-орієнтованістю, гнучкістю та потужністю, що робить її ідеальною для розробки ігор (рис. 3.6).

Програмування в Unity починається зі створення нового C# скрипту через меню «Assets» > «Create» > «C# Script». Після створення скрипт можна прикріпити до об’єкта на сцені, перетягнувши його в ієрархію або через інспектор.

У скрипті визначаються методи, які виконують певні дії. Наприклад, метод «Start» використовується для ініціалізації на початку гри, а «Update» – для коду, що виконується щокадру.

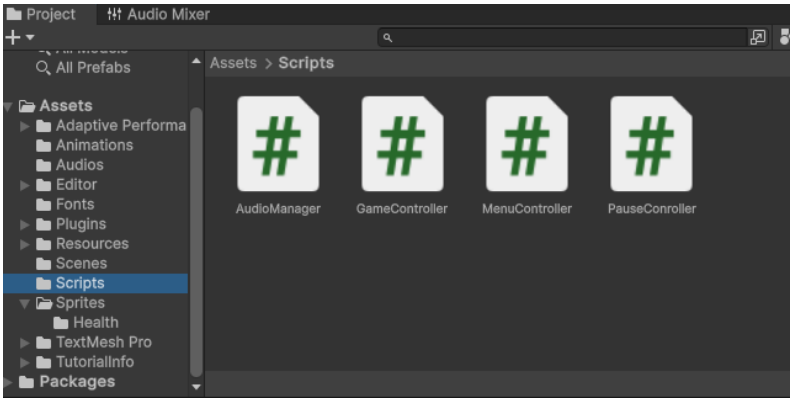


Рисунок 3.6 – C# скрипти

3.7 Тестування та методика тестування ігор

Тестування ігор є критично важливим етапом розробки програмного забезпечення, який забезпечує якість кінцевого продукту та задоволення користувачів. Цей процес охоплює широкий спектр методик і підходів, що дозволяють виявити й виправити помилки, оптимізувати продуктивність та підвищити загальну стабільність гри. У цій статті буде розглянуто основні аспекти тестування ігор, його важливість, методи та інструменти, які використовуються в індустрії.

Індустрія відеоігор є одним з найдинамічніших і найприбутковіших секторів сучасної економіки. З кожним роком ігрові проекти стають дедалі складнішими, а очікування гравців — дедалі вищими. У таких умовах тестування ігор виступає не лише як метод виявлення дефектів, але й як засіб забезпечення високої якості продукту та позитивного користувацького досвіду.

Важливість тестування ігор:

– забезпечення якості: тестування є ключовим етапом у забезпеченні якості ігор. Воно дозволяє розробникам виявляти помилки та недоліки на

ранніх стадіях розробки, що значно знижує витрати на їх виправлення на пізніших етапах;

– задоволення користувачів: висока якість ігор безпосередньо впливає на задоволення користувачів. Помилки, баги та інші проблеми можуть негативно вплинути на користувацький досвід, що призводить до негативних відгуків і втрати довіри гравців;

– економічна вигода: Виявлення та виправлення помилок на стадії розробки дозволяє уникнути значних витрат на виправлення після випуску гри. Це особливо важливо для великих проектів з великим бюджетом.

Методи тестування ігор:

– ручне тестування: ручне тестування включає в себе ретельну перевірку гри вручну, щоб виявити дефекти, баги та інші проблеми. Це найпоширеніший метод, який дозволяє тестувальникам перевіряти функціональність, графіку, звук та інші аспекти гри;

– автоматизоване тестування: втоматизоване тестування передбачає використання спеціальних інструментів і скриптів для автоматизації процесу перевірки гри. Це дозволяє значно скоротити час на тестування та підвищити його ефективність;

– тестування продуктивності: цей метод включає перевірку гри на продуктивність, стабільність та споживання ресурсів. Тестування продуктивності дозволяє виявити проблеми з оптимізацією та забезпечити плавний геймплей на різних пристроях;

– бета-тестування: бета-тестування передбачає випуск попередньої версії гри для обмеженої групи гравців. Це дозволяє отримати зворотній зв'язок від реальних користувачів та виявити проблеми, які могли залишитися непоміченими під час внутрішнього тестування.

Висновки до розділу 3

У третьому розділі було досліджено рушій Unity як потужний інструмент для створення ігор, зокрема для реалізації як 2D, так і 3D графіки. Він надає широкий спектр можливостей для реалізації різноманітних графічних об’єктів, фізики, анімації та інших функцій, які допомагають розробникам створювати захоплюючі ігрові світи. Значний внесок у розвиток Unity має його активна спільнота розробників, а також доступні розширення та підтримка, що сприяє полегшенню процесу розробки і досягненню бажаних результатів.

Середовище розробки Visual Studio також заслуговує уваги як потужний інтегрований інструмент, що надає розробникам широкі можливості для написання, налагодження та керування кодом. Воно забезпечує зручну роботу з мовою програмування C# та іншими мовами, автодоповнення, відступи, контекстну довідку та інші корисні функції. Його інтеграція з Unity дозволяє зручно розробляти ігри безпосередньо в середовищі розробки.

Усі три компоненти – мова програмування C#, рушій Unity та середовище розробки Visual Studio – взаємодіють між собою, утворюючи потужну комбінацію для розробки ігор.

Тестування ігор є невід’ємною частиною розробки програмного забезпечення, яка забезпечує високу якість кінцевого продукту та задоволення користувачів. Використання різноманітних методик та інструментів дозволяє ефективно виявляти та виправляти помилки, оптимізувати продуктивність та забезпечити стабільність гри. У сучасних умовах швидкого розвитку ігрової індустрії тестування стає ключовим фактором успіху будь-якого ігрового проекту.

4 ПРОГРАМНА РЕАЛІЗАЦІЯ ІГРОВОГО ЗАСТОСУНКУ

В ігровому застосунку присутні такі ключові елементи інтерфейсу, як головне меню, меню налаштувань, меню паузи, меню покращення. Для розробки цього інтерфейсу були використані стандартні інструменти Unity, включаючи компоненти UI Canvas, які містять різні елементи, такі як кнопки, текстові поля, панелі тощо.

UI-дизайн в ігровому застосунку відіграє критичну роль, оскільки він формує спосіб, яким користувачі взаємодіють з грою. Добре продуманий дизайн може значно підвищити задоволення від гри, надаючи користувачам комфортну та просту взаємодію з геймплеєм.

Дизайн UI-інтерфейсу в ігровому застосунку виконує важливу роль, не лише забезпечуючи взаємодію користувача з грою, але й впливаючи на загальне враження від геймплею. Ефективно розроблений UI-дизайн може створити гармонійне та привабливе середовище, в якому гравці зможуть легко орієнтуватися та отримувати задоволення від гри.

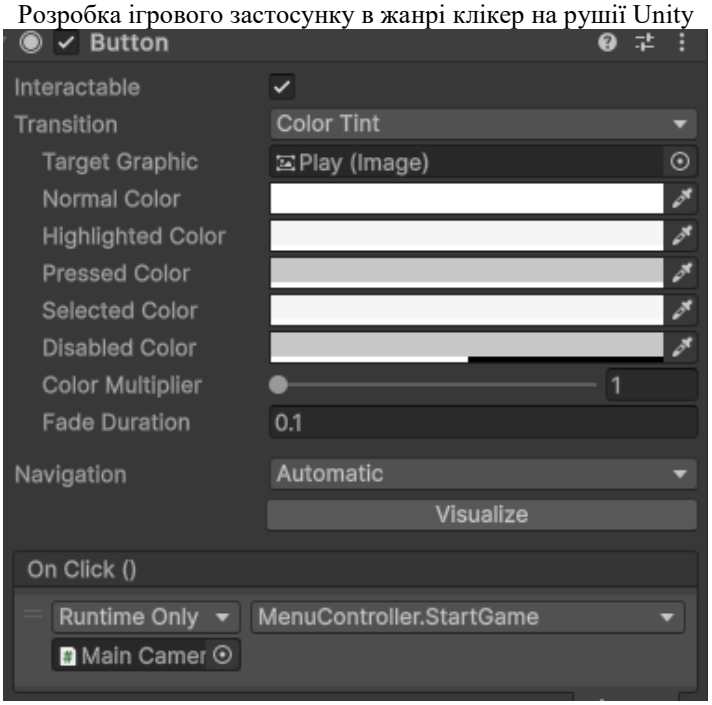
4.1 Реалізація головного меню

На рисунку 4.1, показано головне меню, яке включає три основні кнопки: Play, New game, Exit; та додаткові: Settings, How to play.

По натисканню на кнопку Play, гравець перейде до гри, продовжуючи пригоду з точки, де він раніше зупинився. Якщо гравець натискає New game, тоді гра починається з самого початку. При натисканні кнопки Exit, гравець виходить з гри.

Рисунку 4.1 – Головне меню

Також було розроблено скрипт MenuController (рис. 4.3), який керує головним меню гри, тут реалізовані методи продовження гри (рис.4.2), нової гри, налаштувань та виходу з гри.



Рисунку 4.2 – Кнопка Play

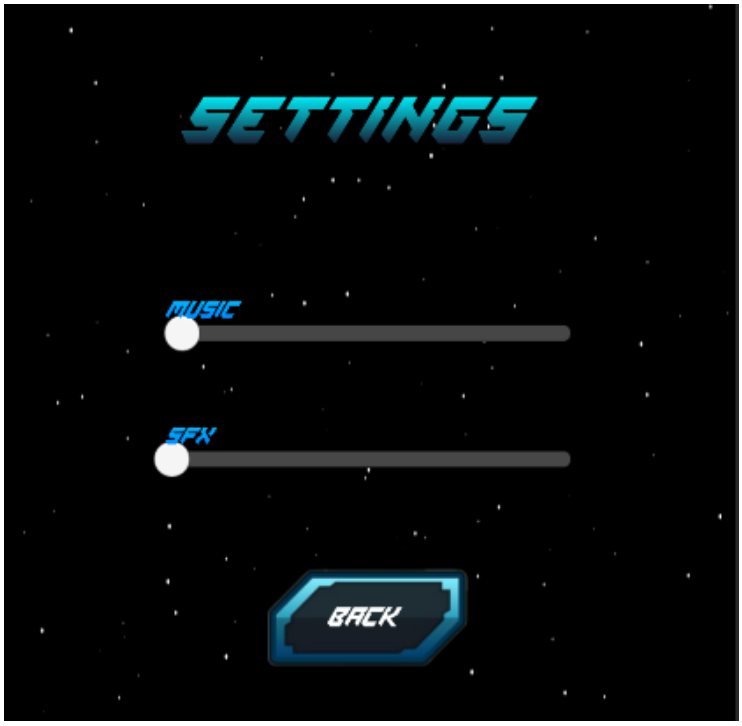
Рисунку 4.3 – Скрипт MenuController

Коли гравець натискає кнопку Settings, відкривається панель налаштувань. Ця панель містить параметри, які гравець може налаштувати за своїм бажанням (рис. 4.4).

Гравець може змінити налаштування двох основних типів звуків: музики та ефектів.

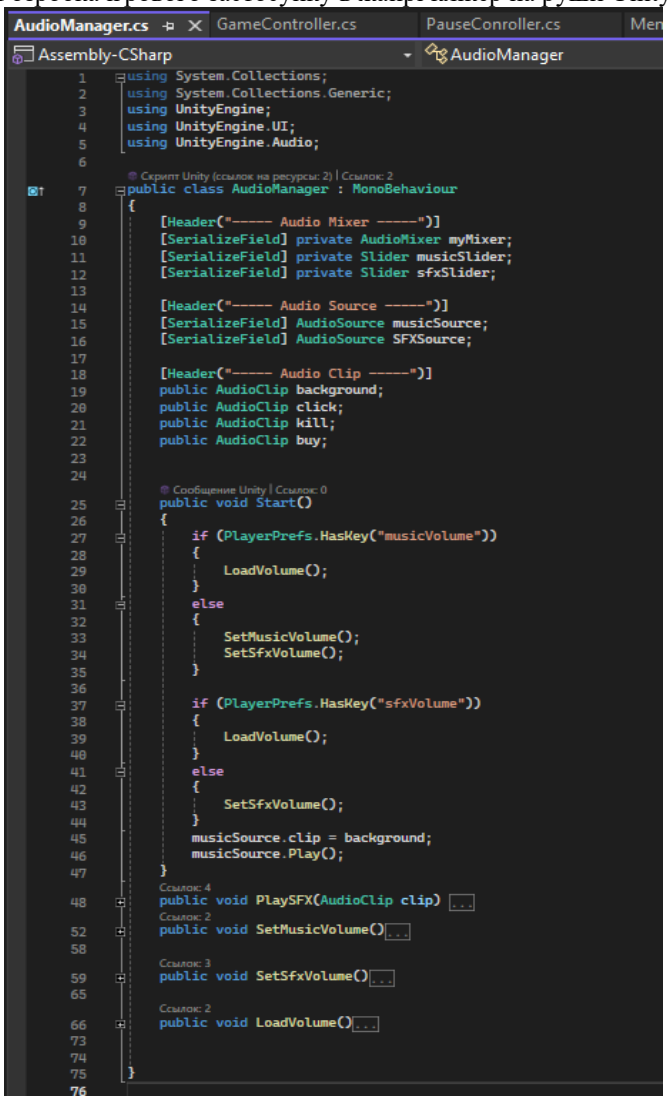
– Музика: гравець може регулювати гучність музики в грі;

– ефекти: гравець також може регулювати гучність звукових ефектів. Звукові ефекти – це звуки, які відтворюються під час виконання певних дій у грі.



Рисунку 4.4 – Панель налаштувань

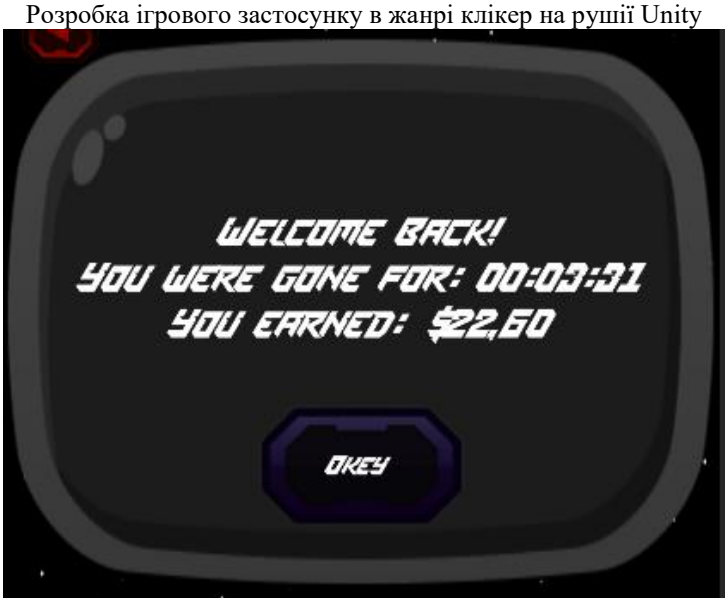
Скрипт AudioController (рис. 4.5), керує музикою та звуковими ефектами в грі, а також відповідає за налаштування звуку.



Рисунку 4.5 – Скрипт AudioController

4.2 Реалізація привітальних панелей

Коли гравець повертається до гри, він відразу потрапляє на головну сцену, де його вітає панель офлайн заробітку (рис. 4.6). Ця панель відображає кількість ресурсів, які гравець заробив під час своєї відсутності. Це дає гравцю відчуття постійного прогресу, навіть коли він не активно грає, та стимулює повернутися до гри. Панель офлайн заробітку – це важливий елемент інтерфейсу, який допомагає утримувати зацікавленість гравця (рис. 4.7).



Рисунку 4.6 – Панель офлайн заробітку

```
public void LoadOfflineProduction()
{
    if (OfflineProgressCheck == 1)
    {
        offlineBox.gameObject.SetActive(true);
        long previousTime = Convert.ToInt64(PlayerPrefs.GetString("OflineTime"));
        oldTime = DateTime.FromBinary(previousTime);
        currentDate = DateTime.Now;
        TimeSpan difference = currentDate.Subtract(oldTime);
        idleTime = (float)difference.TotalSeconds;

        var moneyToEarn = Math.Ceiling(healthCap / 14) / healthCap * (dps / 5) * idleTime;
        money += moneyToEarn;
        TimeSpan timer = TimeSpan.FromSeconds(idleTime);

        offlineTimeText.text = "Welcome Back!" + "\nYou were gone for: " + timer.ToString(@"hh\:mm\:ss") + "\nYou earned: $" + WordNotation(moneyToEarn, "F2");

    }
}
```

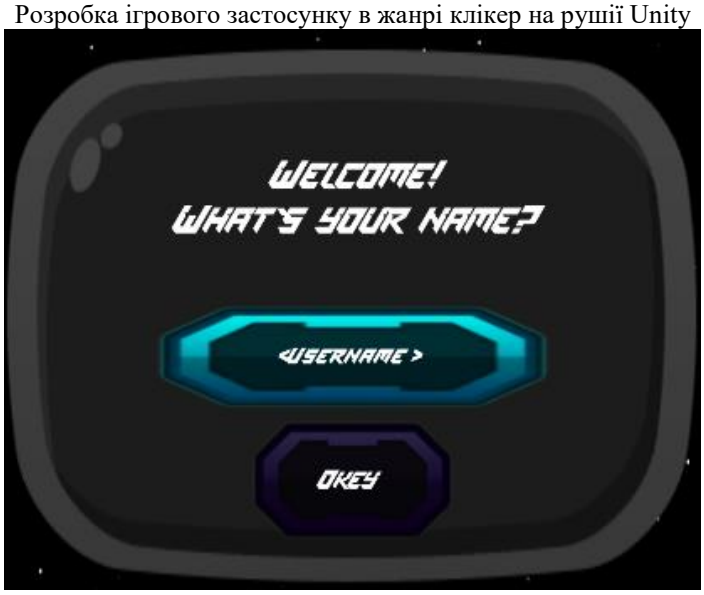
Рисунку 4.7 – Метод офлайн заробітку

Коли гравець починає нову гру, відбувається перехід в основну сцену гри і гравець зустрічається з панеллю введення імені (рис. 4.8). Це важливий момент, оскільки це дозволяє гравцю персоналізувати свій досвід гри.

Панель введення імені має наступні елементи:

– Поле введення: Це місце, де гравець вводить своє ім’я. Воно зазвичай має обмеження на кількість символів, щоб ім’я не було занадто довгим.

– Кнопка «Підтвердити»: Після введення імені гравець натискає цю кнопку, щоб підтвердити свій вибір. Після цього ім’я зберігається і використовується у грі (рис.4.9).



Рисунку 4.8 – Панель вводу імені

```
Ссылок: 0
public void UsernameChange()
{
    username = usernameInput.text;
}

Сообщение Unity | Ссылок: 0
public void Start()
{
    offlineBox.gameObject.SetActive(false);
    multBox.gameObject.SetActive(false);
    upgradeBox.gameObject.SetActive(false);


    Load();
    LoadEnemySprites();
    UpdateEnemyImage();
    if (username == "<Username>" )
    {
        usernameBox.gameObject.SetActive(true);
    }
    else
    {
        usernameBox.gameObject.SetActive(false);
    }
    IsBossChecker();
    health = healthCap;
    timerCap = 30;
    multValue = new System.Random().Next(20, 100);
    timerMultCap = new System.Random().Next(10, 20);
    timerMult = timerMultCap;


}
```

Рисунку 4.9 – Методи для вводу імені гравця

4.3 Реалізація ворогів

Основний елемент гри, планета, є центральним об’єктом, який гравець використовує для заробітку ігрової валюти. Планета представлена у вигляді великого, красиво оформленого об’єкта на головному екрані гри (рис. 4.10).

Рисунку 4.10 – Основна сцена гри

Кожен клік по планеті знімає певну кількість очок здоров’я з планети (рис. 4.11).

```
Ссылок: 0
public void Hit()
{
    health -= dpc;
    if (health <= 0)
    {
        Kill();
        ChangeEnemy();
    }
    AnimationClicked();
    audioManager.PlaySFX(audioManager.click);
}
```

Рисунку 4.11 – Метод нанесення шкоди

Коли здоров’я планети досягає нуля, планета знищується (рис. 4.12), і гравець отримує винагороду у вигляді ігрової валюти.

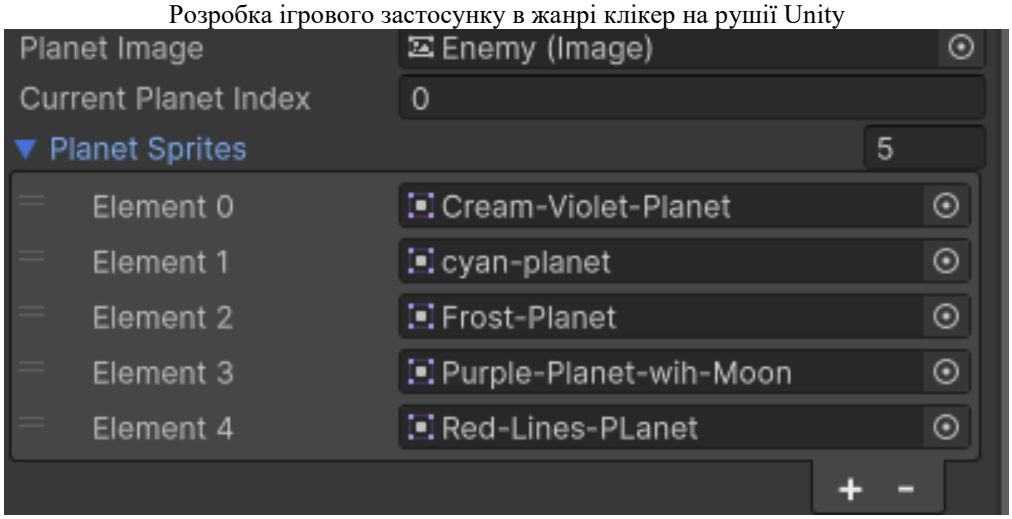
```
Ссылок: 2
public void Kill()
{
    money += Math.Ceiling(healthCap / 14);
    coinExplode.Play("CoinExplode", 0, 0);
    if (stage == stageMax)...
    IsBossChecker();
    health = healthCap;
    if (isBoss > 1) timer = timerCap;
    killsMax = 10;
    audioManager.PlaySFX(audioManager.kill);
}
```

Рисунку 4.12 – Метод знищення та нарахування валюти за знищення

Також після знищення планети з’являється інша планета (рис. 4.13), (рис. 4.14).

```
//enemy changer
Ссылок: 1
public void LoadEnemySprites()
{
    // Download all sprites enemies
    planetSprites = Resources.LoadAll<Sprite>("Planets 2.0 Preview");
    if (planetSprites.Length == 0)
    {
        Debug.LogError("No enemy sprites found in Resources/Planets 2.0 Preview");
    }
}

Ссылок: 2
public void ChangeEnemy()
{
    currentPlanetIndex = (currentPlanetIndex + 1) % planetSprites.Length;
    UpdateEnemyImage();
}

Ссылок: 2
public void UpdateEnemyImage()
{
    if (planetSprites.Length > 0)
    {
        planetImage.sprite = planetSprites[currentPlanetIndex];
    }
}
```

Рисунку 4.13 – Методи зміни планети



Рисунку 4.14 – Налаштування в компонентах

4.4 Реалізація рівнів та босів

У грі впроваджено систему рівнів та босів (рис. 4.15). Щоб перейти на наступний рівень, гравець має знищити десять планет. Кожен п’ятий рівень включає боса, який має значно більше здоров’я і його потрібно знищити за 30 секунд. Гравець має розробити стратегію для його перемоги. Це додає грі більше виклику та стратегічного планування.

```
public void IsBossChecker()
{
    if (stage % 5 == 0)
    {
        isBoss = 10;
        stageText.text = "BOSS Stage - " + stage;
        timer -= Time.deltaTime;
        if (timer <= 0) Back();

        timerText.text = timer.ToString("F2") + "/" + timerCap;
        timerBar.gameObject.SetActive(true);
        timerBar.fillAmount = timer / timerCap;
        killsMax = 1;
        bgBoss.gameObject.SetActive(true);
    }
    else
    {
        isBoss = 1;
        stageText.text = "Stage - " + stage;
        timerText.text = "";
        timerBar.gameObject.SetActive(false);
        timer = 30;
        killsMax = 10;
        bgBoss.gameObject.SetActive(false);
    }
}
```

Рисунку 4.15 – Метод перевірки на боса

Якщо гравець не може перемогти боса, тоді він може повернутись на попередній рівень для того, щоб заробити більше ігрової валюти, придбати покращення, та з більшими силами перемогти боса (рис. 4.16).

```
Ссылок: 1
public void Back()
{
    stage -= 1;
    IsBossChecker();
    health = healthCap;
}

Ссылок: 0
public void Forward()
{
    stage += 1;
    IsBossChecker();
    health = healthCap;
}
```

Рисунку 4.16 – Методи для переміщення між рівнями

4.5 Реалізація магазину покращень

Ігрова валюта може бути використана для купівлі покращень, які допоможуть гравцю наносити більше шкоди за клік чи за секунду. Методи Upgrades (рис. 4.17), BuyUpgrades та UpgradeDefaults реалізують купівлю покращень в грі (рис. 4.18).

```
//Upgrades
Ссылок: 1
public void Upgrades()
{
    cCostText.text = "Cost: $" + WordNotation(cCost, "F2");
    cLevelText.text = "Level: " + WordNotation(cLevel, "");
    cPowerText.text = "+" + WordNotation(cPower, "F2") + " dpc";

    pCostText.text = "Cost: $" + WordNotation(pCost, "F2");
    pLevelText.text = "Level: " + WordNotation(pLevel, "");
    pPowerText.text = "+" + WordNotation(pPower, "F2") + " dps";

}
Ссылок: 0
```

Рисунку 4.17 – Метод Upgrades

```
Ссылок: 0
public void BuyUpgrade(string id)
{
    switch (id)
    {
        case "pl":
            if (money >= pCost)
            {
                UpdateDefaults(ref pLevel, pCost);
                dps += pPower;
                audioManager.PlaySFX(audioManager.buy);
            }
            break;
        case "cl":
            if (money >= cCost)
            {
                UpdateDefaults(ref cLevel, cCost);
                dpc += 1 + cPower;
                audioManager.PlaySFX(audioManager.buy);
            }
            break;
    }
}

Ссылок: 2
public void UpdateDefaults(ref int level, double cost)
{
    money -= cost;
    level++;
}
```

Рисунку 4.18 – Методи BuyUpgrades та UpgradeDefaults

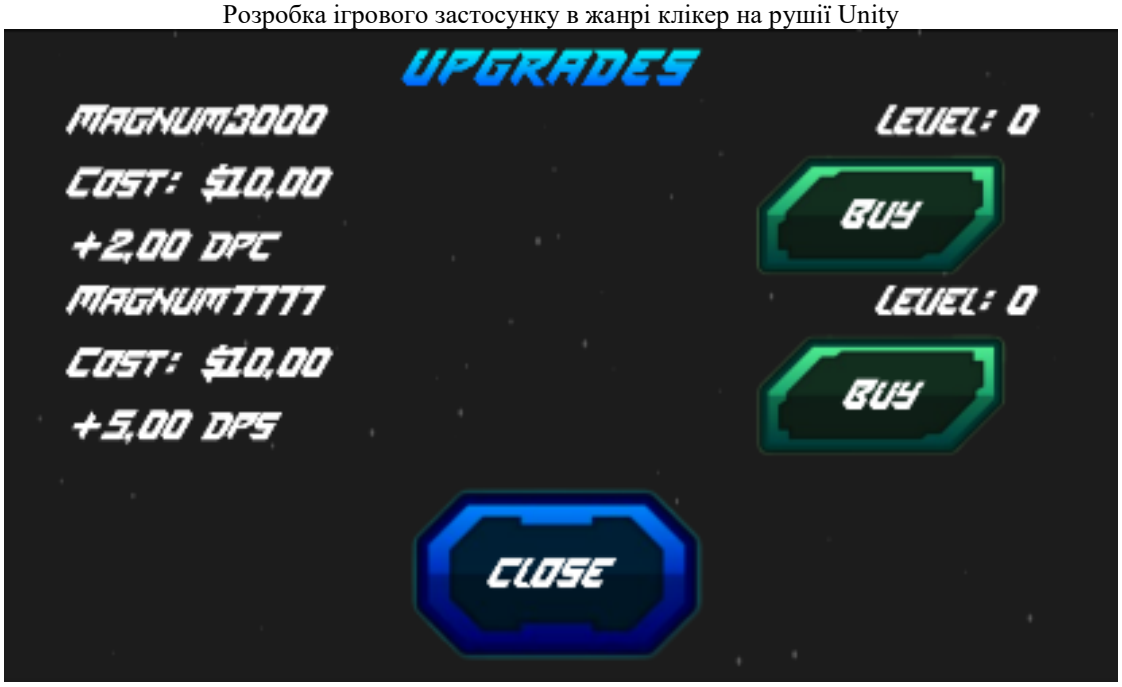
Коли гравець натискає кнопку Upgrades, відкривається спеціальна панель. Ця панель представляє собою інтерактивний інтерфейс, де гравець може покращувати свої навички та здібності (рис. 4.19).

Ось декілька основних елементів цієї панелі:

– шкода за клік: Гравець може збільшувати кількість шкоди, який він завдає при кожному кліку. Це може бути корисним для швидкого знищення ворогів або збору ресурсів;

– пасивна шкода за секунду: Це автоматичний шкода, який гравець завдає кожну секунду, незалежно від того, чи клікає він по екрану. Це дозволяє гравцю накопичувати шкода, навіть коли він не активно грає.

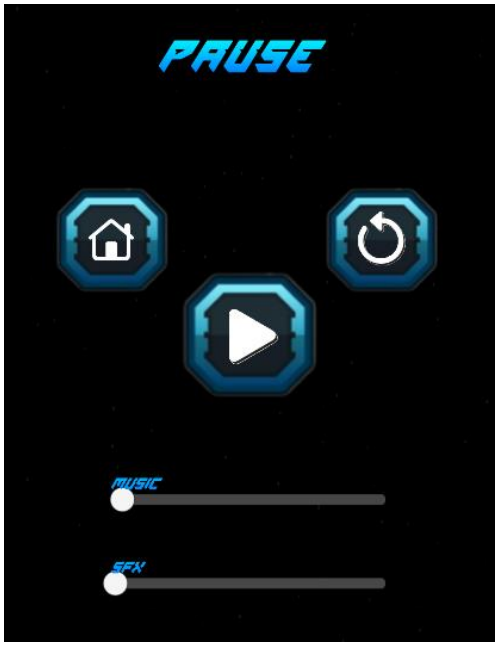
Панель Upgrades є важливою частиною гри, оскільки вона дозволяє гравцям налаштовувати свій стиль гри та стратегію. Вона також додає глибину грі, оскільки гравці повинні вирішувати, в які навички вони хочуть інвестувати свої ресурси. Завдяки добре розробленому інтерфейсу, гравці можуть легко зрозуміти, як вони можуть покращити свої навички та здібності.



Рисунку 4.19 – Панель Upgrades

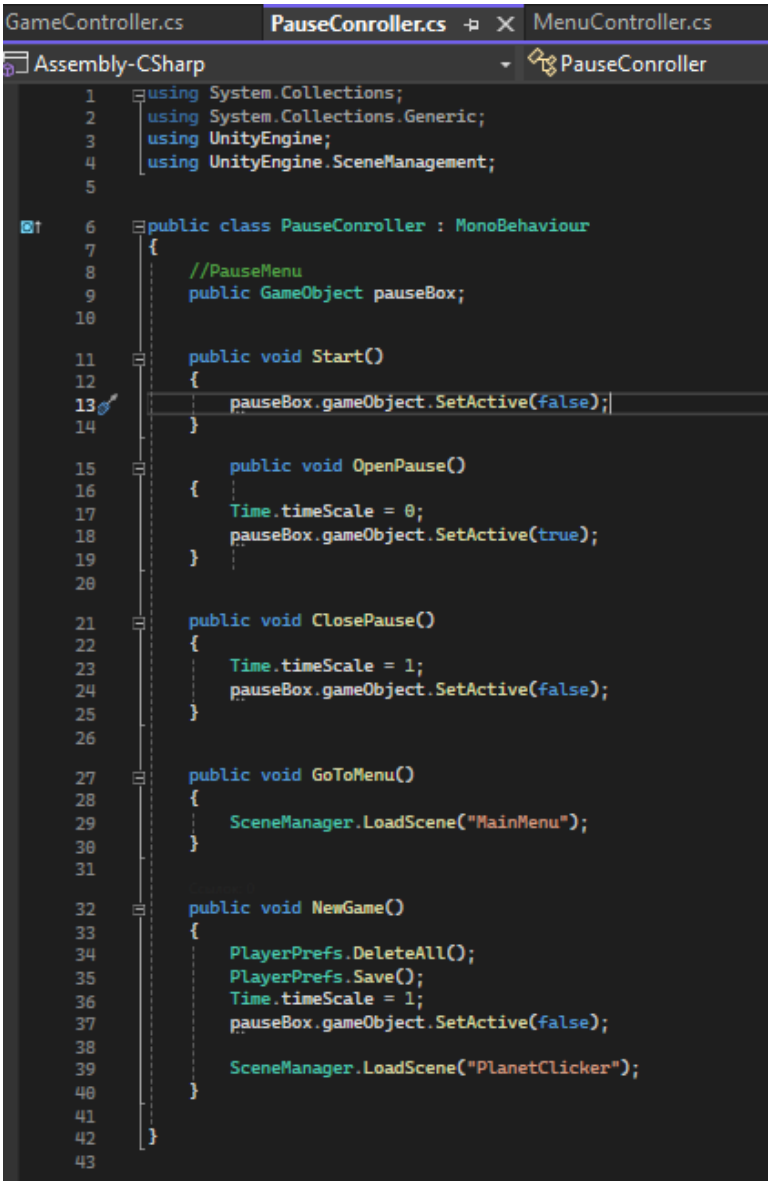
4.6 Реалізація паузи

Кнопка паузи дозволяє гравцю зупинити гру і відпочити (рис. 4.20). Коли гра на паузі, всі дії зупиняються. Звідси, гравець може змінити налаштування, почати нову гру, перейти до головного меню або продовжити гру.



Рисунку 4.20 – Меню паузи

Скрипт PauseController (рис. 4.21), за допомогою якого було реалізовано паузу в грі для того, щоб гравець міг почати нову гру або вийти в головне меню.



Рисунку 4.21 – Скрипт PauseController

4.7 Реалізація івентів та метод WordNotation

До гри було додано івент, який з’являється рандомно в 10-20 секунд (рис. 4.22), коли гравець натискає на елемент івенту, він отримує ігрову валюту, кількість валюти залежить від шкоди за секунду (рис. 4.23).

Рисунку 4.22 – Вигляд івенту в грі

```
Ссылок: 0
public void OpenMult()
{
    multBox.gameObject.SetActive(false);
    money += multValueMoney;
    timerMultCap = new System.Random().Next(10, 20);
    timerMult = timerMultCap;
    multValue = new System.Random().Next(20, 100);
}
```

Рисунку 4.23 – Метод івенту

Метод WordNotation використовується для перетворення великих чисел у більш зручний для сприйняття формат, використовуючи стандартні префікси, такі як K (тисячі), M (мільйони) та інші (рис. 4.23). Це особливо корисно у клікерних іграх, де гравці часто накопичують величезні кількості очок або валюти (рис. 4.24).

Рисунку 4.23 – Вигляд WordNotation на ігровій валюті

Рисунку 4.24 – Вигляд WordNotation на шкалі здоров’я

4.8 Реалізація збереження в грі

Збереження даних у грі – це важливий аспект розробки, який дозволяє гравцям зберігати свій прогрес і повертатися до гри з того місця, де вони зупинилися (рис. 4.25).

PlayerPrefs – це найпростіший спосіб збереження, який дозволяє зберігати прості дані, такі як рівні, очки та налаштування. Методи Save та Load реалізовані для збереження та завантаження даних.

```
public void Save()
{
    OfflineProgressCheck = 1;
    PlayerPrefs.SetString("money", money.ToString());
    PlayerPrefs.SetString("dpc", dpc.ToString());
    PlayerPrefs.SetString("dps", dps.ToString());
    PlayerPrefs.SetString("username", username);
    PlayerPrefs.SetInt("stage", stage);
    PlayerPrefs.SetInt("stageMax", stageMax);
    PlayerPrefs.SetInt("kills", kills);
    PlayerPrefs.SetInt("killsMax", killsMax);
    PlayerPrefs.SetInt("isBoss", isBoss);
    PlayerPrefs.SetInt("cLevel", cLevel);
    PlayerPrefs.SetInt("pLevel", pLevel);
    PlayerPrefs.SetInt("OfflineProgressCheck", OfflineProgressCheck);
    PlayerPrefs.SetInt("currentPlanetIndex", currentPlanetIndex);

    PlayerPrefs.SetString("OflineTime", DateTime.Now.ToBinary().ToString());
}

Ссылок: 1
public void Load()
{
    money = double.Parse(PlayerPrefs.GetString("money", "0"));
    dpc = double.Parse(PlayerPrefs.GetString("dpc", "1"));
    dps = double.Parse(PlayerPrefs.GetString("dps", "0"));
    stage = PlayerPrefs.GetInt("stage", 1);
    stageMax = PlayerPrefs.GetInt("stageMax", 1);
    kills = PlayerPrefs.GetInt("kills", 0);
    killsMax = PlayerPrefs.GetInt("killsMax", 10);
    isBoss = PlayerPrefs.GetInt("isBoss", 1);
    cLevel = PlayerPrefs.GetInt("cLevel", 0);
    pLevel = PlayerPrefs.GetInt("pLevel", 0);
    OfflineProgressCheck = PlayerPrefs.GetInt("OfflineProgressCheck", 0);
    currentPlanetIndex = PlayerPrefs.GetInt("currentPlanetIndex", 0);

    username = PlayerPrefs.GetString("username", "<Username>");
    LoadOfflineProduction();
}
```

Рисунку 4.25 – Методи збереження та завантаження даних

4.9 Реалізація анімацій

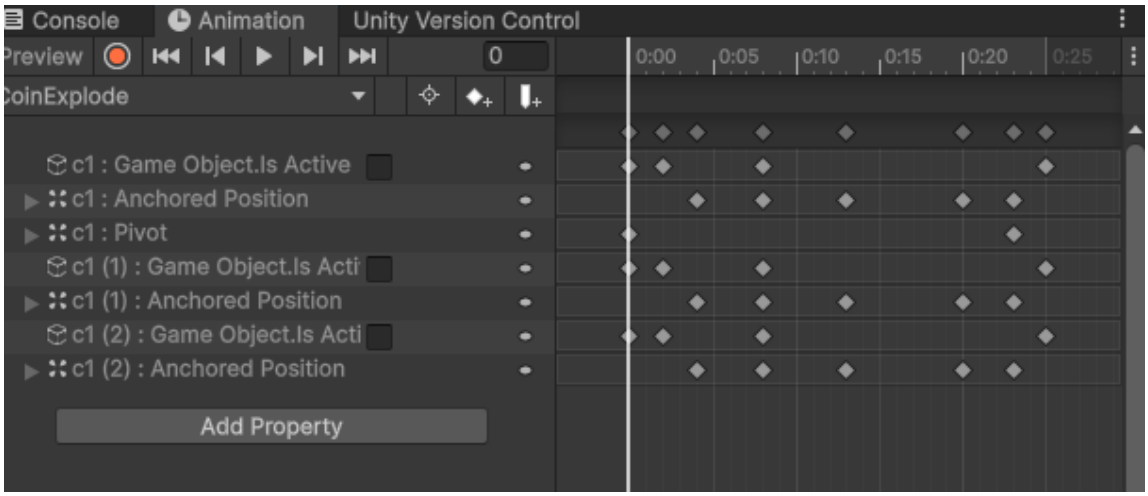
Анімація є важливою частиною розробки гри. Вона додає життя і рух до гри, роблячи її більш привабливою та цікавою для гравців. Перша анімація створена за допомогою DoTween (рис. 4.26).

DOTween (також відомий як HOTween v2) – це швидкий, ефективний, повністю типобезпечний об’єктно-орієнтований двигун анімації для Unity.

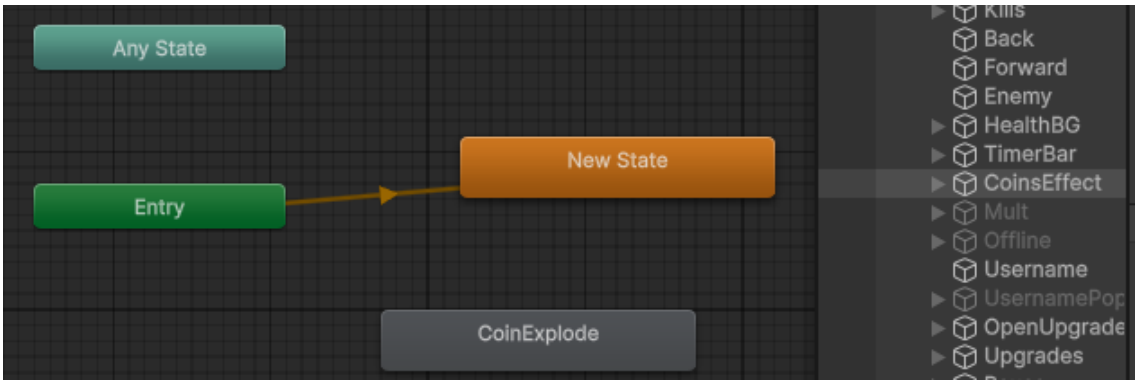
```
Ссылок: 1
public void AnimationClicked()
{
    _planetObj.transform.DOBlendableScaleBy(new Vector3(0.05f, 0.05f, 0.05f), 0.05f).OnComplete(PlanetScaleBack);
    _backgroundObj.transform.DOBlendableScaleBy(new Vector3(0.05f, 0.05f, 0.05f), 0.05f).OnComplete(BackgroundScaleBack);
    _bossBgObj.transform.DOBlendableScaleBy(new Vector3(0.05f, 0.05f, 0.05f), 0.05f).OnComplete(BossBgScaleBack);
}
Ссылок: 1
private void PlanetScaleBack()
{
    _planetObj.transform.DOBlendableScaleBy(new Vector3(-0.05f, -0.05f, -0.05f), 0.05f);
}

Ссылок: 1
private void BackgroundScaleBack()
{
    _backgroundObj.transform.DOBlendableScaleBy(new Vector3(-0.05f, -0.05f, -0.05f), 0.05f);
}

Ссылок: 1
private void BossBgScaleBack()
{
    _bossBgObj.transform.DOBlendableScaleBy(new Vector3(-0.05f, -0.05f, -0.05f), 0.05f);
}
```

Рисунку 4.26 – Анимація при кліканні на планету

Анімація була створена в Unity за допомогою інструментів Animation (рис. 4.27), Animator (рис. 4.28). Ця анімація активується, коли планета знищується.



Рисунку 4.27 – Анимація в Animation



Рисунку 4.28 – Анимація в Animator

Візуально це створює ефект, ніби монети випадають прямо з планети (рис. 4.29).

Рисунку 4.29 – Вигляд анімації в грі

Висновки до розділу 4

В четвертому розділі було детально описано процес розробки та програмної реалізації різноманітних елементів інтерфейсу користувача для гри. Зазначені основні скрипти та їх функціональність, включаючи GameController, PauseController, MenuController, та AudioController, які керують основною логікою гри, паузою, головним меню та аудіо ефектами відповідно.

Також описано важливі елементи гри, такі як планета, система покращень, рівні та боси, івенти, збереження даних та анімація. Кожен з цих елементів має велике значення для геймплею та взаємодії з користувачем.

Програмна реалізація включає в себе ефективне використання різноманітних інструментів та технологій для досягнення мети створення зручного, привабливого та функціонального користувацького інтерфейсу.

ВИСНОВКИ

Під час виконання кваліфікаційної роботи бакалавра було розроблено 2D ігровий застосунок в жанрі клікер. Проведено аналіз актуальності сфери Android-ігор і розглянуто декілька аналогів за обраним жанром. Були проаналізовані їх функціональні можливості, переваги та недоліки. Також був проведений системний аналіз для визначення цілей гри, користувачів та сценаріїв використання.

Крім того, було проведено проєктування гри засобами мови UML, що включає створення діаграми варіантів використання та розробку алгоритму роботи мобільної гри в жанрі клікер на рушії Unity. Діаграма класів надає загальний огляд структури програмного забезпечення, відображаючи класи, їх атрибути та методи, а також зв’язки між ними, що спрощує розуміння архітектури програми та взаємозв’язків між компонентами. Діаграми станів і переходів візуалізують різні стани системи або об’єктів та їхні переходи, а сценарії використання демонструють взаємодію користувачів із системою для досягнення конкретних цілей.

В окремому розділі було розглянуто Unity як потужний інструмент для створення ігор, здатний реалізувати як 2D, так і 3D графіку. Завдяки своїм можливостям у створенні графічних об’єктів, фізики та анімації, Unity дозволяє розробникам створювати захоплюючі ігрові світи. Активна спільнота розробників і доступні розширення значно полегшують процес розробки. Інтеграція Visual Studio з Unity створює ефективну комбінацію для розробки ігор, забезпечуючи зручний і продуктивний процес роботи.

Результатом кваліфікаційної роботи є готовий ігровий застосунок, який відповідає поставленим вимогам і досягає поставлених цілей. Розроблений 2D ігровий застосунок є результатом комплексного підходу до аналізу, дизайну, розробки та тестування, що забезпечило створення функціональної та привабливої гри.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. William Hosch. Encyclopedia Britannica: electronic platform game, 2009. 375 p.

2. Alexander S. They create worlds: the story of the people and companies that shaped the video game industry, vol. I: 1971-1982. Taylor & Francis Group, 2019. 586 p.

3. Donovan T. Replay: the history of video games. East Sussex, England: Yellow Ant, 2010. 501 p.

4. ESRB ratings guides, categories, content descriptors. ESRB Ratings. URL: https://www.esrb.org/ratings-guide/ (date of access: 03.05.2024).

5. Гра Clicker Heroes: вебсайт. URL: https://store.steampowered.com/app/363970/Clicker_Heroes/ (дата звернення: 28.04.2024).

6. Гра Cookie Clicker: вебсайт. URL: https://orteil.dashnet.org/cookieclicker/ (дата звернення: 28.04.2024).

7. Гра Spaceplan: вебсайт. URL: https://store.steampowered.com/app/616110/SPACEPLAN/ (дата звернення: 28.04.2024).

8. Bhadeshia H. K. D. H. Neural Networks in Materials Science: book. ISIJ International 39, 1999. (10): 966–979 p.

9. Усі жанри комп’ютерних ігор. UA PLAY. URL: https://uaplay.com.ua/usizhanry-pc-ihor/ (дата звернення: 03.05.2024).

10. Moore M. Basics of game design. Taylor & Francis Group, 2011. 378 p.

11. Quandt T., Kowert R. Video game debate: unravelling the physical, social, and psychological effects of video games. Taylor & Francis Group, 2015. 204 p.

12. Технічна документація Microsoft. Microsoft; вебсайт. URL: https://docs.microsoft.com/ (дата звернення: 12.04.2024).

13. Офіційний сайт Unity3D: вебсайт. URL: https://unity3d.com/ (дата звернення: 28.03.2024).

14. Офіційний сайт Visual Studio: вебсайт. URL: https://www.visualstudio.com/ (дата звернення: 04.04.2024).

15. Gold J. Object-Oriented Game Development: підручник. UK: Pearson Education Limited, 2004. 404 с.

16. Gregory J. Game Engine Architecture: підручник. USA: A K Peters / CRC Press, 2009. 864 с.

17. Unity Manual, Unity Documentation: документація. URL: https://docs.unity3d.com/Manual (дата звернення: 12.03.2024).

18. Докуметнація мови C#. Microsoft: вебсайт. URL: https://docs.microsoft.com/en-us/dotnet/csharp/ (дата звернення 14.04.2024)

19. Calabrese D. Unity 2D game development. Packt Publishing, 2014. 126 p.

20. Unity game development essentials. Packt Publishing, 2009. 203 p.

21. Thorn A. Mastering unity scripting. Packt Publishing, 2015. 380 p.

22. Корнілов А.В. UNITY. Повне керівництво. Наука і техніка. 2021. 209 с.

23. Buttfield-Addison P., Manning J., Nugent T. Unity game development cookbook: essentials for every game. O'Reilly Media, 2019. 406 p.

ДОДАТОК А

GameController.cs

```
using DG.Tweening;
using System;
using System.Collections.Generic;
using TMPro;
using UnityEngine;
using UnityEngine.UI;
using UnityEngine.SceneManagement;

public class GameController : MonoBehaviour
{
   public double money;
   public double moneyPerSec
   {
      get
      {
         return Math.Ceiling(healthCap / 14) / healthCap * dps;
      }
   }
   public double dpc;
   public double dps;
   public double health;
   public double healthCap
   {
      get
      {
         return (10 * System.Math.Pow(2, stage - 1) * isBoss);
      }
   }

   public float timer;

   public int stage;
   public int stageMax;
   public int kills;
   public int killsMax;
   public int isBoss;
   public int timerCap;

   public Text moneyText;
   public Text dPCText;
   public Text dPSText;
   public Text stageText;
   public Text killsText;
   public Text healthText;
   public Text timerText;

   public GameObject back;
   public GameObject forward;

   public Image healthBar;
   public Image timerBar;

   public Animator coinExplode;
```

```
public GameObject coinExplodeGameObject;

//Anim
public GameObject _planetObj;
public GameObject _backgroundObj;
public GameObject _bossBgObj;

//Offline
public DateTime currentDate;
public DateTime oldTime;
public int OfflineProgressCheck;
public float idleTime;
public Text offlineTimeText;
public float saveTime;
public GameObject offlineBox;


//Mult
public Text multText;
public double multValue;
public float timerMult;
public float timerMultCap;
public double multValueMoney;
public GameObject multBox;

// Username
public TMP_InputField usernameInput;
public string username;
public Text usernameText;
public GameObject usernameBox;

//Upgrades
public Text pCostText;
public Text pLevelText;
public Text pPowerText;
public GameObject upgradeBox;

//Change planet
public Image planetImage;
public int currentPlanetIndex = 0;
public Sprite[] planetSprites;

public double pCost
{
   get
   {
      return 10 * Math.Pow(1.11, pLevel);
   }
}

public int pLevel;
public double pPower
{
   get
   {
      return 5 * Math.Pow(1.07, pLevel);
   }
```

```
}

public Text cCostText;
public Text cLevelText;
public Text cPowerText;
public double cCost
{
   get
   {
      return 10 * Math.Pow(1.11, cLevel);
   }
}

public int cLevel;
public double cPower
{
   get
   {
      return 2 * Math.Pow(1.07, cLevel);
   }
}
//boss bg
public Image bgBoss;

//Audio
AudioManager audioManager;

private void Awake()
{
   audioManager = GameObject.FindGameObjectWithTag("Audio").GetComponent<AudioManager>();
}

public void Start()
{
   offlineBox.gameObject.SetActive(false);
   multBox.gameObject.SetActive(false);
   upgradeBox.gameObject.SetActive(false);


   Load();
   LoadEnemySprites();
   UpdateEnemyImage();
   if (username == "<Username>" )
   {
      usernameBox.gameObject.SetActive(true);
   }
   else
   {
      usernameBox.gameObject.SetActive(false);
   }
   IsBossChecker();
   health = healthCap;
   timerCap = 30;
   multValue = new System.Random().Next(20, 100);
   timerMultCap = new System.Random().Next(10, 20);
   timerMult = timerMultCap;
```

```
}

public void Update()
{
    if (health <= 0)
    {
        Kill();
        ChangeEnemy();
    }
    else
        health -= dps * Time.deltaTime;

    //Mult
    multValueMoney = multValue * moneyPerSec;
    multText.text = "$" + WordNotation(multValueMoney, "F2");

    if (timerMult <= 0) multBox.gameObject.SetActive(true);
    else
        timerMult -= Time.deltaTime;
    moneyText.text = WordNotation(money, "F2");
    stageText.text = "Stage - " + stage;
    killsText.text = kills + "/" + killsMax + " Kills";
    healthText.text = WordNotation(health, "F2") + "/" + WordNotation(healthCap, "F2") + " HP";
    dPCText.text = WordNotation(dpc, "F2") + " dpc";
    dPSText.text = WordNotation(dps, "F2") + " dps";

    healthBar.fillAmount = (float)(health / healthCap);

    if (stage > 1)
    {
        back.gameObject.SetActive(true);
    }
    else
    {
        back.gameObject.SetActive(false);
    }

    if (stage != stageMax)
    {
        forward.gameObject.SetActive(true);
    }
    else
    {
      forward.gameObject.SetActive(false);
    }
    IsBossChecker();
    usernameText.text = username;
    Upgrades();

    saveTime += Time.deltaTime;
    if (saveTime >= 3)
    {
        saveTime = 0;
        Save();
    }
}
```

```
//enemy changer
public void LoadEnemySprites()
{
   // Download all sprites enemies
   planetSprites = Resources.LoadAll<Sprite>("Planets 2.0 Preview");
   if (planetSprites.Length == 0)
   {
      Debug.LogError("No enemy sprites found in Resources/Planets 2.0 Preview");
   }
}

public void ChangeEnemy()
{
   currentPlanetIndex = (currentPlanetIndex + 1) % planetSprites.Length;
   UpdateEnemyImage();
}

public void UpdateEnemyImage()
{
   if (planetSprites.Length > 0)
   {
      planetImage.sprite = planetSprites[currentPlanetIndex];
   }
}

//Upgrades
public void Upgrades()
{
   cCostText.text = "Cost: $" + WordNotation(cCost, "F2");
   cLevelText.text = "Level: " + WordNotation(cLevel, "");
   cPowerText.text = "+" + WordNotation(cPower, "F2") + " dpc";

   pCostText.text = "Cost: $" + WordNotation(pCost, "F2");
   pLevelText.text = "Level: " + WordNotation(pLevel, "");
   pPowerText.text = "+" + WordNotation(pPower, "F2") + " dps";

}
public void UsernameChange()
{
   username = usernameInput.text;
}

public void CloseUsernameBox()
{
   usernameBox.gameObject.SetActive(false);
}
public void IsBossChecker()
{
   if (stage % 5 == 0)
   {
      isBoss = 10;
      stageText.text = "BOSS Stage - " + stage;
      timer -= Time.deltaTime;
      if (timer <= 0) Back();

      timerText.text = timer.ToString("F2") + "/" + timerCap;
      timerBar.gameObject.SetActive(true);
```

```
            timerBar.fillAmount = timer / timerCap;
            killsMax = 1;
            bgBoss.gameObject.SetActive(true);
        }
        else
        {
            isBoss = 1;
            stageText.text = "Stage - " + stage;
            timerText.text = "";
            timerBar.gameObject.SetActive(false);
            timer = 30;
            killsMax = 10;
            bgBoss.gameObject.SetActive(false);
        }
    }

    public void Hit()
    {
        health -= dpc;
        if (health <= 0)
        {
            Kill();
            ChangeEnemy();
        }
        AnimationClicked();
        audioManager.PlaySFX(audioManager.click);
    }


    public void AnimationClicked()
    {
        _planetObj.transform.DOBlendableScaleBy(new Vector3(0.05f, 0.05f, 0.05f),
0.05f).OnComplete(PlanetScaleBack);
        _backgroundObj.transform.DOBlendableScaleBy(new Vector3(0.05f, 0.05f, 0.05f),
0.05f).OnComplete(BackgroundScaleBack);
        _bossBgObj.transform.DOBlendableScaleBy(new Vector3(0.05f, 0.05f, 0.05f),
0.05f).OnComplete(BossBgScaleBack);
    }
    private void PlanetScaleBack()
    {
        _planetObj.transform.DOBlendableScaleBy(new Vector3(-0.05f, -0.05f, -0.05f), 0.05f);
    }

    private void BackgroundScaleBack()
    {
        _backgroundObj.transform.DOBlendableScaleBy(new Vector3(-0.05f, -0.05f, -0.05f), 0.05f);
    }

    private void BossBgScaleBack()
    {
        _bossBgObj.transform.DOBlendableScaleBy(new Vector3(-0.05f, -0.05f, -0.05f), 0.05f);
    }

    public void Kill()
    {
        money += Math.Ceiling(healthCap / 14);
        coinExplode.Play("CoinExplode", 0, 0);
```

```
    if (stage == stageMax)
    {
        kills += 1;
        if (kills >= killsMax)
        {
            kills = 0;
            stage += 1;
            stageMax += 1;
        }
    }
    IsBossChecker();
    health = healthCap;
    if (isBoss > 1) timer = timerCap;
    killsMax = 10;
    audioManager.PlaySFX(audioManager.kill);
}


public void Back()
{
    stage -= 1;
    IsBossChecker();
    health = healthCap;
}

public void Forward()
{
    stage += 1;
    IsBossChecker();
    health = healthCap;
}


public void BuyUpgrade(string id)
{
    switch (id)
    {
        case "p1":
            if (money >= pCost)
            {
                UpdateDefaults(ref pLevel, pCost);
                dps += pPower;
                audioManager.PlaySFX(audioManager.buy);
            }
            break;
        case "c1":
            if (money >= cCost)
            {
                UpdateDefaults(ref cLevel, cCost);
                dpc += 1 + cPower;
                audioManager.PlaySFX(audioManager.buy);
            }
            break;
    }
}

public void UpdateDefaults(ref int level, double cost)
```

```
{
    money -= cost;
    level++;
}

public string WordNotation(double number, string digits)
{
    double digitsTemp = (Math.Floor(Math.Log10(number)));
    IDictionary<double, string> prefixes = new Dictionary<double, string>()
    {
        { 3, "K"},
        { 6, "M"},
        { 9, "B"},
        { 12, "T"},
        { 15, "q"},
        { 18, "Q"},
        { 21, "s"},
        { 24, "S"},
        { 27, "O"},
        { 30, "N"},
        { 33, "d"},
        { 36, "U"},
        { 39, "D"},
        { 42, "!"},
        { 45, "*"},
        { 48, "@"},
        { 51, "#"},
        { 54, "&"},
    };
    double digitsEvery3 = 3 * Math.Floor(digitsTemp / 3);
    if (number >= 1000)
        return (number / Math.Pow(10, digitsEvery3)).ToString(digits) + prefixes[digitsEvery3];
    return number.ToString(digits);
}

public void Save()
{
    OfflineProgressCheck = 1;
    PlayerPrefs.SetString("money", money.ToString());
    PlayerPrefs.SetString("dpc", dpc.ToString());
    PlayerPrefs.SetString("dps", dps.ToString());
    PlayerPrefs.SetString("username", username);
    PlayerPrefs.SetInt("stage", stage);
    PlayerPrefs.SetInt("stageMax", stageMax);
    PlayerPrefs.SetInt("kills", kills);
    PlayerPrefs.SetInt("killsMax", killsMax);
    PlayerPrefs.SetInt("isBoss", isBoss);
    PlayerPrefs.SetInt("cLevel", cLevel);
    PlayerPrefs.SetInt("pLevel", pLevel);
    PlayerPrefs.SetInt("OfflineProgressCheck", OfflineProgressCheck);
    PlayerPrefs.SetInt("currentPlanetIndex", currentPlanetIndex);

    PlayerPrefs.SetString("OflineTime", DateTime.Now.ToBinary().ToString());
}

public void Load()
{
```

```
        money = double.Parse(PlayerPrefs.GetString("money", "0"));
        dpc = double.Parse(PlayerPrefs.GetString("dpc", "1"));
        dps = double.Parse(PlayerPrefs.GetString("dps", "0"));
        stage = PlayerPrefs.GetInt("stage", 1);
        stageMax = PlayerPrefs.GetInt("stageMax", 1);
        kills = PlayerPrefs.GetInt("kills", 0);
        killsMax = PlayerPrefs.GetInt("killsMax", 10);
        isBoss = PlayerPrefs.GetInt("isBoss", 1);
        cLevel = PlayerPrefs.GetInt("cLevel", 0);
        pLevel = PlayerPrefs.GetInt("pLevel", 0);
        OfflineProgressCheck = PlayerPrefs.GetInt("OfflineProgressCheck", 0);
        currentPlanetIndex = PlayerPrefs.GetInt("currentPlanetIndex", 0);

        username = PlayerPrefs.GetString("username", "<Username>");
        LoadOfflineProduction();
    }
    public void LoadOfflineProduction()
    {
        if (OfflineProgressCheck == 1)
        {
            offlineBox.gameObject.SetActive(true);
            long previousTime = Convert.ToInt64(PlayerPrefs.GetString("OflineTime"));
            oldTime = DateTime.FromBinary(previousTime);
            currentDate = DateTime.Now;
            TimeSpan difference = currentDate.Subtract(oldTime);
            idleTime = (float)difference.TotalSeconds;

            var moneyToEarn = Math.Ceiling(healthCap / 14) / healthCap * (dps / 5) * idleTime;
            money += moneyToEarn;
            TimeSpan timer = TimeSpan.FromSeconds(idleTime);

            offlineTimeText.text = "Welcome Back!" + "\nYou were gone for: " + timer.ToString(@"hh\:mm\:ss") +
"\nYou earned: $" + WordNotation(moneyToEarn, "F2");

        }
    }

    public void CloseOfflineBox()
    {
        offlineBox.gameObject.SetActive(false);
    }

    public void OpenMult()
    {
        multBox.gameObject.SetActive(false);
        money += multValueMoney;
        timerMultCap = new System.Random().Next(10, 20);
        timerMult = timerMultCap;
        multValue = new System.Random().Next(20, 100);
    }

    public void OpenUpgrades()
    {
        upgradeBox.gameObject.SetActive(true);
    }

    public void CloseUpgrades()
```

```
    {
        upgradeBox.gameObject.SetActive(false);
    }
}
```

MenuController.cs

```
using UnityEngine;
using UnityEngine.SceneManagement;

public class MenuController : MonoBehaviour
{
    public GameObject infoBox;
    public GameObject settingsBox;

    public void Start()
    {
        infoBox.SetActive(false);
        settingsBox.SetActive(false);
    }
    public void StartGame()
    {
        Time.timeScale = 1;
        SceneManager.LoadScene("PlanetClicker");
    }

    public void MenuNewGame()
    {
        PlayerPrefs.DeleteAll();
        PlayerPrefs.Save();
        Time.timeScale = 1;
        SceneManager.LoadScene("PlanetClicker");
    }

    public void QuitGame()
    {
        Application.Quit();
        Debug.Log("You are leave the game!");
    }

    public void OpenInfoBox()
    {
        infoBox.SetActive(true);
    }
    public void CloseInfoBox()
    {
        infoBox.SetActive(false);

    }

    public void OpenSettingsBox()
    {
        settingsBox.SetActive(true);
    }
    public void CloseSettingsBox()
    {
        settingsBox.SetActive(false);

    }
}
```

PauseController.cs

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

```
using UnityEngine.SceneManagement;

public class PauseConroller : MonoBehaviour
{
   //PauseMenu
   public GameObject pauseBox;

   public void Start()
   {
      pauseBox.gameObject.SetActive(false);
   }
      public void OpenPause()
   {
      Time.timeScale = 0;
      pauseBox.gameObject.SetActive(true);
   }

   public void ClosePause()
   {
      Time.timeScale = 1;
      pauseBox.gameObject.SetActive(false);
   }

   public void GoToMenu()
   {
      SceneManager.LoadScene("MainMenu");
   }

   public void NewGame()
   {
      PlayerPrefs.DeleteAll();
      PlayerPrefs.Save();
      Time.timeScale = 1;
      pauseBox.gameObject.SetActive(false);

      SceneManager.LoadScene("PlanetClicker");
   }

}
```

AudioController.cs

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

public class AudioManager : MonoBehaviour
{
   [Header("----- Audio Mixer -----")]
   [SerializeField] private AudioMixer myMixer;
   [SerializeField] private Slider musicSlider;
   [SerializeField] private Slider sfxSlider;

   [Header("----- Audio Source -----")]
   [SerializeField] AudioSource musicSource;
   [SerializeField] AudioSource SFXSource;

   [Header("----- Audio Clip -----")]
   public AudioClip background;
   public AudioClip click;
   public AudioClip kill;
   public AudioClip buy;
```

```
    public void Start()
    {
        if (PlayerPrefs.HasKey("musicVolume"))
        {
            LoadVolume();
        }
        else
        {
            SetMusicVolume();
            SetSfxVolume();
        }

        if (PlayerPrefs.HasKey("sfxVolume"))
        {
            LoadVolume();
        }
        else
        {
            SetSfxVolume();
        }
        musicSource.clip = background;
        musicSource.Play();
    }
    public void PlaySFX(AudioClip clip)
    {
        SFXSource.PlayOneShot(clip);
    }
    public void SetMusicVolume()
    {
        float volume = musicSlider.value;
        myMixer.SetFloat("music", Mathf.Log10(volume) * 20);
        PlayerPrefs.SetFloat("musicVolume", volume);
    }

    public void SetSfxVolume()
    {
        float volume = sfxSlider.value;
        myMixer.SetFloat("sfx", Mathf.Log10(volume) * 20);
        PlayerPrefs.SetFloat("sfxVolume", volume);
    }

    public void LoadVolume()
    {
        musicSlider.value = PlayerPrefs.GetFloat("musicVolume");
        sfxSlider.value = PlayerPrefs.GetFloat("sfxVolume");
        SetMusicVolume();
        SetSfxVolume();
    }

}
```