ЗВО УНІВЕРСИТЕТ КОРОЛЯ ДАНИЛА

Факультет суспільних та прикладних наук
Кафедра інформаційних технологій

на правах рукопису

Литвак Володимир Юрійович

УДК 004.378

Розробка 2D відеогри для мобільної платформи Android із жанру «Run or Die»

Спеціальність 121 – «Інженерія програмного забезпечення»
Кваліфікаційна робота на здобуття кваліфікації бакалавра

Нормоконтроль

_____Стисло О.В.
(підпис, дата, розшифрування підпису)

Студент

_____Литвак В.Ю.
(підпис, дата, розшифрування підпису)

Допускається до захисту
Завідувач кафедри

_____к.т.н., доц. Пашкевич О.П.
(підпис, дата, розшифрування підпису)

Керівник роботи

_____ к.ф-м.н., доц. Бойчук А
(підпис, дата, розшифрування підпису)

Івано-Франківськ – 2023

ЗВО УНІВЕРСИТЕТ КОРОЛЯ ДАНИЛА

Факультет суспільних та прикладних наук

Кафедра інформаційних технологій

Освітній ступінь: «бакалавр»

Спеціальність: 121 «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____

« ____ »_____ 2023 року

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Литвак Володимир Юрійович

(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи

Розробка 2D відеогри для мобільної платформи Android із жанру «Run or Die»

керівник роботи:

Бойчук Андрій Михайлович, фізико-математичних наук, доцент кафедри ІТ

затверджена наказом вищого навчального закладу від « 11» листопада 2022 року

№ 155/1НВ

2. Термін подання студентом роботи 14.06.2023

3. Вихідні дані роботи: Мова програмування C#

4. Зміст кваліфікаційної роботи

1. Розробка відеогри «Run or Die»

2. Технології та інструменти відеогри

3. Розробка структури відеогри

5. Дата видачі завдання 10.10.2022

КОНСУЛЬТАНТИ РОЗДІЛІВ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Розділ	Консультант (прізвище, ініціали та посада)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Термін виконання етапів роботи	Примітка
1.	Вибір теми та технологій розробки	15.02.2023	Виконано
2.	Опис предметної області	14.03.2023	Виконано
3.	Аналіз вимог	07.04.2023	Виконано
4.	Розробка гри	28.04.2023	Виконано
5.	Розробка текстур та моделей	05.05.2023	Виконано
6.	Розробка музичного супроводу	19.05.2023	Виконано
7.	Оформлення пояснювальної записки дипломної роботи завідувачем кафедри	03.06.2023	Виконано

Студент _____ Литвак В.Ю.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Бойчук А.М.
(підпис) (прізвище та ініціали)

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

Сторінка	Опис графічного матеріалу	Сторінка	Опис графічного матеріалу
21	Use-Case	46	LocCharList
22	Діаграма діяльності	47	StartGame
24	Діаграма класів		
26	Основна структура поєкту		
27	Уривок коду Gameplay		
27	Уривок коду Gameplay		
29	Camera Move		
32	BoxLogic		
33	SharkLogic		
37	StoneGenerator		
39	TrapLogoc		
43	Location		
44	Location		
45	GameData		

АНОТАЦІЯ

У дипломній роботі здійснено розробку мобільної гри у жанрі Endless runner з елементами platformer та arcade на Unity з використанням мови програмування C#

В першому розділі виконаний опис предметної області та аналіз існуючих рішень.

В другому розділі проведений аналіз вимог та описана специфікація вимог.

У третьому розділі описані використані технології та патерни.

Четвертий та п’ятий розділ виділені на детальний опис розробки, тестування й тд.

Ключові слова: Unity, arcade, android, C#

ANNOTATION

In the diploma work with the effective development of a mobile game in the genre of Endless runner with platformer and arcade elements on Unity using the C# programming language

In the first section, a description of the subject area and an analysis of existing solutions are performed.

In the second section, the analysis of the requirements was carried out and the specification of the requirements was described.

The third section describes the technologies and patterns used.

The fourth and fifth chapters are divided into a detailed description of development, testing, etc.

Key words: Unity, arcade, android, C#

ЗМІСТ

ВСТУП

У сучасному світі мобільні ігри здобувають все більшу популярність серед користувачів. Їх привабливість полягає у можливості отримати захоплюючий ігровий досвід прямо на своєму смартфоні або планшеті. Одним з популярних жанрів серед мобільних ігор є "Endless runner", який пропонує гравцям не кінцевий шлях, де вони повинні уникати перешкод, збирати бонуси і пройти якомога більше відстані.

Ця дипломна робота присвячена розробці мобільної гри у жанрі "Endless runner" з елементами platformer та arcade. Метою роботи є створення захоплюючої та динамічної гри, яка надасть користувачам можливість насолоджуватися геймплеєм у будь-який час та в будь-якому місці.

Для реалізації проекту буде використано інструментарій Unity - потужну інтегровану середу розробки для створення ігор, яка надає широкі можливості для реалізації різноманітних ігрових механік і ефектів. Мова програмування C# буде використана для розробки логіки гри та взаємодії різних компонентів.

У рамках цієї роботи будуть досліджені та реалізовані основні елементи гри, такі як: система керування гравцем, генерація рівнів, взаємодія з перешкодами та бонусами, система очків та рекордів, а також створення захоплюючої геймплейної інтеракції для користувачів.

Очікується, що результатом дипломної роботи буде повноцінна мобільна гра, яка пропонує захоплюючий геймплей та задовольняє вимоги сучасних гравців. Розроблена гра може бути розповсюджена через мобільні платформи та приносити задоволення користувачам у світі мобільних ігор.

Розробити мобільну гру у жанрі Endless runner з елементами platformer та arcade на Unity з використанням мови програмування C# та вивчити технології, що використовуються при розробці мобільних ігор.

Об’єкт дослідження

Об’єктом дослідження є розробка мобільної гри у жанрі Endless runner з елементами platformer та arcade на платформі Unity з використанням мови програмування C#. Гра в цьому жанрі вимагає від розробників вирішення різних технічних та творчих завдань, що робить її цікавим об’єктом дослідження.

Предмет дослідження

Предметом дослідження є особливості розробки гри. Дослідження дозволяє зосередитися на процесі розробки гри та вивчити технологічні, дизайнерські та геймплейні аспекти цього процесу.

Мета:

1. Дослідити жанр Endless Runner, платформерів та аркадних ігор, вивчити їх особливості, геймплейні механіки та успішні приклади.

2. Визначити ключові елементи, які будуть включені в розроблену гру, зокрема, безкінечний рух персонажа, перешкоди, збір предметів та бонусів.

3. Розробити концепцію гри, включаючи стиль, графіку та звуковий дизайн.

4.Реалізувати основні механіки ігри, такі як управління персонажем, обробку колізій та баланс геймплею.

5. Розробити систему прогресу гравця, включаючи набрані очки, досягнення та можливості покращення персонажа.

6. Протестувати гру, виявити та виправити помилки та баги, оптимізувати продуктивність та взаємодію з користувачем.

7. Підготувати документацію, включаючи технічний опис гри, пояснювальну записку та інструкцію з використання.

В результаті виконання цих завдань буде створено повноцінну мобільну гру у жанрі Endless Runner з елементами платформера та аркади, яка забезпечує користувачам приємні емоції від гри.

РОЗДІЛ 1. ОПИС ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Обґрунтування актуальності теми

Мобільні пристрої стали невід'ємною частиною нашого повсякденного життя, і все більше людей користується ними для розваг. Мобільні ігри займають передову позицію серед розважальних додатків, тому розробка мобільних ігор є актуальною та вигідною сферою для дослідження та реалізації.

Завдяки розвитку технологій розробка мобільних ігор стала доступнішою і швидшою. Unity є однією з найпопулярніших інтегрованих середовищ розробки для створення ігор, що надає широкі можливості для реалізації ігрових механік та ефектів. Мова програмування C# є однією з найпоширеніших мов для розробки ігор на Unity, що дозволяє розробникам швидко та ефективно створювати ігровий функціонал[3].

Отже, розробка мобільної гри у жанрі Endless runner з елементами platformer та arcade на Unity з використанням мови програмування C# є актуальною темою, яка поєднує популярні жанри та технології розробки ігор, і має потенціал привернути увагу та задовольнити потреби сучасних гравців.

Об’єктом дослідження є розробка мобільної гри у жанрі Endless runner з елементами platformer та arcade на платформі Unity з використанням мови програмування C#. Гра в цьому жанрі вимагає від розробників вирішення різних технічних та творчих завдань, що робить її цікавим об’єктом дослідження.

Предметом дослідження є особливості розробки гри. Дослідження дозволяє зосередитися на процесі розробки гри та вивчити технологічні, дизайнерські та геймплейні аспекти цього процесу[2,4].

1.2 Методи дослідження

Для виконання роботи були використані наступні методи дослідження:

1. Аналіз літератури: вивчення наукових та технічних джерел, статей, публікацій та книг, пов'язаних з розробкою ігор. Цей аналіз допоміг збагатити розуміння предметної області, виявити успішні приклади та найкращі практики.

2. Дослідження існуючих ігор: аналіз та дослідження інших ігор, зокрема таких як Temple Run, Subway Surfers, Jetpack Joyride тощо. Це дозволяє виявити спільні та унікальні особливості цих ігор, а також зрозуміти, як вони були реалізовані та успішно впроваджені.

3. Експериментальні дослідження: розробка прототипів гри, тестування різних геймплейних механік та функціональності, проведення користувацьких тестів для збору фідбеку та оцінки задоволення користувачів. Це дозволяє перевірити припущення, виправити помилки та вдосконалити гру на ранніх етапах розробки.

Ці методи дослідження дозволяють здійснити глибокий аналіз предметної області, визначити найкращі практики та вдосконалення, а також забезпечити високу якість та задоволення від розроблюваної мобільної гри.

1.3 Практичне значення отриманих результатів

Практичне значення полягає в наступному:

1. Розробка гри для мобільних платформ: Результатом дипломної роботи буде готова мобільна гра, яка може бути опублікована в мобільних маркетплейсах, таких як App Store та Google Play. Це дає можливість користувачам насолоджуватись грою на своїх мобільних пристроях та дозволяє розробникам продемонструвати свої навички та творчість.

2. Розширення навичок розробки ігор: Розробка мобільної гри на Unity з використанням мови програмування C# вимагає глибокого розуміння

технологій, геймдизайну та програмування. Виконання дипломної роботи дозволить поглибити свої знання та навички у цих областях, що може бути корисним для подальшої кар'єри в галузі розробки ігор[24].

3. Приклад успішної реалізації гри у жанрі Endless Runner: Дипломна робота може послужити як приклад успішної реалізації гри у жанрі Endless Runner з елементами платформера та аркади. Інші розробники можуть використовувати цей досвід та реалізацію як основу для своїх власних проектів або вдосконалення існуючих ігор.

4. Потенційна комерційна цінність: Успішна мобільна гра може мати потенційну комерційну цінність, забезпечуючи прибуток розробнику через продажі, рекламу, внутрішні покупки тощо. Якщо гра набуває популярності та отримує позитивні відгуки, це може привести до успіху на ринку мобільних ігор.

Отже, практичне значення дипломної роботи полягає в розробці та випуску гри, розширенні навичок розробки ігор, наданні прикладу успішної реалізації гри у жанрі Endless Runner та потенційній комерційній цінності.

1.4 Аналіз існуючих рішень

Перед розробкою потрібно проаналізувати вже готові рішення від конкурентів, щоб зрозуміти, що можна запозичити, а що покращити, зрозуміти, що саме потрібно для реалізації.

У жанрі Endless runner доволі популярними є Alto's Adventure та Alto's Odyssey, Jetpack Joyride, Temple Run та Subway Surfers[25].

//Alto's Adventure та Alto's Odyssey зроблені в мінімалістичному стилі.

Я вирішив зробити щось середнє між Jetpack Joyride та Temple Run.

Аналіз існуючих ігор допомагає зрозуміти, які елементи та підходи можна використати у власному проекті, а також розробити стратегію створення унікального та привабливого геймплею.

Отже, після аналізу можна зробити аналіз вимог.

1.5 Огляд досліджень в галузі розробки мобільних ігор

Дослідження в галузі розробки мобільних ігор включають в себе різні аспекти, такі як технології, тренди, дизайн, монетизація, користувацький досвід та багато іншого. Нижче подано огляд деяких ключових досліджень та тенденцій у цій галузі:

1. Мобільні платформи: Одні з найпопулярніших мобільних платформ для розробки ігор - це iOS та Android. Дослідження орієнтовані на оптимізацію та підтримку цих платформ, включаючи нові функції, версії операційних систем та розробницькі інструменти.

2. Ігрові движки: Ігрові движки, такі як Unity, Unreal Engine та Cocos2d, відіграють важливу роль у розробці мобільних ігор. Дослідження спрямовані на вдосконалення та оптимізацію цих інструментів, щоб забезпечити більшу продуктивність та зручність для розробників.

3. Розробка мультиплатформових ігор: Зростаюча популярність мультиплатформових ігор, які можна грати на різних пристроях, включаючи смартфони, планшети та ПК. Дослідження спрямовані на розробку стратегій та підходів до створення мультиплатформових ігор, що забезпечують сумісність та зручність гри на різних пристроях.

4. Розширена реальність (AR) та віртуальна реальність (VR): AR та VR відкривають нові можливості для мобільних ігор, дозволяючи гравцям зануритися у віртуальний світ або поєднувати віртуальні об'єкти з реальними. Дослідження спрямовані на вдосконалення технологій AR та VR, а також на розробку ігор, які максимально використовують їх можливості.

5. Монетизація: Дослідження також зосереджені на різних моделях монетизації мобільних ігор, включаючи рекламу, покупки в програмі (in-app purchases), підписки та інші стратегії. Ці дослідження допомагають розробникам знайти ефективні способи отримання прибутку зі своїх ігор.

6. Соціальні мобільні ігри: Значна частина мобільних ігор базується на соціальному взаємодії та мережевій грі. Дослідження орієнтовані на вивчення соціального впливу та взаємодії гравців у мобільних іграх, а також на розробку функцій, що підтримують соціальний аспект гри.

7. Вивчення користувацького досвіду: Дослідження в галузі розробки мобільних ігор також орієнтовані на вивчення користувацького досвіду та впливу різних факторів на задоволення гравців. Ці дослідження допомагають розробникам створювати більш привабливі та задоволені гри[15].

1.6 Опис предметної області

Предметна область розробки мобільних ігор охоплює всі аспекти, пов'язані з процесом створення та розповсюдження ігор для мобільних пристроїв, таких як смартфони та планшети. Це включає в себе технічні аспекти розробки, дизайн, монетизацію, маркетинг, користувацький досвід та інше.

Основні складові предметної області розробки мобільних ігор включають:

1. Технології розробки: Мобільні ігри можуть бути розроблені з використанням різних технологій та програмних інструментів. Це включає мови програмування, такі як Java, C++, C#, Objective-C, Swift, а також ігрові движки, такі як Unity, Unreal Engine, Cocos2d і інші.

2. Дизайн ігор: Дизайн мобільних ігор включає в себе створення геймплею, визначення вигляду інтерфейсу користувача, графіки, звуків та інших аспектів, що впливають на візуальний та звуковий досвід гравців.

3. Монетизація: Одним із ключових аспектів розробки мобільних ігор є їх монетизація, тобто заробіток на іграх. Це може включати в себе різні моделі, такі як реклама, покупки в програмі (in-app purchases), підписки, спонсорські контракти та інші стратегії.

4. Маркетинг і розповсюдження: Успіх мобільної гри залежить від її популярності серед гравців. Ефективний маркетинг та стратегії розповсюдження

можуть допомогти залучити більше користувачів. Це включає в себе просування гри в магазинах додатків, соціальні медіа, рекламні кампанії та інші маркетингові зусилля.

5. Користувацький досвід: Досвід гравців є критичним фактором успіху мобільної гри. Дослідження користувацького досвіду допомагають розробникам зрозуміти, як гравці взаємодіють з грою, як вони реагують на різні елементи геймплею, і як поліпшити загальний досвід гравців.

Вивчення предметної області розробки мобільних ігор допомагає розробникам розуміти потреби та вимоги гравців, впроваджувати нові технології та інновації, а також створювати грові вироби для мобільних пристроїв[21].

1.7 Висновок розділ 1

В розділі було проведено огляд аналогів мобільних ігор у жанрі Endless Runner з елементами Platformer та Arcade. Дослідження показали, що на сьогоднішній день існує значна кількість подібних ігор, які надають користувачам можливість насолоджуватися швидким та захоплюючим геймплеєм.

У аналізі існуючих аналогів було виявлено кілька ключових особливостей. Перш за все, багато ігор у жанрі Endless Runner пропонують різноманітні рівні складності та виклики, що сприяє підвищенню інтересу та залученню гравців. Більшість з них також надають можливість збирати бонуси та покращувати навички головного героя.

В результаті проведених досліджень, визначено потребу в розробці нової мобільної гри у жанрі Endless Runner з елементами Platformer та Arcade. Гра буде містити захоплюючий геймплей, динамічні рівні та різноманітні перешкоди, що забезпечать цікавість та виклики для гравців. Крім того, вона буде мати стильну графіку та приємний звуковий супровід, що створить позитивну атмосферу під час гри.

Отже, розробка нової мобільної гри у жанрі Endless Runner з елементами Platformer та Arcade на Unity з використанням мови програмування C# має потенціал стати захоплюючим та популярним проектом, який приверне увагу широкого кола гравців. Така гра може надати користувачам відмінну можливість провести час з задоволенням та випробувати свої навички у швидкій та захоплюючій грі на мобільних пристроях.

РОЗДІЛ 2. АНАЛІЗ ВИМОГ

2.1 Опис проекту та його сфера застосування

Проект передбачає розробку мобільної гри у жанрі Endless Runner з елементами platformer та arcade на Unity з використанням мови програмування C#. Гра буде створена для мобільних платформ, таких як iOS та Android, і буде доступна для завантаження з мобільних маркетплейсів, наприклад, App Store та Google Play.

Сфера застосування проекту - розважальна галузь ігор. Мобільні ігри у жанрі Endless Runner є дуже популярними серед користувачів мобільних пристроїв, оскільки вони надають швидкий та захоплюючий геймплей. Гравці будуть мати змогу насолоджуватись безкінечним бігом та перешкодами, збираючи бонуси та досягаючи як найвищих результатів. Гра також має елементи платформера та аркади, що робить її ще більш різноманітною та цікавою для гравців.

Сфера застосування проекту може бути різноманітною, включаючи:

1. Розваги: Гра створена для забезпечення розважального відпочинку та задоволення гравців. Вона надає можливість насолоджуватись захоплюючим геймплеєм та випробувати свої навички у керуванні головним персонажем.

2. Мобільна геймінг-індустрія: Завдяки популярності мобільних ігор та жанру Endless Runner, гра може стати успішним комерційним проектом, принести дохід розробнику через продажі, рекламу та внутрішні покупки.

3. Показник навичок: Розробка гри на Unity з використанням мови програмування C# дозволяє продемонструвати свої навички в розробці мобільних ігор та програмуванні, що може бути корисним при пошуку роботи у галузі ігрової розробки.

Отже, проект спрямований на розробку мобільної гри у жанрі Endless Runner з елементами platformer та arcade, яка знаходить застосування в розважальній галузі ігор, мобільний геймінг-індустрії та демонстрації навичок розробки ігор та програмування[4,15].

2.2 Специфікація функціональних вимог

Тут описав функціональні можливості, які повинні бути присутніми для того, щоб користувач міг здійснювати послуги, що надаються програмним забезпеченням, а також варіанти використання для активних дійових осіб.

2.2.1 Use-Case

Use-Case - використовують у системному аналізі для визначення, уточнення та організації вимог до системи. Варіант використання складається з набору можливих послідовностей взаємодій між системами та користувачами в певному середовищі та для певної мети. Метод дає змогу отримати документ, що описує всі кроки користувача для виконання дії.

Ці сценарії пишуться бізнес-аналітиками на всіх етапах розробки ПЗ, вони допомагають розробникам як повинна працювати система й на яких етапах можливе виникнення помилки, щоб завчасно їх виправити. Теж документ із цими сценаріями допоможе створити інструкцію для користувачів.

Кожен варіант використання містить три основні елементи:

Актор - Користувачем системи може бути людина або група людей, які взаємодіють із процесом.

Мета - Кінцевий успішний результат, що завершує процес.

Система - Процес і кроки, зроблені для досягнення кінцевої мети, включно з необхідними функціональними вимогами та їхньою очікуваною поведінкою[14].

В даному випадку я створив загальну діаграму прецедентів для можливого функціоналу для кожної групи користувачів (Рис 2.1).

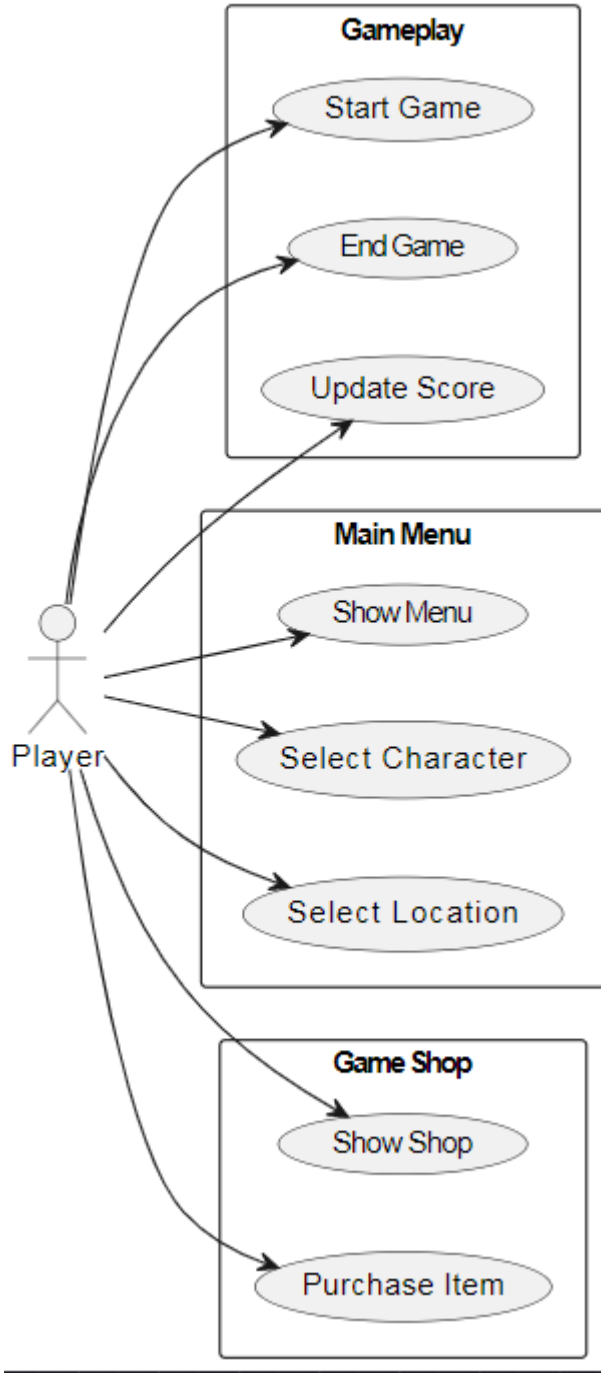


Рисунок 2.1 — Use-Case

2.2.2 Створення діаграми діяльності

Діаграма діяльності - це візуальне представлення послідовності операцій або потоку управління в системі, подібне до діаграми потоків даних або діаграми потоків даних. Діаграми діяльності часто використовуються в моделюванні бізнес-процесів. Вони також можуть описувати кроки на діаграмі варіантів використання. Діяльність, що моделюється, може бути послідовною або паралельною. В обох випадках діаграма діяльності має початок (початковий стан) і кінець (кінцевий стан) (Рис 2.2).

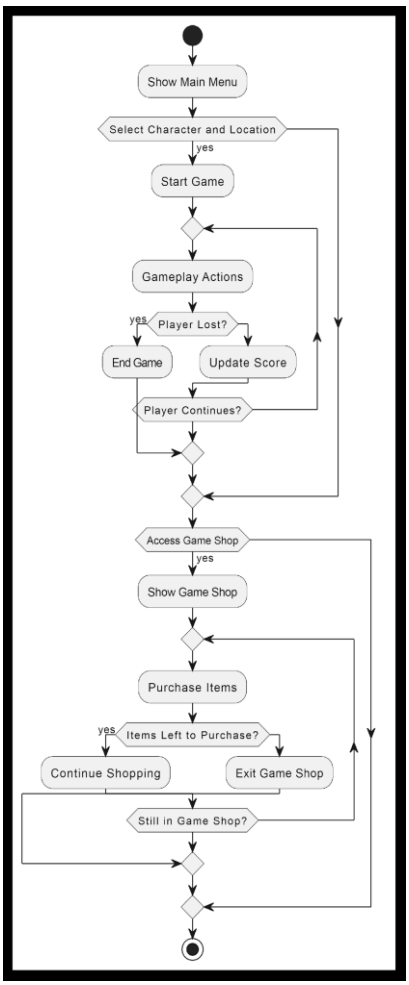


Рисунок 2.2 — Діаграма діяльності.

2.2.3 Створення діаграми класів

Діаграма класів – це UML зображення зв’язків вихідного коду та залежності між класами, воно використовується в об’єктно-орієнтованому програмуванні для визначення конкретних сутностей або одиниць коду. Ці сутності, які називаються об’єктами, визначають методи та змінні в межах класів. Незважаючи на те, що концепція діаграм класів існує вже багато років, вона була вдосконалена з розвитком парадигми моделювання ООП і є корисним інструментом у всіх формах ООП.

На діаграмах класів зазвичай зустрічаються групи класів, які мають подібні характеристики. Квадратне представлення використовується для позначення кожного окремого класу, подібно до блок-схем. Кожен квадрат містить три менших прямокутника, верхній з яких містить назву класу, середній відображає його атрибути, а нижній показує його операції класу. Класи можуть бути з'єднані між собою лінією, на кінцях якої можуть бути або не мати стрілки. Ці лінії, які зв’язують всі класи між собою називаються зв’язками.

Теж існують декілька видів відношення між класами:

- Відношення агрегації – це відношення, за якого два класи, які беруть участь у зв’язку не є рівнозначними, вони мають зв’язок типу «ціле-частина».
- Відношення композиції - це вид агрегації, але тут зв’язок є настільки сильним, що частини не можуть існувати без цілого.
- Відношення наслідування – визначає дочірний та батьківський клас.
- Асоціація – це звичайний зв’язок[16].

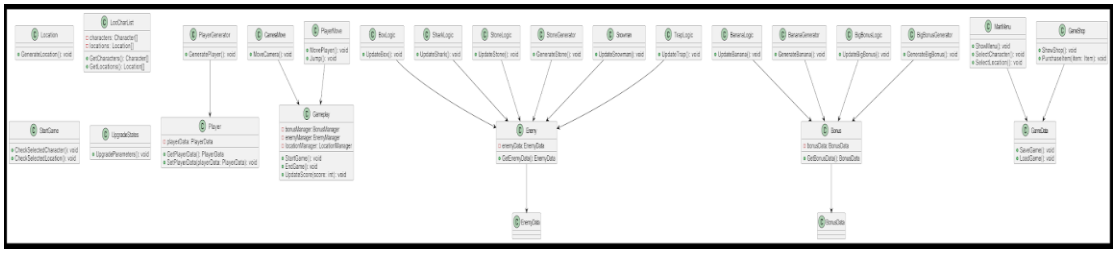


Рисунок 2.3 — Діаграма класів

На зображенні (Рис 2.3) я побудував діаграму класів й зв’язки між ними.

2.3 Використані технології

Основна технологія розробки в мене – це Unity. Я вибрав саме цю технологію, адже вона дуже активно розвивається й є однією із найперспективніших.

Unity надає широкий набір інструментів та функціональностей для розробки ігор. Він має вбудовану графічну систему, фізичний движок, систему анімації, систему управління ресурсами та багато іншого. Крім того, Unity підтримує мову програмування C#, яка є потужним і розповсюдженим інструментом для розробки ігрової логіки та взаємодії.

З використанням Unity можна створювати різноманітні 3D і 2D ігри, налаштовувати графічні ефекти, створювати анімацію персонажів, взаємодіяти з фізикою об'єктів, робити управління гравцем та багато іншого. Unity також підтримує інтеграцію з різними платформами, що дозволяє розгортати гру на різних пристроях, таких як Android, iOS, Windows, Mac тощо.

Unity забезпечує зручний інтерфейс для розробки, що дозволяє вам візуально організовувати ваші об'єкти, компоненти та налаштування. Також можна використовувати магазин активів Unity, де можна знайти готові ресурси, графіку, звуки та інші компоненти, щоб прискорити процес розробки.

Загалом, Unity є потужним інструментом для розробки мобільних ігор, який надає широкі можливості для реалізації поставленого завдання[4,8,15].

2.4 Висновок розділ 2

У другому розділі дипломної роботи "Розробка мобільної гри у жанрі Endless runner з елементами platformer та arcade на Unity з використанням мови програмування C#" було розглянуто кілька важливих аспектів процесу розробки гри.

Починаючи з визначення основних функціональних вимог, були сформульовані потреби гравців, які надали важливі орієнтири для подальшої роботи. Це дозволило уявити ситуацію з перспективи користувача та спрямувати розробку гри на задоволення їхніх потреб та очікувань.

Далі, були розроблені прототипи геймплею та рівнів гри, що надали візуальну та функціональну уяву про вигляд та механіку гри. Це дозволило перевірити й аналізувати елементи гри, виявити потенційні проблеми та зробити відповідні коrективи для поліпшення геймплею.

У розділі також були розглянуті використані технології розробки, зокрема Unity та мова програмування C#. Unity, як потужне інтегроване середовище розробки, забезпечило широкі можливості для створення гри та реалізації різноманітних функцій. Мова програмування C# виявилася ефективним інструментом для реалізації логіки гри та взаємодії об'єктів.

Висновок з даного розділу полягає в тому, що розробка мобільної гри у жанрі Endless runner з елементами platformer та arcade на Unity з використанням мови програмування C# є доцільною та перспективною. Відповідно до вимог та очікувань гравців, була розроблена захоплююча гра з цікавим геймпле

єм та високою рівнем взаємодії. Застосування відповідних технологій та методик розробки гарантує якість та задоволення користувачів грою.

РОЗДІЛ 3. РОЗРОБКА

3.1 Створення основних класів та папок.

Нижче наведено основні класи та їх опис (Рис 3.1).

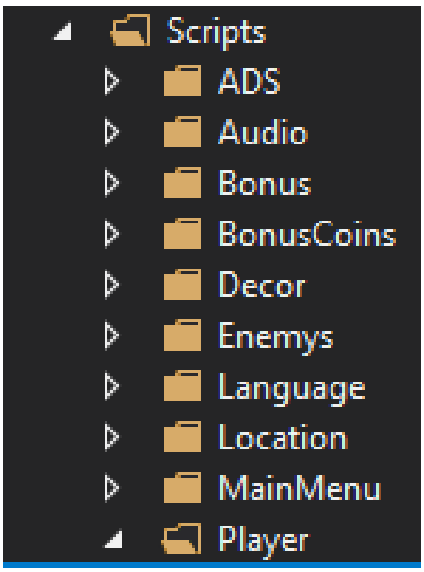


Рисунок 3.1 — Основна структура проєкту

Папка **Player** містить в собі наступні класи:

Клас **Gameplay** відповідає за реалізацію ігрового процесу (взаємодію з бонусами збільшення кількості зароблених балів, взаємодію з небезпечними предметами, взаємодію з локаціями, містить дані про унікальні можливості персонажа); (Рис 3.2 - 3.3).

```
using UnityEngine.SocialPlatforms;

/* Скрипт ігрового процесу*/

public class Gameplay : MonoBehaviour{
    [HideInInspector] private const string leaderBoard = "CgkI3obC5-8HEAIQAQ";

    Animator anim;

    public AdAfterTime adAfterTime;

    public AudioClip coinSound;
    public AudioClip bonusSound;
    public AudioClip overSound;

    float gameTimer = 0; //вираховує час гри, використовується для ускладнення рівня з часом

    int score;
    int coins;

    int coinPlusB = GameData.smallBonusCoinsBase;
    int coinPlusBB = GameData.bigBonusCoinsBase;
    int scorePlus;
    int scorePlusB = GameData.smallBonusScoreBase; //Збільшення кількості балів від бонусів
    int scorePlusBB = GameData.bigBonusScoreBase; //....великих бонусів
```

Рисунок 3.2 — Уривок коду з Gameplay

```

    public Text scoreUI;
    public Text coinsUI;
    public Text endScoreUI;
    public Text endCoinsUI;
    public GameObject gameOverUI;

    public GameObject[] control; //Кнопки управління
```

Рисунок 3.3 — Уривок коду з Gameplay

Цей сценарій відповідає за ігровий процес. Основні функції цього сценарію включають управління грою, розрахунок очків і монет, реакцію на зіткнення з бонусами та перешкодами, відображення екрану гри та екрану закінчення гри.

- Основне що тут було реалізовано:
- дошки лідерів Google Play Games;
- керування анімацією персонажа;
- показ реклами після певного часу гри;
- звукові ефекти,
- час гри, використовується для ускладнення рівня з часом;
- поточний рахунок та кількість монет гравця;
- скільки очків і монет додається при зборі бонусів;
- відображення рахунку та кількість монет гравця в кінці гри

– елементи інтерфейсу для екрану закінчення гри;
– кнопки управління грою;
– реалізація зіткнення з перешкодами;
– генерація великих бонусів;
– пауза
– зберігання даних
– взаємодія з бонусами та перешкодами;
– перезапуск гри після закінчення;
– відображення рахунку та кількості монет на екрані гри;
– збереження даних гравця;

Рагалом в класі 260 рядків тому я не став повністю його приводити. Та в подальшому надто великі класи я буду просто описувати.

Клас **CameraMove** відповідає за рух камери (Рис 3.4);

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

public class CameraMove : MonoBehaviour{
    public float dumping = 0f; // Зменшення швидкості пересування камери
    public Vector3 offset; // Відстань між камерою і гравцем
    public GameObject player; // Посилання на об'єкт гравця

    void Start(){
        // Встановлення початкового значення відстані
        offset = new Vector3(0f, 5.5f, -10f);
    }

    void Update(){
        offset = new Vector3(0f, 5.5f, -10f);
        // Зміщення позиції камери на відстань offset від позиції гравця
        transform.position = player.transform.position + offset;

    }
}

```

Рисунок 3.4 - CameraMove

dumping - змінна, яка визначає зменшення швидкості руху камери

offset - вектор, який визначає відстань між камерою і гравцем. Змінна ініціалізується у методі Start().

player - посилання на об'єкт гравця.

Start() - метод, який викликається один раз при запуску скрипту. У цьому методі встановлюється початкове значення offset.

Update() - метод, який викликається кожен кадр. У цьому методі оновлюється позиція камери на основі позиції гравця та значення offset.

Даний скрипт відповідає за рух камери відносно гравця в грі. Використовуючи вектор offset, відстань між камерою і гравцем залишається сталою. Це створює ефект слідування за гравцем під час його руху.

Клас **PlayerMove**. відповідає за управління рухом гравця в грі. Він містить логіку для керування рухом гравця, включаючи рух вперед і назад, стрибки та анімацію.

Цей клас містить наступні методи та змінні:

- Змінна `speed` визначає швидкість руху гравця.

- Змінна `jumpHeight` визначає висоту стрибка гравця.

- Змінна `velocity` зберігає вектор швидкості гравця.

- Змінна `gravity` визначає значення гравітації, що впливає на рух гравця.

- Змінна `jump` вказує, чи гравець має здійснити стрибок.

- Змінна `fly` вказує, чи гравець знаходиться у повітрі.

- Змінна `isGround` вказує, чи гравець знаходиться на землі.

- Змінні `runRight` та `runLeft` вказують, чи гравець рухається вправо або вліво.

- Об'єкт `joystick` використовується для керування гравцем.

- Метод `Idle()` відтворює анімацію бездіяльності гравця.

- Метод `Jump()` відтворює анімацію стрибка гравця.

- Метод `Update()` викликається на кожному кадрі гри і керує рухом гравця, анімацією та обробляє стрибки.

Клас `PlayerMove` має підключення до компонентів `Animator` та `CharacterController`, які використовуються для управління анімацією та рухом гравця відповідно. Клас також включає обробку подій дотику до екрану за допомогою `EventTrigger`[20].

Клас **PlayerGenerator** відповідає за генерацію та управління гравцем у грі. Він містить логіку для вибору персонажа, створення головного гравця, налаштування компонентів гравця та управління подіями під час гри.

Цей клас має наступні поля та змінні:

- Змінна `adAfterTime` вказує на компонент `AdAfterTime`, який використовується для відтворення реклами після певного часу гри.

- Зміні `coinSound`, `bonusSound` та `overSound` містять аудіокліпи для звукових ефектів гри, таких як звук монети, бонусу та кінця гри.

- Масив `player` містить різні графічні моделі гравців.

- Об'єкт `joystick` використовується для керування гравцем.

- Текстові поля `scoreUI` та `coinsUI` використовуються для відображення рахунку та кількості монет під час гри.

- Текстові поля `endScoreUI` та `endCoinsUI` використовуються для відображення кінцевого рахунку та кількості монет після завершення гри.

- Об'єкти `gameOverUI` та `control` використовуються для управління інтерфейсом під час гри.

- Об'єкт `bigBonusGenerator` використовується для генерації великого бонусу в грі.

- Об'єкт `jumpButton` представляє `EventTrigger`, який використовується для обробки події натискання кнопки стрибка.

Метод `Awake()` викликається при запуску гри і відповідає за створення головного гравця та налаштування його компонентів. Зокрема, він налаштовує керування гравцем, встановлює зв'язки з компонентами головної камери та фону, передає необхідні значення для відображення інформації під час гри та налаштовує звуки гри.

Метод `Update()` перевіряє, чи встановлено флаг `exitGame` у `GameData`. Якщо так, то він завантажує сцену "EnterLocChar", щоб повернутися до вибору місця та персонажа.

Метод `OnDestroy()` викликається перед знищенням об'єкта `PlayerGenerator` і відповідає за знищення головного гравця.

Папка **Enemies** містить дані про ворогів та ворожі предмети:

Клас **BoxLogic** відповідає за логіку небезпечних перешкод на локаціях (Рис 3.5);

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

public class BoxLogic : MonoBehaviour{
    float boxSpeed; // Швидкість руху
    void Start(){
        boxSpeed = 5f; // Встановлення початкової швидкості
    }

    void OnTriggerStay(Collider collision){
        if (collision.CompareTag("ground")){
            if (boxSpeed > 0) boxSpeed -= 0.07f; // Зменшення швидкості, якщо об'єкт знаходиться на землі
        }
    }

    void Update(){
        // Рух об'єкту вниз з заданою швидкістю
        if (boxSpeed > 0)
            transform.position = new Vector3(transform.position.x, transform.position.y - boxSpeed * Time.deltaTime, transform.position.z);
        //знищити об'єкт якщо

        //гра закінчилась      гра перезапускається        гра закривається
        if (GameData.gameOver || GameData.restartGame || GameData.exitGame){
            Destroy(gameObject);
        }
    }
}
```

Рисунок 3.5 - BoxLogic

boxSpeed - змінна, що визначає швидкість руху об'єкту.

Start() - метод, який викликається один раз при запуску скрипту. У цьому методі встановлюється початкова значення швидкості boxSpeed.

OnTriggerStay(Collider collision) - метод, який викликається, коли об'єкт перебуває в колізії з іншим коллайдером. Якщо колізія відбувається з об'єктом, який має тег "ground", швидкість boxSpeed зменшується на 0.07.

Update() - метод, який викликається кожен кадр. У цьому методі реалізовано рух об'єкту вниз зі швидкістю boxSpeed. Також, перевіряється стан гри (GameData.gameOver, GameData.restartGame, GameData.exitGame), і якщо гра

закінчилась або перезапускається або закривається, об'єкт знищується (Destroy(gameObject)).

Клас **SharkLogic** відповідає за логіку ворогів які циклічно рухаються по локації (Рис 3.6);

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

public class SharkLogic : MonoBehaviour{
    float powerMove = 0.2f;
    float moveBorder = 0f;
    bool moveUp = true;

    float gameTimer = 0;
    public int activateTime = 20;

    float downLimit = 0; //Ліміт опускання акули під воду (глибина)

    void Start(){}

    void FixedUpdate(){
        gameTimer += Time.deltaTime;

        // Після вказаної секунди активовується акула
        if (gameTimer > activateTime){
            if (moveBorder > 100){
                moveUp = false;
                downLimit = Random.Range(10f, -100f);
            }
            if (moveBorder < downLimit{
                moveUp = true;
            }

            if (moveUp == true{
                transform.position += transform.right / powerMove * Time.deltaTime;
                moveBorder++;
            }else{
                transform.position -= transform.right / powerMove * Time.deltaTime;
                moveBorder--;
            }
        }
    }
```

Рисунок 3.6 – SharkLogic

Цей клас `SharkLogic` відповідає за логіку руху акули в грі. Він містить логіку переміщення акули вздовж горизонтальної осі та управління її рухом вгору та вниз[21].

Цей клас має наступні поля та змінні:

- Змінна `powerMove` визначає швидкість руху акули.

- Змінна `moveBorder` вказує межу переміщення акули по горизонтально

- Змінна `moveUp` вказує, чи рухається акула вгору.

- Змінна `gameTimer` відстежує час, що пройшов після початку гри.

- Змінна `activateTime` визначає час, після якого акула стає активною.

- Змінна `downLimit` визначає ліміт, до якого акула може опуститися під воду (глибина).

Метод `Start()` викликається при запуску гри і не містить жодної логіки.

Метод `FixedUpdate()` викликається з фіксованою частотою і виконує головну логіку руху акули. Після того, як пройде визначений час `activateTime`, акула стає активною. Акула переміщується вздовж горизонтальної осі з використанням змінної `powerMove`. Акула також може рухатися вгору та вниз в залежності від значень `moveBorder` та `downLimit`. Якщо `moveBorder` перевищує 100, акула починає рухатися вниз і встановлюється випадкове значення для `downLimit`. Коли `moveBorder` стає меншим за `downLimit`, акула починає рухатися вгору. Рух акули вгору або вниз залежить від значення `moveUp`.

Клас **StoneLogic** відповідає за логіку небезпечних предметів які падуть з неба; Він містить логіку руху перешкод, зіткнення з об'єктами та їх знищення.

Цей клас має наступні поля та змінні:

- Змінна `anim` використовується для доступу до компонента анімації перешкод.

- Змінна `particles` зберігає посилання на частинки, які будуть використовуватись після зіткнення перешкоди із землею.

- Змінна `particleSystemUse` використовується для інстанціювання об'єкту частинок.

- Змінна `particle` містить посилання на компонент `ParticleSystem`, що керує частинками.

- Змінна `lifeTime` відстежує час, що пройшов після початку гри.

- Змінна `stoneSpeed` визначає швидкість руху перешкод.

- Змінна `stoneConnect` позначає, чи перешкода взаємодіє з іншими об'єктами.

Метод `Start()` викликається при запуску гри і ініціалізує деякі поля та компоненти. Якщо `boom` встановлено на `true`, створюються частинки і зберігається посилання на компонент `ParticleSystem`. В іншому випадку ініціалізується компонент анімації.

Змінна `stoneSpeed` встановлюється на значення 5f.

Метод `OnTriggerEnter(Collider collision)` викликається, коли перешкоди зіштовхуються з іншим об'єктом. Якщо зіштовхнення відбувається з "ground", то в залежності від значення `boom` вимикається колайдер перешкоди або включається відтворення частинок. Змінна `lifeTime` скидається на 0, а `stoneConnect` встановлюється в `true`. Якщо зіштовхнення відбувається з "destroyer", камінь знищується.

Метод `OnTriggerStay(Collider collision)` викликається, коли перешкода перебуває у взаємодії з іншим об'єктом. Якщо вона знаходиться на "ground" або зіштовхнувся з "Player", то його швидкість `stoneSpeed` зменшується.

Метод `Update()` викликається кожен кадр і відповідає за рух перешкод. Координата Y змінюється в залежності від `stoneSpeed` та часу, що пройшов з останнього кадру.

Метод `FixedUpdate()` викликається з фіксованою частотою і містить логіку знищення перешкод. Якщо гра закінчилася або перезапускається, перешкоди знищуються. Якщо `stoneConnect` встановлено в `true`, збільшується `lifeTime`. Якщо `lifeTime` досягло значення 30, камінь знищується.

Метод `OnDesttroy()` викликається перед знищенням об'єкту перешкод і знищує їх.

Клас **StoneGenerator** відповідає за генерацію небезпечних предметів що падають (Рис 3.7);

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

public class StoneGenerator : MonoBehaviour{
    public GameObject stone; // Префаб каменя
    int stoneNums; // Кількість каменів, які генеруються
    float generateTimer = 0; // Таймер для контролю інтервалу генерації
    float generateSpeed; // Швидкість генерації каменів
    public float minX = -25f; // Мінімальне значення X-координати для генерації каменів
    public float maxX = 25f; // Максимальне значення X-координати для генерації каменів
    public float angle = 0f; // Кут повороту каменів

    void Start(){
        generateSpeed = 3; // Чим більше значення, тим довше інтервал генерації
        stoneNums = 7; // Кількість каменів, які генеруються
    }

    void Update(){
        if (generateTimer >= generateSpeed)
            generateTimer = 0;

        if (generateTimer == 0){
            // Генерація каменів
            for (int i = 0; i < stoneNums; i++){
                // Встановлюємо випадкові координати для генерації каменя
                float randomX = transform.position.x + Random.Range(minX, maxX);
                float randomY = transform.position.y + Random.Range(-10f, 10f);
                Vector3 spawnPosition = new Vector3(randomX, randomY, transform.position.z);

                // Поворот каменя
                Quaternion spawnRotation = Quaternion.Euler(0, angle, 0);

                // Генерація каменя
                Instantiate(stone, spawnPosition, spawnRotation);
            }
        }
        generateTimer += Time.deltaTime;
    }
}
```

Рисунок 3.7 – StoneGenerator

Примітка:

Префаб (Prefab) - це тип об'єкта у Unity, який використовується для створення копій одного об'єкта з однаковими властивостями та компонентами. Префаб є шаблоном, який можна використовувати для створення багатьох однакових об'єктів у грі.

Префаб містить в собі всі необхідні компоненти, налаштування та параметри об'єкта, які можна використовувати при його створенні. Коли префаб розміщений у сцені, можна створювати екземпляри цього префаба, що дозволяє швидко створювати багато об'єктів з однаковими властивостями.

Одним з головних переваг використання префабів є можливість змінювати один префаб, а ці зміни автоматично відображатимуться на всіх створених екземплярах цього префаба у сцені. Це спрощує редагування та управління багатьма об'єктами одного типу[21].

Клас **Snowman** відповідає за фонових ворогів, які заважають гравцю;

Основні його елементи:

Animator anim: змінна для доступу до компонента Animator об'єкта

float time: час, який пройшов після початку гри

float timer: таймер для визначення інтервалу атаки ворога

bool animPlay: прапорець, що вказує, чи грається анімація атаки

int interval: інтервал між атаками

bool runRight: прапорець, що вказує, чи рухається ворог вправо

bool move: прапорець, що вказує, чи рухається ворог

float speed: швидкість руху ворога

int activateTime: час, після якого ворог активується

У методі Start() ініціалізується змінна anim і отримується доступ до компонента Animator об'єкта.

У методі OnTriggerEnter(Collider collision) встановлюються прапорці runRight в залежності від того, з якими об'єктами стикається ворог.

У методі Update() проводиться оновлення логіки ворога. Перевіряється час, що пройшов, та стан анімації. Залежно від цього рухається ворог вправо або вліво. Після досягнення певного часу активується атака ворога.

Клас **TrapLogic** відповідає за логіку пасток, які генеруються на локації (Рис 3.8);

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

public class TrapLogic : MonoBehaviour {
    Animation anim;

    void Start() {
        anim = GetComponent<Animation>();
    }

    // Викликається, коли об'єкт зіштовхується з іншим коллайдером
    void OnTriggerEnter(Collider collider) {
        if (collider.CompareTag("Player")) {
            anim.Play();
        }
    }

    // Оновлюється один раз на кадр
    void Update() {
        // Перевіряємо стани гри для знищення об'єкта
        if (GameData.gameOver || GameData.restartGame || GameData.exitGame) {
            Destroy(gameObject);
        }
    }
}
```

Рисунок 3.8 - TrapLogic

Коли об'єкт зіштовхується з колайдером, який має тег "Player", відтворюється анімація. Крім того, у кожному кадрі перевіряється стан гри (GameData.gameOver, GameData.restartGame, GameData.exitGame), і якщо хоча б одна з цих умов є істинною, об'єкт пастки буде знищено.

Папка **Bonus** містить дані про бонусні предмети:

Клас **BananaLogic** відповідає за логіку бонусів що падають;

Цей код відповідає за логіку бонусів у грі, зокрема за бонуси-банани. Ось розбір функцій:

1. У функції `Start` встановлюються початкові значення для анімації та швидкості руху банана.

2. Функція `OnTriggerEnter` викликається, коли об'єкт потрапляє в тригер. Якщо зіткнення відбулося з об'єктом з тегом "ground" або "Player", відтворюється анімація банана і встановлюється прапорець `bananaConnect` у значення `true`. У разі зіткнення з об'єктом з тегом "destroyer", банан знищується.

3. Функція `OnTriggerStay` викликається, якщо об'єкт перебуває в тригері. Якщо зіткнення відбувається з об'єктом з тегом "ground" або "Player", швидкість руху банана зменшується з кожним кадром.

4. У функції `Update` змінюється позиція банана вздовж осі Y залежно від швидкості руху. Якщо гра закінчилася або гравець перезапустив гру, банан знищується.

5. Функція `FixedUpdate` викликається один раз на фіксовану кількість кадрів. Якщо `bananaConnect` має значення `true`, збільшується лічильник `lifeTime`. Якщо `lifeTime` досягає значення 30, банан знищується.

6. У функції `OnDestroy` банан знищується при його знищенні.

Цей код реалізує рух та взаємодію бонусів-бананів у грі. Залежно від зіткнень з об'єктами з тегами "ground" та "Player", банан може відтворювати анімацію, змінювати швидкість та знищуватися.

Клас BananaGenerator відповідає за генерацію бонусів що падають;

Цей код відповідає за генерацію бонусів-бананів у грі. Ось розбір функцій:

1. У функції `Start` встановлюються початкові значення для швидкості генерації та кількості бананів.

2. У функції `Update` перевіряється час, що пройшов з останньої генерації бананів. Якщо `generateTimer` досягає значення `generateSpeed`, він скидається до нуля і генерується задана кількість бананів. Кожен банан розташовується у випадкових координатах на осі X та Y навколо об'єкта, до якого прикріплений цей скрипт (`transform.position`), з випадковими відхиленнями. Банани створюються за допомогою `Instantiate` з використанням префабу `banana` і створюються з початковим обертанням `Quaternion.identity`.

3. `generateTimer` збільшується на величину часу, що пройшов з останнього кадру за допомогою `Time.deltaTime`.

Цей код дозволяє генерувати задану кількість бонусів-бананів з заданою швидкістю у випадкових місцях навколо об'єкта, до якого прикріплений цей скрипт.

Клас BigBonusLogic відповідає за логіку великих бонусів на краях локацій;

2. У функції `OnTriggerEnter` перевіряється зіткнення з гравцем. Якщо бонус зіткнувся з гравцем, він знищується (`Destroy(gameObject)`). Якщо `NightLocation` дорівнює `true`, встановлюється інтенсивність світла (`NightLight`) на максимальне значення.

3. У функції `FixedUpdate` перевіряється поточний стан гри. Якщо гра перезапускається, бонус знищується (`Destroy(gameObject)`).

4. Змінні `powerMove`, `moveBorder` та `moveUp` використовуються для керування рухом бонусу. Бонус рухається вгору і вниз за допомогою трансформаційної матриці `transform`. Коли `moveBorder` досягає значення 10, `moveUp` стає `false`, що означає рух вниз. Коли `moveBorder` досягає значення 0, `moveUp` стає `true`, що означає рух вгору. Кожного кадру бонус обертається навколо осі Y за допомогою `Quaternion.AngleAxis` і переміщується вгору або вниз залежно від значення `moveUp` та `powerMove`.

5. У функції `OnDestroy` знищується бонус.

Цей код відповідає за рух та взаємодію великих бонусів у грі. Наприклад, при зіткненні з гравцем вони знищуються, а в разі використання на конкретній локації може змінюватися інтенсивність світла.

Клас BigBonusGenerator відповідає за генерацію великих бонусів;

1. У функції `Start` змінна `bonusActive` встановлюється на значення `false`.

2. У функції `Update` перевіряється стан генератора. Якщо `bonusActive` дорівнює `false`, а `activeGenerator` дорівнює `true`, то створюється екземпляр об'єкту `bonus` (великого бонусу) за допомогою `Instantiate`. Встановлюється положення новоствореного бонусу на позицію генератора. Якщо `NightLocation` дорівнює `true`, то на новоствореному бонусі встановлюється параметр `NightLocation` і `NightLight`. Змінні `bonusActive` та `activeGenerator` встановлюються на відповідні значення, а також змінні `bonusActive` та `activeGenerator` іншого генератора (`otherGenerator`) оновлюються.

Цей код відповідає за генерацію великих бонусів у грі. Він створює новий екземпляр великого бонусу на позиції генератора та налаштовує його параметри в залежності від умов.

Location відповідає за генерацію локації, обраної в списку локацій в меню (Рис 3.9);

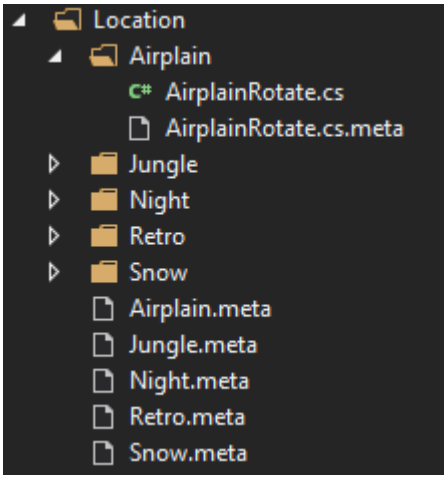


Рисунок 3.9 - Location

Цей код відповідає за генерацію перешкод (ящиків) у грі. Давайте розглянемо його детальніше (Рис 3.10):

- `float gameTimer = 0;` - це змінна, яка відстежує час гри.

- `bool createBox = false;` - це прапорець, який показує, чи потрібно створювати новий ящик.

- `public GameObject box;` - це префаб ящика, який буде генеруватися.

- `public int timeBoxGeneration = 30;` - це інтервал часу (у секундах), після якого буде генеруватися новий ящик.

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

public class JungleEvents : MonoBehaviour
{
    float gameTimer = 0; //вираховує час гри, використовується для ускладнення рівня з часом
    bool createBox = false;
    public GameObject box; //перешкода: ящик
    public int timeBoxGeneration = 30;

    void Start(){ }

    void GenerateFirstBox(){ //генерує першу постійну перешкоду: ящик
        if (createBox) return;
        if (timeBoxGeneration == 30) Instantiate(box, new Vector3(1, 20, 9), Quaternion.Euler(0, 140, 0));
        else if (timeBoxGeneration == 60) Instantiate(box, new Vector3(8, 20, 9), Quaternion.Euler(0, 140, 0));
        else if (timeBoxGeneration == 90)Instantiate(box, new Vector3(22, 20, 9), Quaternion.Euler(0, 140, 0));
        else Instantiate(box, new Vector3(15, 20, 9), Quaternion.Euler(0, 140, 0));
    }

    void FixedUpdate(){
        if (GameData.restartGame == true){
            gameTimer = 0;
            createBox = false;
        }

        gameTimer += Time.deltaTime;

        if (gameTimer > timeBoxGeneration){ // Генерація перешкоди, ящика, час вказано в секундах
            GenerateFirstBox();
            createBox = true;
        }

    }
}

```

Рисунок 3.10 - Location

У методі `GenerateFirstBox` генерується перша постійна перешкода (ящик) залежно від значення `timeBoxGeneration`. Використовуються різні позиції та повороти для ящиків.

Метод `FixedUpdate` викликається з фіксованою частотою і виконує логіку гри. Якщо гра перезапускається (`GameData.restartGame == true`), змінні

`gameTimer` та `createBox` скидаються до початкових значень. В іншому випадку, змінна `gameTimer` збільшується з кожним кадром.

Якщо `gameTimer` перевищує `timeBoxGeneration`, викликається метод `GenerateFirstBox`, який генерує новий ящик. Змінна `createBox` встановлюється в `true`, щоб уникнути множинну генерацію ящиків до наступного інтервалу.

Цей код використовується для створення системи генерації перешкод у грі, забезпечуючи ускладнення рівня з часом.

GameData реалізує збереження та загрузку даних з пристрою

(Рис 3.11);

```
51      public static int smallBonusCoinsBase = 10;
52      public static int bigBonusCoinsBase = 100; // Базове значення монет за бонуси (малий та великий)
53      public static int smallBonusScoreBase = 10;
54      public static int bigBonusScoreBase = 100; // Базове значення балів за бонуси (малий та великий)
55
56      public static int smallBonusCoinsCost = 1000;
57      public static int bigBonusCoinsCost = 1000; // Базова ціна монет за бонуси (малий та великий)
58      public static int smallBonusScoreCost = 1000;
59      public static int bigBonusScoreCost = 1000; // Базова ціна балів за бонуси (малий та великий)
60
61      public static int smallBonusCoinsMultiplier = 2000;
62      public static int bigBonusCoinsMultiplier = 2000; // Множитель ціни монет за бонуси (малий та великий)
63      public static int smallBonusScoreMultiplier = 2000;
64      public static int bigBonusScoreMultiplier = 2000; // Множитель ціни балів за бонуси (малий та великий)
65
66      public static int smallBonusCoinsLevel = 0;
67      public static int bigBonusCoinsLevel = 0; // Рівень додаткових монет за бонуси (малий та великий)
68      public static int smallBonusScoreLevel = 0;
69      public static int bigBonusScoreLevel = 0; // Рівень додаткових балів за бонуси (малий та великий)
70
71      public static int[] smallBonusCoinsUp = { 0, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 3, 3, 3, 3, 3, 5, 5, 5, 5, 5, 10, 1
72      public static int[] bigBonusCoinsUp = { 0, 2, 2, 2, 2, 2, 5, 5, 5, 5, 5, 10, 10, 10, 10, 10, 15, 15, 15, 15, 1
73      // Збільшення монет на певне значення в залежності від рівня
74      public static int[] smallBonusScoreUp = { 0, 1, 1, 1, 1, 1, 5, 5, 5, 5, 5, 10, 10, 10, 10, 10, 15, 15, 15, 15,
75      public static int[] bigBonusScoreUp = { 0, 5, 5, 5, 5, 5, 10, 10, 10, 10, 10, 20, 20, 20, 20, 20, 30, 30, 30,
76      // Збільшення балів на певне значення в залежності від рівня
77
78      public static bool reloadAd = false;
79      public static float ADStime = 0; //значення в секундах (600 секунд - 10 хвилин). 10 хв. гри = 1 обов'язкова ре
80      public static float FreeCoins = 300; //Час до отримання безкоштовних монет
81
82      //Містить ціни локацій та персонажів
83      public static int[] locsCost = { 0, 10000, 35000, 70000, 125000, 100000, 250000};
84      public static int[] charsCost = { 0, 10000, 50000, 30000, 60000, 150000, 175000, 275000, 300000};
85
86      public static bool[] locsActive = new bool[locsNum]; //Перевіряє чи доступна локація або персонаж (чи куплена)
87      public static bool[] charsActive = new bool[charsNum];
88
89      public static int[] scoreLocation = new int[locsNum]; //Містить рекорди на конкретних локаціях або персонажах
90      public static int[] scoreCharacter = new int[charsNum];
```

Рисунок 3.11 - GameData

Цей код містить різноманітні дані та налаштування для гри.

Наприклад тут зберігаються дані про загальну кількість монет та найкращий результат гри, ідентифікатори поточної локації та персонажа у грі,

кількість монет, які гравець заробив на поточному рівні, ціни для локацій та персонажів у грі, чи доступні локації та персонажі (чи вони куплені), рекорди гравця для кожної локації та персонажа, чи ввімкнені звук та музика у грі.

Також визначаються різні параметри та налаштування, такі як кількість рівнів та базові значення для бонусів, ціни, мультиплікатори, рівні та підвищення для монет та балів.

Клас **LocCharList** відповідає за реалізацію списків персонажів та локацій (Рис 3.12);

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

public class LocCharListPosition : MonoBehaviour {
    public int ident = 0; // 0-locs, 1-chars
    Vector3 coord;
    RectTransform cont;

    void Start() {
        cont = GetComponent<RectTransform>();

        // Встановлюємо початкову позицію залежно від значення ident
        if (ident == 0) {
            coord = cont.position;
            coord.y = GameData.coordLocs;
            cont.anchoredPosition = coord;
        }
        else if (ident == 1) {
            coord = cont.position;
            coord.y = GameData.coordChars;
            cont.anchoredPosition = coord;
        }
    }

    void Update() {
        // Оновлюємо координати в залежності від значення ident
        if (ident == 0) {
            GameData.coordLocs = cont.anchoredPosition.y;
        }

        else if (ident == 1) {
            GameData.coordChars = cont.anchoredPosition.y;
        }
    }
}
```

Рисунок 3.12 – LocCharList

Цей скрипт відповідає за позицію списку розташувань (locs) або персонажів (chars) в грі. Залежно від значення ident, встановлюється початкова позиція списку. При оновленні кожного кадру, координати позиції зберігаються в змінній GameData.

Клас **StartGame** відповідає за перевірку обраного персонажа та локації і розпочинає гру (генерує локацію та персонажа) (Рис 3.13);

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

public class StartGame : MonoBehaviour
{
    // Старт викликається перед першим оновленням кадру
    void Start(){}

    public void startGame(){
        SceneManager.LoadScene("Jungle");
    }

    //Оновлення викликається один раз за кадр
    void Update(){}
}
```

Рисунок 3.13 - StartGame

Клас **UpgradeStates**;

У цьому скрипті оновлюються і відображаються дані про покращення. ConvertCost перетворює вартість покращення на текстовий формат зі зрозумілими позначеннями (наприклад, 1K для 1000). ButtonDisplay відповідає за відображення кнопок покращення залежно від рівня покращення і наявності коштів. DataDisplay оновлює дані про покращення (інформацію, рівень, наступний рівень) залежно від наявних даних гри. StartGame завантажує сцену при початку гри[23].

Клас **GameShop** відповідає за реалізацію ігрового магазину.

1. Скрипт визначає масиви для елементів інтерфейсу користувача, таких як `locsCostUI` і `charsCostUI`, які використовуються для відображення вартості локацій та персонажів у гральному магазині.

2. Також є масиви для ігрових об'єктів, таких як `locs` і `chars`, які представляють кнопки для покупки локацій та персонажів.

3. Функція `ConvertCost` перетворює вартість товару у зрозумілий формат, наприклад, "1K" для 1000 або "1M" для 1000000.

4. Функція `buttonDisplay` відповідає за відображення кнопок для покупки локацій та персонажів на основі прогресу гравця та наявних монет. Якщо локація або персонаж вже активні, кнопка для них приховується. Якщо у гравця є достатньо монет, кнопка стає активною, в іншому випадку вона вимкнена та відображається з сірим кольором.

5. Функція `LocCharBuy` здається заповнювачем і наразі порожня.

6. У функції `Start` початкові вартості локацій та персонажів перетворюються у зрозумілий формат і відображаються на інтерфейсі користувача.

7. Функція `Update` викликається один раз на кадр і оновлює відображення кнопок на основі прогресу гравця та наявних монет. Вона також викликає порожню функцію `LocCharBuy`.

Загалом, цей скрипт керує інтерфейсом грального магазину, включаючи відображення вартостей, увімкнення/вимкнення кнопок та обробку покупок локацій та персонажів гравцем.

3.2 Вибір і налаштування музики та звукового супроводу

Для створення музики я обрав FL Studio, його переваги:

1. Професійні можливості: FL Studio є повноцінною цифровою аудіостанцією, яка надає широкий набір інструментів і функцій для створення

музики. Вона має синтезатори, семплери, ефекти та засоби мікшування, що дозволяють створювати різноманітні музичні композиції.

2. Інтуїтивний інтерфейс: FL Studio має дружній та інтуїтивно зрозумілий інтерфейс користувача, що спрощує процес створення музики навіть для початківців. Зручні робочі області, панелі інструментів та перетягування-покласти функції дозволяють швидко налаштовувати та редагувати музичні компоненти.

3. FL Studio надає засоби для роботи з семплами звуків, включаючи можливість скидання, зациклювання, зміни темпу та тону, а також різні ефекти обробки звуку. Це дозволяє легко створювати унікальні звукові та ритмічні шари для музики.

4. Велика спільнота користувачів: FL Studio має велику спільноту користувачів, що означає, що доступна велика кількість ресурсів, уроків, плагінів та зразків, які можна використовувати для поліпшення своєї музики. Це створює можливості для навчання, обміну досвідом та отримання підтримки в процесі творчості.

Загалом, FL Studio є інструментом для створення музики, який поєднує в собі простоту використання, багатий функціонал і велику спільноту користувачів[17,18].

3.3 Дизайн та інтеграція графіки та анімації

Blender є універсальним інструментом для створення 3D моделей і анімацій. Ось кілька причин, які обґрунтовують вибір Blender для проекту:

1. Безкоштовність: Blender є вільним програмним забезпеченням, доступним безкоштовно. Це робить його дуже привабливим для незалежних розробників, студентів та інших людей з обмеженим бюджетом. Використовуючи Blender, можна значно зменшити витрати на програмне забезпечення.

2. Моделювання: Blender має потужні інструменти для створення 3D моделей. Він підтримує різні техніки моделювання, включаючи полігональне, субдивізіонне, скелетне і NURBS моделювання. З його допомогою можна створювати складні і деталізовані моделі, від персонажів до об'єктів оточення.

3. Анімація: Blender також має вбудовані інструменти для створення анімацій. Він підтримує ключову анімацію, скелетну анімацію, інверсну кінематику та багато іншого. З допомогою Blender можна створювати реалістичні рухи персонажів, ефекти анімації середовища та багато іншого.

4. Візуалізація: Blender надає потужні можливості візуалізації, які допомагають реалістично відтворити матеріали, освітлення та тіні. З його допомогою можна створювати красиві і привабливі візуальні сцени, що додають глибину і насиченість до гри.

5. Спільнота та ресурси: Blender має велику та активну спільноту користувачів і розробників. Існує багато онлайн-ресурсів, таких як посібники, відеоуроки і форуми, де можна отримати допомогу, поради та підтримку в процесі роботи з Blender. Це дозволяє швидко розвиватися і покращувати навички використання цього інструменту[19].

3.4 Висновок розділ 3

У третьому розділі дипломної роботи "Розробка мобільної гри у жанрі Endless runner з елементами platformer та arcade на Unity з використанням мови програмування C#" було проведено розробку додатку з використанням різних технологій та інструментів.

Для розробки гри було використано Unity, яке є потужним движком розробки ігор і надає широкі можливості для створення геймплею, графіки та звуків. Мова програмування C# була обрана для реалізації логіки гри, оскільки вона є основною мовою програмування, що підтримується Unity.

Під час розробки було створено головного героя гри з елементами platformer та arcade. Було реалізовано рух персонажа, його взаємодію з оточуючим середовищем та перешкодами. Також було розроблено систему очків та життів героя, а також можливість збирання бонусів та покращень.

Додаток отримав графічне оформлення, включаючи стилізовані фони, персонажів, об'єкти та ефекти. Звуковий супровід був також реалізований для створення належної атмосфери гри.

У результаті розробки було створено мобільну гру у жанрі Endless runner з елементами platformer та arcade, що пропонує гравцеві захоплюючий геймплей, високу якість графіки та звукового супроводу. Використання Unity та мови програмування C# дозволило забезпечити швидкий та ефективний процес розробки гри.

Висновок з даного розділу полягає в тому, що розробка мобільної гри у жанрі Endless runner з елементами platformer та arcade на Unity з використанням мови програмування C# є вдалим вибором для створення захоплюючих та якісних ігрових додатків.

ВИСНОВОК

В ході виконання курсової роботи був отриманий повнофункціональний веб-сайт, повністю готовий до застосування. Даний сайт орієнтований для широкого спектру застосування в on-line інформації. З його допомогою користувачі зможуть отримувати необхідну інформацію про газету, а також про події, які відбуваються у сучасному світі. При розміщенні його в глобальній мережі географія розповсюдження зростає до масштабів всього світу.

В результаті розробки мобільної гри у жанрі Endless runner з елементами platformer та arcade на Unity з використанням мови програмування C#, було здійснено наступні кроки:

1. Була розроблена концепція гри, визначений її жанр і головні характеристики. Гра поєднує в собі елементи Endless runner, де гравець змушений довго уникати перешкод, з платформером, де є необхідність переміщатись по платформам, та аркадою, що передбачає швидкий та динамічний геймплей.

2. Було проведено огляд досліджень у галузі розробки мобільних ігор, зокрема у жанрі Endless runner. Було проаналізовано особливості цього жанру, розглянуто популярні ігри та їхні особливості.

3. Було оглянуто особливості розробки мобільних ігор на Unity з використанням мови програмування C#. Розглянуто різні аспекти розробки, такі як проектування інтерфейсу та геймплею, вибір і налаштування музики та звукового супроводу, розробка структури проекту та налаштування робочого середовища Unity.

4. Була реалізована механіка гри на мові програмування C#. Було забезпечено генерацію елементів гри, таких як бонуси та небезпечні предмети, переміщення персонажа та його взаємодію з ігровим світом. Також було реалізовано збереження ігрової валюти та рахунку на пристрої.

5. Було розроблено управління та обробку взаємодії з користувачем. Було створено інтерфейс користувача для керування грою, оброблено події взаємодії користувача, такі як натискання кнопок, жести та дотики екрану.

Отже, розробка мобільної гри у жанрі Endless runner з елементами platformer та arcade на Unity з використанням мови програмування C# є актуальною та цікавою задачею. Результати дослідження та розробки можуть бути використані в мобільній ігровій індустрії для створення захопливих та динамічних ігрових проектів.

ПЕРЕЛІК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Brathwaite, B., & Schreiber, I. (2009). Challenges for Game Designers. Charles River Media.

2. Unity Documentation: https://docs.unity3d.com/ - Офіційна документація Unity, яка надає докладну інформацію про розробку ігор на Unity.

3. C# Programming Yellow Book by Rob Miles: http://www.csharpcourse.com/ - Книга, що надає практичне введення в мову програмування C#.

4. Unity Asset Store: https://assetstore.unity.com/ - Офіційний магазин Unity, де можна знайти різноманітні ресурси, такі як готові 2D та 3D арти, звукові ефекти, скрипти та інші активи, що полегшують розробку ігор.

5. Gamasutra: https://www.gamasutra.com/ - Онлайн-портал, присвячений ігровій індустрії, де можна знайти статті, новини та інтерв'ю з розробниками ігор.

6. Walker, H. (2017). The Beginner's Guide to Android Game Development. Packt Publishing Ltd.

7. Harrison, J. (2018). Mastering Unity 2D Game Development. Packt Publishing Ltd.

8. Unity in Action: Multiplatform Game Development in C# by Joe Hocking. Manning Publications.

9. Rollings, A., & Morris, D. (2004). Game Architecture and Design: A New Edition. New Riders.

10. Cook, P., & Bjork, S. (2019). Game Design Workshop: A Playcentric Approach to Creating Innovative Games. CRC Press.

11. Juul, J. (2013). The Art of Failure: An Essay on the Pain of Playing Video Games. MIT Press.

12. Schell, J. (2014). The Art of Game Design: A Book of Lenses. CRC Press.

13. Moore, M. (2018). Level Up! The Guide to Great Video Game Design. Wiley.

14. Game Programming Patterns by Robert Nystrom: http://gameprogrammingpatterns.com/ - Книга, що описує різноманітні патерни програмування для розробки ігор.

15. Unity Learn: https://learn.unity.com/ - Офіційна платформа Unity для навчання розробки ігор, яка надає доступ до навчальних матеріалів, відеоуроків та проектів.

16. Extra Credits YouTube Channel: https://www.youtube.com/user/ExtraCreditz - YouTube-канал, де розглядаються різні аспекти геймдизайну та ігрової індустрії.

17. Game Developer Magazine: https://www.gdmag.com/ - Журнал для професіоналів геймдеву, що містить статті, інтерв'ю, новини та інші матеріали.

18. GameDev.net: https://www.gamedev.net/ - Веб-сайт, присвячений розробці ігор, з форумами, статтями, туторіалами та іншими ресурсами.

19. GameAnalytics Blog: https://www.gameanalytics.com/blog/ - Блог, що містить статті та практичні поради щодо аналітики та монетизації ігор.

20. GameDevHQ: https://gamedevhq.com/ - Онлайн-платформа для навчання розробки ігор, що надає доступ до курсів, відеоуроків та ресурсів.

21. Udemy: https://www.udemy.com/ - Онлайн-платформа для навчання, на якій можна знайти курси з розробки ігор на різних платформах.

22. Coursera: https://www.coursera.org/ - Онлайн-платформа, яка співпрацює з університетами та організаціями, щоб надати доступ до курсів з різних тем, включаючи розробку ігор.

23. Stack Overflow: https://stackoverflow.com/ - Онлайн-форум для програмістів, де можна знайти відповіді на технічні питання та проблеми, пов'язані з розробкою ігор.

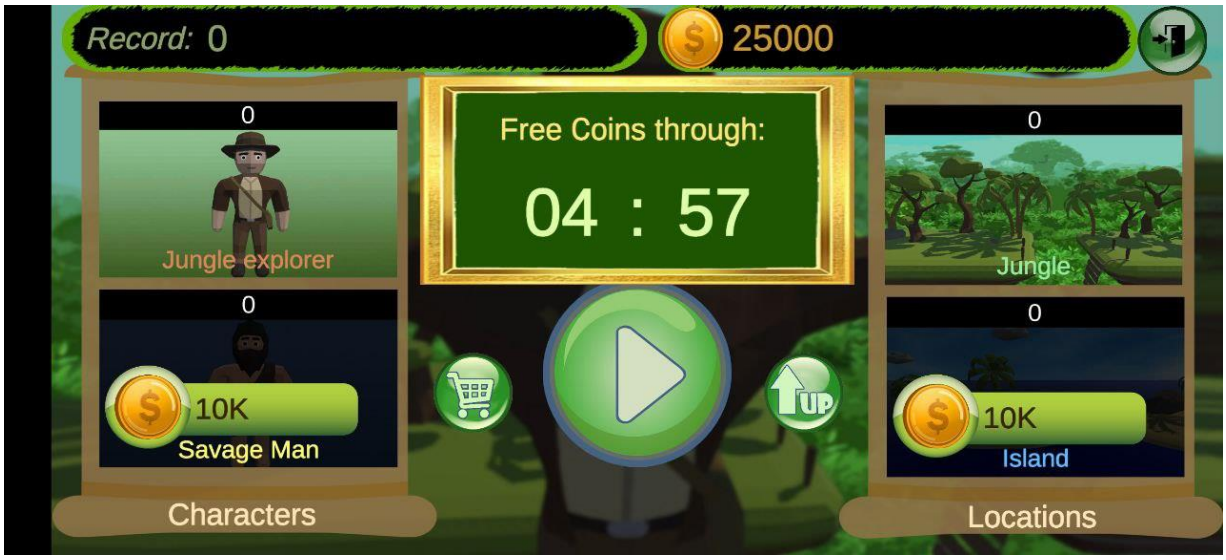
24. YouTube: https://www.youtube.com/ - Популярна платформа з великою кількістю відеоуроків та каналів, присвячених розробці ігор.

25. GitHub: https://github.com/ - Веб-платформа для спільної роботи над проектами розробки програмного забезпечення, включаючи відкриті проекти з розробки ігор.

ДОДАТКИ

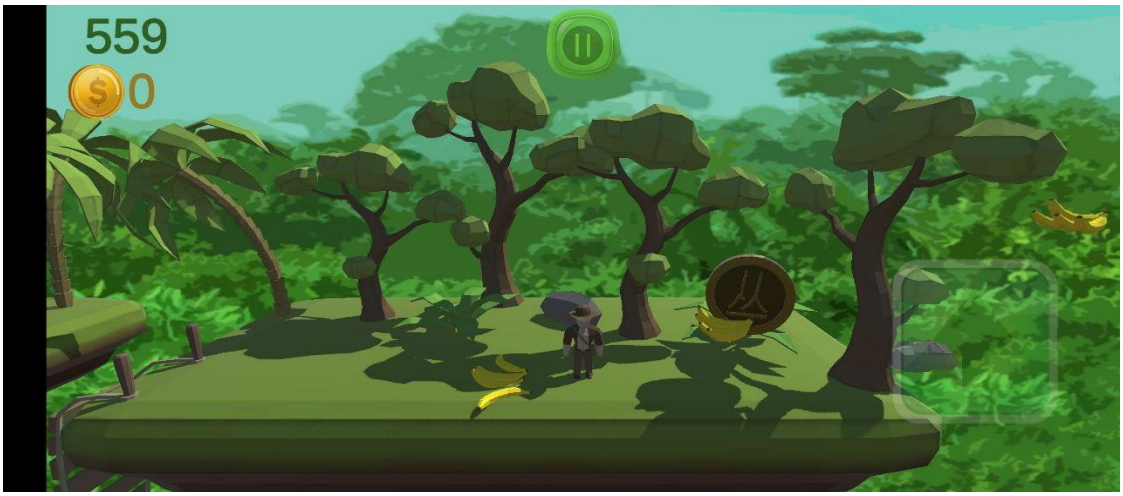
Додаток А

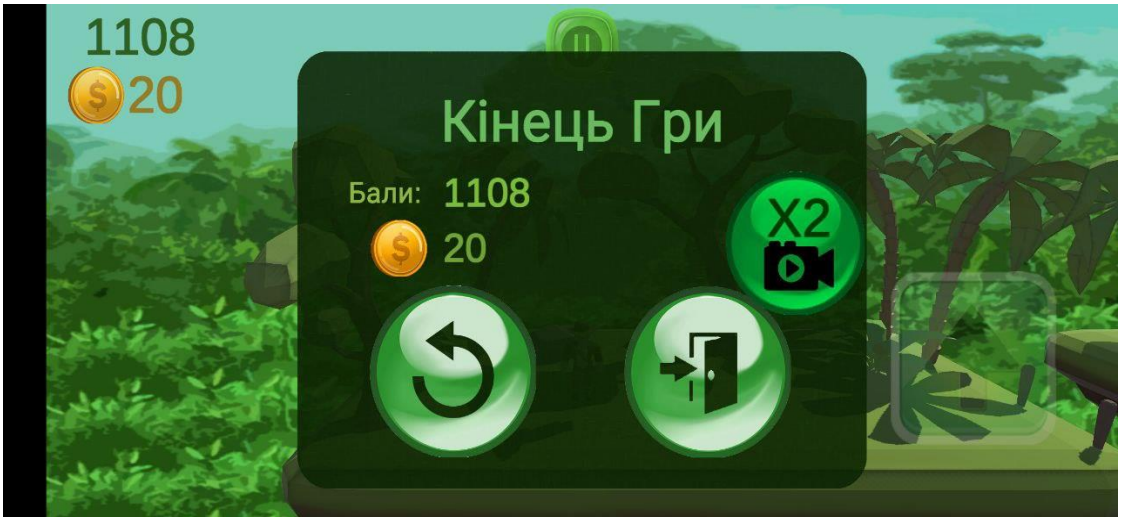
Результати роботи




Налаштування
Мова
Музика
Звуки
Зберегти


242
0


559
0


1108
20
Кінець Гри
Бали: 1108
20
X2


25020
Покращення
Рівень 0
Монети + 1
1K
Монети за 1 бонус: 10
Рівень 0
Монети + 2
1K
Монети за 1 великий бонус: 100
Рівень 0
Бали + 1
1K