Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизації
Кафедра комп'ютерних наук

Бакалаврська дипломна робота на тему:

КОМП'ЮТЕРНА ГРА ЖАНРУ "RUNNER" З ВНУТРІШНЬОІГРОВИМ МАГАЗИНОМ

Виконав: студент 4 курсу, групи 1КН–20б
спеціальності 122 – Комп'ютерні науки
(шифр і назва напряму підготовки, спеціальності)
Макогончук Д.А.
(прізвище та ініціали)

Керівник: д.т.н., проф. каф. КН
Іванчук Я. В.
(прізвище та ініціали)
«07» 06 2024 р.

Рецензент: д.т.н., проф., зав. каф. КСУ
Ковтун В. В.
(прізвище та ініціали)
«07» 06 2024 р.

Допущено до захисту
Зав. кафедри проф., д.т.н. Яровий А.А.
«07» 06 2024 р.

Вінниця ВНТУ – 2024 рік

Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизації
Кафедра комп'ютерних наук
Рівень вищої освіти перший (бакалаврський)
Галузь знань – 12 Інформаційні технології
Спеціальність – 122 Комп'ютерні науки
Освітньо–професійна програма – Комп'ютерні науки

ЗАТВЕРДЖУЮ
Завідувач кафедри КН
проф., д.т.н. Яровий А.А.
«07» 03 2024р.

З А В Д А Н Н Я
НА БАКАЛАВРСЬКУ ДИПЛОМНУ РОБОТУ СТУДЕНТУ
Макогончуку Дмитру Андрійовичу

1. Тема роботи: Комп'ютерна гра жанру "runner" з внутрішньоігровим магазином
Керівник роботи: д.т.н., професор Іванчук Я. В. затверджені наказом вищого навчального закладу "11" 03 2024 року №80
2. Строк подання студентом роботи 27 05 2024р.
3. Вихідні дані до роботи: кількість гравців – 1 чол., параметри роздільної здатності дисплею – 1920x1080 пкс., частота кадрів – 60 кадр./сек., мова програмування – C#, Unity Engine.
4. Зміст текстової частини (перелік питань, які потрібно розробити): аналіз предметної області, аналіз існуючих концепцій ігор жанру "runner", аналіз використання властивості кроссплатформенності комп'ютерних ігор, розробка і проєктування програмного модуля, проєктування інтерфейсу користувача, проєктування ігрового циклу, програмна реалізація комп'ютерної гри жанру "runner" з внутрішньоігровим магазином, розробка графічних матеріалів, розробка користувацького інтерфейсу, розробка основних модулів і тестування додатку комп'ютерної гри.
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень): схеми користувацького інтерфейсу, схема алгоритму роботи програми, функціональна модель персонажа, модель реалізації ігрової системи, діаграма ігрового циклу, загальний вигляд інтерфейсного вікна середовища Piskel, загальний вигляд вікна середовища Unity, зображення ігрового процесу.

6. Консультанти розділів проекту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завда прий
1–3	Іванчук Я. В., д.т.н., проф.. каф. КН		

7. Дата видачі завдання 07.03.2024р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва та зміст етапу	Термін виконання		П
		Початок	Закінчення	
1	Аналіз предметної області, існуючих методів, алгоритмів та засобів для комп'ютерних ігор жанру "runner"	06.05.2024	09.05.2024	
2	Проектування програмного модулю комп'ютерної ігор жанру "runner"	10.05.2024	14.05.2024	
3	Програмна реалізація модулю системи комп'ютерної ігор жанру "runner"	15.05.2024	20.05.2024	
4	Розробка інструкції користувача.	21.05.2024	24.05.2024	
5	Оформлення пояснювальної записки	25.05.2024	28.05.2024	
6	Оформлення матеріалів до захисту БКР	29.05.2024	06.06.2024	

Студент _____ Макогончук Д. А.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Іванчук Я. В.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Бакалаврська дипломна робота складається з 89 сторінок формату А4, на яких є 30 рисунків, список використаних джерел містить 16 найменувань.

Дана бакалаврська робота присвячена розробці та впровадженню комп'ютерної гри жанру раннер з внутрішньоігровим магазином. Основною метою роботи є створення захопливої та функціональної гри, яка включає можливість здійснення покупок всередині гри. Була розроблена структурна схема для опису функціонування гри, схема алгоритму роботи основних механік, UML–діаграма алгоритму організації внутрішньоігрового магазину. Unity та C# використовуються для розробки гри та забезпечення взаємодії з користувачем. Архітектура гри реалізується за підходом, що дозволяє забезпечити розділення логіки, представлення та обробки даних.

роботи є захоплива і надійна гра, яка дозволяє гравцям ефективно здійснювати покупки всередині гри, тим самим покращуючи їхній ігровий досвід.

Ключові слова: комп'ютерна гра, раннер, внутрішньоігровий магазин, алгоритм, Unity, C#.

ABSTRACT

The bachelor's thesis consists of 89 A4 pages, including 30 illustrations, and the list of references contains 16 sources.

This bachelor's thesis is dedicated to the development and implementation of a “runner” game with an in–game store. The main objective of the thesis is to create an engaging and functional game that includes the capability for in–game purchases. A structural diagram was developed to describe the game's functionality, along with an algorithm schema for the main mechanics and a UML diagram for the organization of the in–game store. Unity and C# are used for game development and user interaction. The game architecture is implemented with an approach that ensures the separation of logic, presentation, and data processing.

The result of the work is an engaging and reliable game that allows players to make in–game purchases effectively, enhancing their gaming experience.

Keywords: computer game, “runner”, in–game store, algorithm, Unity, C#.

ЗМІСТ

ВСТУП 7
1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ 10
1.1 Аналіз існуючих концепцій ігор жанру “runner” 10
1.2 Аналіз використання властивості кроссплатформенності комп’ютерних ігор 17
1.3 Аналіз відомих комп’ютерних ігор жанру “runner” 23
1.4 Висновок до розділу 1 25
2 РОЗРОБКА І ПРОЄКТУВАННЯ ПРОГРАМНОГО МОДУЛЯ КОМП’ЮТЕРНОЇ ГРИ 26
2.1 Проєктування інтерфейсу користувача 26
2.2 Проєктування моделі реалізації ігрової системи 29
2.3 Проєктування алгоритму роботи програми 35
2.4 Висновок до розділу 2 41
3 ПРОГРАМНА РЕАЛІЗАЦІЯ КОМП’ЮТЕРНОЇ ГРИ 42
3.1 Аналіз і обґрунтування вибору мов програмування 42
3.2 Програмна розробка основних модулів додатку комп’ютерної гри 45
3.3 Розробка графічних матеріалів 52
3.4 Результати тестування програмного модуля комп’ютерної гри 57
3.5 Висновки до розділу 3 67
ВИСНОВКИ 69
СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ 70
ДОДАТОК А Протокол перевірки кваліфікаційної роботи на наявність текстових запозичень 72
ДОДАТОК Б Інструкція користувача 73
ДОДАТОК В Лістинг програми 75
ДОДАТОК Г Графічна частина 85

ВСТУП

Актуальність даної дипломної роботи зумовлена стрімким розвитком індустрії комп'ютерних ігор, зокрема жанру “runner”, який є популярним серед мобільних гравців. Ігрова індустрія є однією з найдинамічніших та найприбутковіших галузей сучасної цифрової економіки. З розвитком мобільних технологій та широким доступом до інтернету, ігри стали невід'ємною частиною життя багатьох людей. Серед різноманітних жанрів комп'ютерних ігор особливе місце займають ігри жанру “runner”, які вирізняються своєю простотою управління, динамічним геймплеєм та високою [1].

Дослідження ефективних підходів до інтеграції внутрішньоігрових магазинів є важливим для покращення монетизації та користувацького досвіду. Зростання потужності мобільних пристроїв і значення кроссплатформенності вимагають оптимізації ігор для різних платформ. Отримані результати сприятимуть розробці якісних та комерційно успішних ігор, що задовольнять потреби сучасних гравців і підтримають економічне зростання галузі.

Дана робота включає кілька ключових етапів:

1. Аналіз існуючих концепцій та жанрів відеоігор – визначення основних характеристик та механік, які роблять ігри успішними.

2. Аналіз кроссплатформенності відеоігор – дослідження важливості та способів забезпечення підтримки різних платформ.

3. Розробка власної кросплатформенної гри – розробити гру у жанрі “runner”.

На основі проведених досліджень буде розроблено концепцію гри, яка поєднує традиційні елементи жанру “runner” з ефективно реалізованим внутрішньоігровим магазином. Це дозволить не лише забезпечити захоплюючий ігровий процес, але й створити стійку бізнес–модель для розробників [1].

Також передбачається аналіз існуючих концепцій та видів “runner”–ігор, дослідження аспектів кроссплатформенності та вивчення ефективних підходів до інтеграції внутрішньоігрових покупок. У процесі дослідження буде розглянуто, як внутрішньоігровий магазин може вплинути на геймплей, залученість гравців та монетизацію гри [1].

Очікується, що результати цієї роботи зроблять внесок у розуміння ефективних підходів до розробки та монетизації ігор жанру “runner”, а також допоможуть розробникам створювати більш якісні та комерційно успішні ігрові продукти.

Метою роботи є реалізація дозвілля людини на основі розробленої комп’ютерної гри жанру “runner” з внутрішньоігровим магазином, яка дозволяє нормалізувати психоемоційний стан і підвищити інтелектуальний рівень людини.

Задачі дослідження:

1. Виконати аналіз предметної області.
2. Провести аналіз існуючих комп’ютерних ігор жанру “runner” та можливості використання кроссплатформенності в комп’ютерних іграх.
3. Розробити загальну структурну схему функціонування програмного модуля комп’ютерної гри у жанрі “runner” з внутрішньоігровим магазином.
4. Розробити модуль інтерфейсу комп’ютерної гри у жанрі “runner” з внутрішньоігровим магазином.
5. Реалізація програмного модуля комп’ютерної гри у жанрі “runner” з внутрішньоігровим магазином.
6. Тестування програмного модуля комп’ютерної гри у жанрів “runner” з внутрішньоігровим магазином.
7. Розробка інструкції користувача.

Об’єктом дослідження є процес розробки комп’ютерної гри у жанрі “runner” з внутрішньоігровим магазином.

Предметом дослідження є алгоритми та програмні засоби комп’ютерної гри у жанрі “runner” з внутрішньоігровим магазином.

Апробація роботи. Результати дослідження були апробовані на всеукраїнській науково-технічній конференції факультету інтелектуальних інформаційних технологій та автоматизації [2].

Публікації: електронні тези науково-технічної конференції «Всеукраїнська науково-технічна конференція факультету інтелектуальних інформаційних технологій та автоматизації» [2].

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Аналіз існуючих концепцій ігор жанру “runner”

Комп'ютерні ігри на сьогоднішній день – це величезна та різноманітна індустрія, яка поєднує в собі технологічний прогрес, культурні аспекти, розваги та спілкування. Кілька ключових аспектів предметної області комп’ютерних ігор [3]:

1. Технологічний прогрес: З роками графіка, звук та фізика в комп'ютерних іграх вдосконалюються. Від кольорових пікселів до фотореалістичних візуальних ефектів, від простих мелодій до об'ємного звуку – технології використовуються для створення більш іммерсивного ігрового досвіду.
2. Жанри ігор: Існує безліч жанрів комп'ютерних ігор, від стратегій та шутерів до ігор у жанрі рольових ігор (RPG), пригодницьких ігор та ігор імітації життя. Кожен жанр має свою унікальну механіку та стиль геймплею.
3. Індустрія та економіка: Комп'ютерні ігри стали однією з найбільших розважальних галузей світу. Вони генерують величезні прибутки через продажі ігор, мікротранзакції, додатковий контент та інші послуги.
4. Онлайн та мультиплеєрність: З поширенням Інтернету комп'ютерні ігри стали більш соціальними. Гравці можуть спілкуватися, співпрацювати або змагатися один з одним у реальному часі, незалежно від того, де вони знаходяться.
5. Культурний вплив: Комп'ютерні ігри впливають на культуру та суспільство. Вони стали об'єктом вивчення для академічних досліджень, вони інспірують кіно, музику та інші форми мистецтва, а також впливають на спосіб, яким ми ділимося та спілкуємося в цифровому світі.
6. Виклики та проблеми: Як і будь–яка інша індустрія, комп'ютерні ігри мають свої виклики, такі як проблеми з поведінковою залежністю, контентом, який може бути нецензурним або насильницьким, а також проблеми з безпекою даних та

приватністю.

Загалом, комп'ютерні ігри стали невід'ємною частиною сучасної культури та відіграють важливу роль у розвагах, соціальному спілкуванні та навіть навчанні.
У концепції типу "Біг за безкінечністю" (Endless runner) гравець керує персонажем, який безперервно біжить вперед, спробуючи пройти якомога далі, уникаючи перешкод та збираючи бонуси. Гра не має кінцевого рівня, але стає складнішою з часом .

Прикладом даного типу комп’ютерних ігор на рисунку 1.1, 1.2: “Temple Run”, “Jetpack Joyride” [3].

Рисунок 1.1 – Загальний вигляд ігрового процесу комп’ютерної гри “Temple Run 2” концепції типу “Біг за безкінечністю” (Endless runner)

Рисунок 1.2 – Загальний вигляд ігрового процесу комп’ютерної гри “Jetpack Joyride” концепції типу “Біг за безкінечністю” (Endless runner)

Ця концепція підходить для швидкого та динамічного геймплею, де гравцеві доводиться реагувати на обставини на ходу.

У концепції типу "Інтерактивність середовища" (Environment Interaction) гравець керує персонажем, який біжить через середовище, що постійно змінюється. Гравець може взаємодіяти з елементами оточуючого світу, наприклад, пересувати платформи, активувати механізми тощо.

Прикладом даного типу комп’ютерних ігор на рисунках 1.3, 1.4, 1.5: "Rayman Jungle Run", "Canabalt", "Bit.Trip runner" [3].

Рисунок 1.3 – Загальний вигляд ігрового процесу комп’ютерної гри “Rayman Jungle Run” концепції типу "Інтерактивність середовища" (Environment Interaction)

Рисунок 1.4 – Загальний вигляд ігрового процесу комп’ютерної гри “Canabalt” концепції типу "Інтерактивність середовища" (Environment Interaction)

Рисунок 1.5 – Загальний вигляд ігрового процесу комп’ютерної гри “Bit.Trip runner” концепції типу "Інтерактивність середовища" (Environment Interaction)

Ця концепція додає глибину до геймплею, оскільки гравець повинен не лише швидко реагувати, а й використовувати стратегічне мислення для взаємодії з оточенням.

Концепція "Стратегічне керування" (Strategic Management):

У таких іграх гравець керує не тільки персонажем, який біжить, але і вправляється у керуванні різними аспектами гри, такими як розвиток бази, керування ресурсами або збір команди.

Прикладом даного типу комп’ютерних ігор на рисунку 1.6, 1.7: “Zombie Tsunami”, “Blades of Brim” [3].

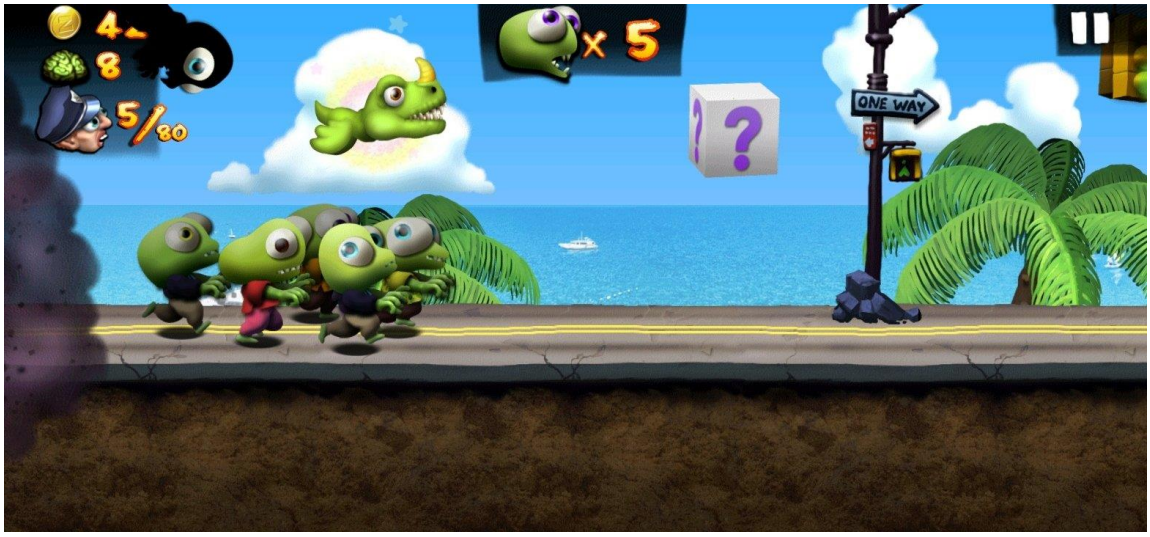

Рисунок 1.6 – Загальний вигляд ігрового процесу комп’ютерної гри “Zombie Tsunami” концепції типу "Стратегічне керування" (Strategic Management)

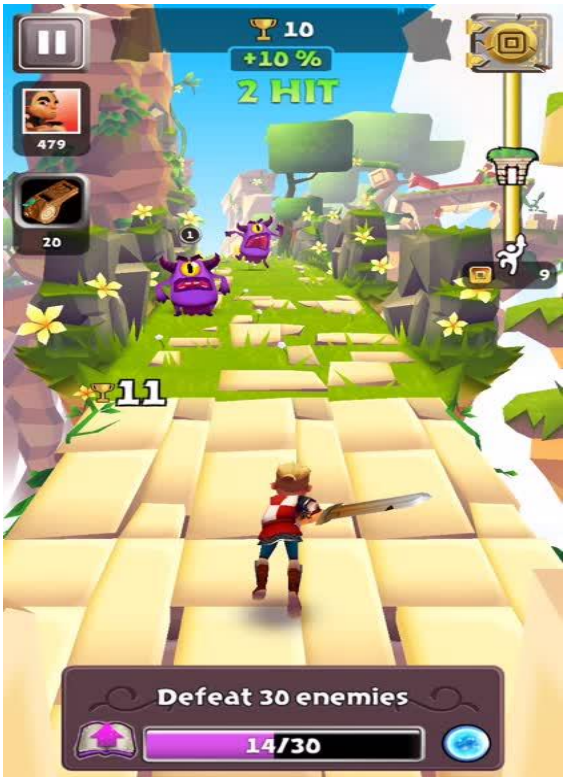

Рисунок 1.7 – Загальний вигляд ігрового процесу комп’ютерної гри “Blades of Brim” концепції типу "Стратегічне керування" (Strategic Management)

Ця концепція додає стратегічні елементи до типового gameplay runner ігор, що може привертати любителів глибокого геймплею та підвищити інтерес до жанру.

Концепція "Сюжетно–орієнтовані" (Narrative–driven):

В цих комп’ютерних іграх існує виражений сюжет або місія, яку гравець повинен виконати під час бігу. Історія може розкриватися під час гри через відеоролики, діалоги або елементи гри.

Прикладом даного типу комп’ютерних ігор на рисунку 1.8, 1.9: "Rayman Origins", "Journey" [3].

Рисунок 1.8 – Загальний вигляд ігрового процесу комп’ютерної гри “Rayman Origins” концепції типу "Сюжетно–орієнтовані" (Narrative–driven)

Рисунок 1.9 – Загальний вигляд ігрового процесу комп’ютерної гри “Journey” концепції типу “Сюжетно–орієнтовані” (Narrative–driven)

Ця концепція надає більше значення кожному руху гравця, оскільки кожна дія може вплинути на розвиток історії або кінцевий результат.

1.2 Аналіз використання властивості кроссплатформенності комп’ютерних ігор

У наш час налічується чимало різноманітних платформ, на які випускаються нові, та перевидаються старі ігри. На рисунку 1.10 зображений рейтинг популярності ігрових платформ:

Рисунок 1.10 – Діаграма рейтингу популярності ігрових платформ

Мобільна ігрова платформа в останні десятиліття стала однією з найбільш впливових у світі геймінгу. Проведемо аналіз даної платформи:

1. Доступність і поширеність: Мобільні пристрої, такі як смартфони та планшети, стали практично необхідними аксесуарами для більшості людей у сучасному світі. Це робить мобільну ігрову платформу дуже доступною та поширеною. Гравці можуть легко завантажувати ігри з магазинів додатків, таких як App Store чи Google Play, і грати в них у будь–який час та в будь–якому місці [3].

2. Різноманітність ігор: Мобільна платформа має вражаючий асортимент ігор, що охоплює різні жанри і аудиторії. Від простих головоломок і аркад до складних стратегій та RPG, ігри для мобільних пристроїв можуть задовольнити потреби різних типів гравців [3].

3. Ігрова економіка: Мобільні ігри використовують різні моделі монетизації, такі як безкоштовні ігри з мікротранзакціями, підписки, реклама та платні додаткові контенти. Ця різноманітність дозволяє розробникам максимізувати дохід від своїх ігор та привертати різні аудиторії [3].

4. Соціальний аспект: Багато мобільних ігор пропонують можливості для соціального взаємодії, такі як гра з друзями, конкурси, спільні досягнення та

співробітництво. Це дозволяє гравцям спілкуватися та взаємодіяти у віртуальному середовищі.

5. Технологічний прогрес: Із зростанням потужності мобільних пристроїв, які працюють на основі сучасних мобільних процесорів та графічних чіпів, які набагато потужніші, ніж у попередні роки, графіка та фізика в мобільних іграх значно покращилися, дозволяючи створювати більш іммерсивні ігрові досвіди.

6. Проблеми та обмеження: Одним з викликів для мобільних ігор є обмежені можливості управління, особливо для ігор, які потребують точності та швидкості. Крім того, конкуренція в магазинах додатків дуже велика, що може зробити важким виділення вашої гри серед інших [3].

Загалом мобільна ігрова платформа є важливим сегментом індустрії геймінгу, який продовжує рости і розвиватися з кожним роком [3]. Рейтинг найпопулярніших безкоштовних та платних ігор на мобільній платформі зображено на рисунку 1.11.

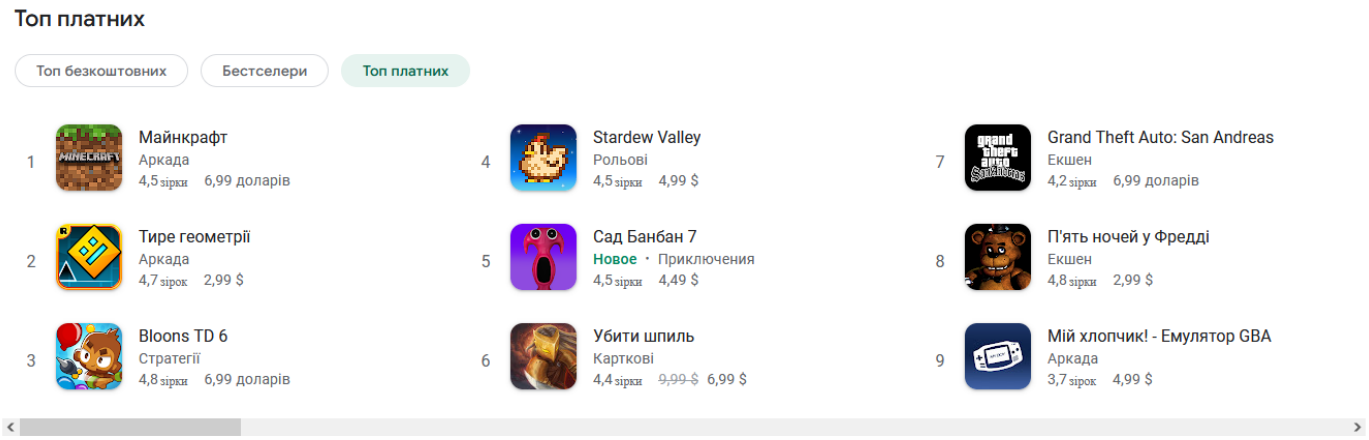

Рисунок 1.11 – Рейтинг найпопулярніших платних мобільних ігор у магазині Google Play

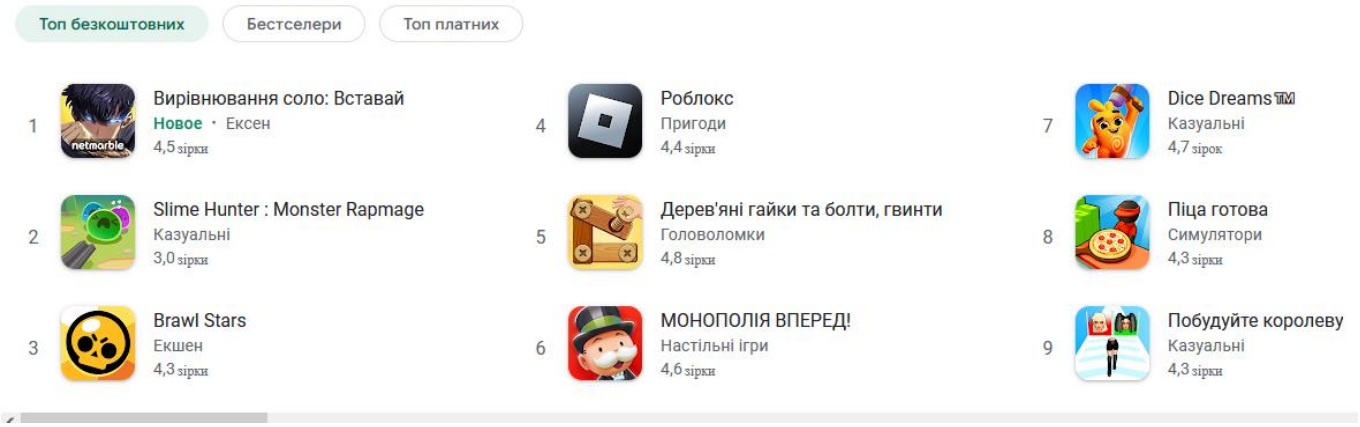

Рисунок 1.12 – Рейтинг найпопулярніших безкоштовних мобільних ігор у магазині Google Play

На сьогоднішній день комп’ютери є основною платформою для великих студій, оскільки вони дозволяють створювати великі та важкі ігри, з красивою графікою, великими світами, так звані AAA (triple A) ігри. Проведемо аналіз даної платформи:

1. Гнучкість і можливості: Комп'ютери є найбільш гнучкою платформою для ігор, оскільки вони забезпечують широкий спектр можливостей для налаштування та модифікації. Гравці можуть використовувати різні типи контролерів, від клавіатури та миші до геймпадів та керм. Крім того, комп'ютери можуть бути оновлені, щоб забезпечити кращі графічні можливості та продуктивність [3].

2. Графічна якість та продуктивність: Комп'ютери зазвичай мають найвищу графічну якість серед ігрових платформ. Вони можуть відтворювати графіку з високою деталізацією, роздільною здатністю та плавними кадрами на високих налаштуваннях графіки. Крім того, комп'ютери зазвичай мають потужні процесори та відеокарти, що дозволяє їм запускати графічно вимогливі ігри без проблем [3].

3. Інтерактивність та соціальні можливості: Комп'ютерні ігри можуть надавати більші можливості для взаємодії та соціального спілкування, особливо в онлайн–іграх. Гравці можуть взаємодіяти один з одним у реальному часі через спеціальні ігрові сервіси та платформи спільного гри [3].

4. Ексклюзивність ігор: Існують багато ексклюзивних ігор, доступних тільки на платформі комп'ютерів, що робить її привабливою для багатьох гравців. Ці ігри можуть бути створені спеціально для комп'ютерів з урахуванням їх унікальних можливостей та функцій [3].

5. Розвиток і моддинг: Комп'ютерні ігри часто залучають активну спільноту моддерів, які створюють модифікації для існуючих ігор або навіть створюють свої власні ігри. Це дозволяє гравцям насолоджуватися новими варіантами гри та розширювати її функціональність [3].

6. Вартість інвестиції: Вартість власного комп'ютера для гри може бути значною, особливо якщо ви прагнете мати найновіші компоненти для максимальної продуктивності. Крім того, підтримка та поновлення обладнання також може стати витратною [3].

Загалом комп'ютери залишаються однією з найбільш популярних і впливових платформ для ігор завдяки своїм гнучкості, продуктивності та широкому вибору ігор [3]. Список рейтингу найпопулярніших комп’ютерних ігор зображено на рисунку 1.13.

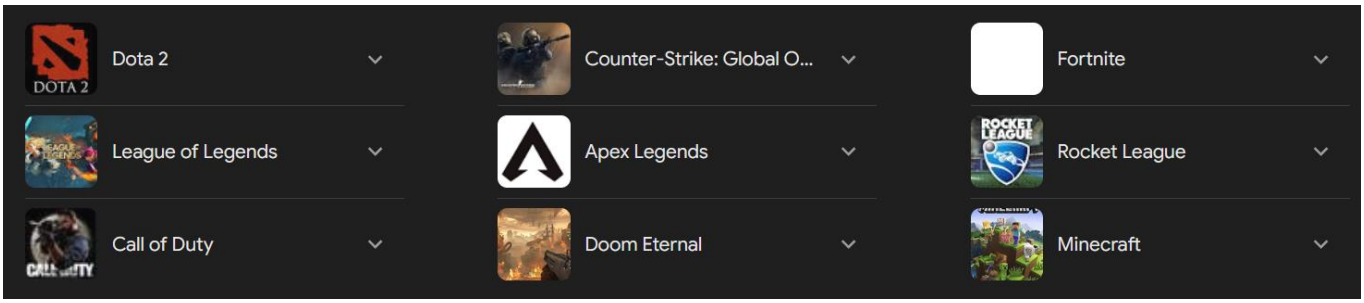

Рисунок 1.13 – Список рейтингу найпопулярніших комп’ютерних ігор

На сьогоднішній день, на відміну від кількох минулих років, консолі почали набирати все більше і більше популярності, причиною цього може бути багато факторів, основні з яких, на мою думку, це вищезгадана пандемія, та можливість

одноразової покупки девайсу, без потреби у його апгрейді. Проведемо аналіз даної платформи:

1. Ексклюзивність ігор: Однією з найбільших переваг консолей є наявність ексклюзивних ігор, тобто ігор, які доступні тільки на певній консолі. Це включає в себе такі відомі франшизи, як "The Last of Us" для PlayStation [4] або "Halo" для Xbox [4]. Ексклюзивні ігри можуть привертати до консолей гравців, які хочуть насолоджуватися унікальними ігровими досвідами [4].

2. Спеціалізовані контролери: Консолі надають спеціалізовані контролери, такі як геймпади, які часто оптимізовані для конкретних жанрів ігор. Це робить геймплей більш комфортним та приємним для гравців [4].

3. Простота використання: Консолі зазвичай мають простий та інтуїтивно зрозумілий інтерфейс, що робить їх доступними для широкої аудиторії, включаючи навіть тих, хто не має досвіду з комп'ютерами [4].

4. Онлайн сервіси та мережева гра: Більшість консольних виробників надають мережеві сервіси, такі як PlayStation Network або Xbox Live, які дозволяють гравцям грати в ігри онлайн, спілкуватися з друзями та отримувати доступ до додаткового контенту [4].

5. Функціональні можливості: Сучасні консолі часто включають в себе додаткові функціональні можливості, такі як стрімінг відео, музики та інших розважальних послуг, що робить їх центром розваг у домашньому середовищі.

6. Вартість і підтримка ігор: Вартість консолі та ігор може бути високою, а також вартість підписок на онлайн сервіси. Крім того, консолі зазвичай мають обмежену сумісність з іграми попередніх поколінь, що може призвести до необхідності перезакупівлі ігор на нову платформу [4].

Загалом консолі залишаються популярною платформою для ігор завдяки своїм ексклюзивним іграм, спеціалізованим контролерам та простоті використання. Вони є

хорошим варіантом для гравців, які шукають спеціалізовану ігрову платформу з високоякісними ексклюзивами та мережевими можливостями [4].

1.3 Аналіз відомих комп'ютерних ігор жанру "runner"

Перевагами 2D комп'ютерних ігор є:

Простота геймплею – Більшість 2D ігор жанру "runner" мають прості та легко зрозумілі механіки гри. Гравець керує персонажем, який рухається по горизонталі або вертикалі, уникаючи перешкод та збираючи бонуси.

Відсутність глибини простору – У 2D іграх гравець зазвичай обмежений рухом вгору–вниз або вліво–вправо. Це робить їх простими для розуміння та грабельків для новачків.

Стиль – Багато 2D ігор жанру "runner" мають яскраву та стилізовану графіку, що робить їх привабливими для гравців [5].

Приклад комп'ютерної гри "Vector" жанру "runner" у 2D зображено на рисунку 1.14:

Рисунок 1.14 – Загальний вигляд ігрового процесу комп'ютерної гри Vector жанру "runner" у 2D

Первагами 3D комп’ютерних ігор є:

Розширені можливості руху – У 3D іграх гравцю доступні більші можливості руху. Вони можуть включати обертання камери, рух у трьох вимірах та використання більш складних архітектурних об'єктів [5].

Глибший геймплей – Більшість 3D ігор жанру “runner” мають глибший геймплей, оскільки гравець може взаємодіяти з більшим обсягом середовища та об'єктів.

Візуальні ефекти – 3D ігри часто використовують вражаючі візуальні ефекти, такі як реалістичне освітлення, тіні та текстури, щоб створити більш іммерсивний досвід для гравців.

Складніше управління – Управління персонажем у 3D іграх може бути складнішим через більший обсяг можливих рухів та взаємодій .

Приклад комп’ютерної гри “Subway Surfers” жанру “runner” у 3D [5] зображено на рисунку 1.15:

Рисунок 1.15 – Загальний вигляд ігрового процесу комп’ютерної гри Subway Surfers жанру “runner” у 3D

1.4 Висновок до розділу 1

Було досліджено існуючі концепції ігор жанру “runner”. З’ясовано, що основними характеристиками цього жанру є безперервне пересування персонажа через нескінченний світ, уникання перешкод, збір бонусів та виконання місій. Ігри жанру “runner” зазвичай мають просте управління, що робить їх доступними для широкої аудиторії, включаючи дітей та дорослих. Прикладами популярних ігор цього жанру є Subway Surfers та Temple Run. Ці ігри демонструють високий рівень динаміки та захоплюючий геймплей.

Було проведено аналіз кроссплатформенності комп'ютерних ігор, що показав важливість підтримки різних платформ для досягнення комерційного успіху гри. Кроссплатформені ігри дозволяють охопити широку аудиторію, збільшуючи потенційні прибутки. Приклади успішних кроссплатформених “runner”–ігор включають Temple Run та Sonic Dash, які доступні на різних пристроях, включаючи мобільні телефони, планшети та комп'ютери

Було досліджено різні види комп'ютерних ігор жанру “runner”. Виявлено, що основні підвиди включають класичні нескінченні раннери, рівневі раннери та гібридні раннери. Класичні нескінченні раннери, такі як Subway Surfers, фокусуються на досягненні найвищого можливого рахунку шляхом безперервного бігу персонажа. Рівневі раннери, як Rayman Jungle Run, пропонують більш структуроване проходження з конкретними завданнями на кожному рівні. Гібридні раннери, наприклад, Blades of Brim, поєднують елементи інших жанрів, таких як RPG, додаючи глибину і різноманітність ігровому процесу [6].

Було досліджено існуючі концепції ігор жанру “runner”, наведено приклади популярних ігор та визначено їхні ключові характеристики. Аналіз кроссплатформенності показав важливість підтримки різних платформ для досягнення успіху гри та наведено приклади інструментів, які спрощують цей процес.

2 РОЗРОБКА І ПРОЄКТУВАННЯ ПРОГРАМНОГО МОДУЛЯ КОМП'ЮТЕРНОЇ ГРИ

2.1 Проєктування інтерфейсу користувача

Проектування інтерфейсу користувача. Інтерфейс користувача(англ. UI – User interface) це графічні елементи взаємодії користувача із застосунком. Зазвичай складається з певних вікон, меню, кнопок, слайдерів і так далі. Метою розробників є створити максимально простий та зрозумілий інтерфейс.

Якщо застосунок є кросплатформеним, то розробники стикаються ще з одною проблемою, а саме з інтеграцією інтерфейсу на різні пристрої, які використовують різні інструменти введення: телефони це сенсорний екран, комп'ютер – клавіатура, миша, консолі – геймпади і так далі [7].

Під час проектування було складено список елементів, які будуть використані під час створення інтерфейсі користувача:

1. Головне меню – першу, що бачить гравець коли заходить в гру. Дане меню надаватиме доступ до основних можливостей гри. Схему головного меню зображено на рисунку 2.1
2. Меню налаштувань – містити інструменти для зміни ігрового середовища та ігрового досвіду. Схему меню налаштувань зображено на рисунку 2.2.
3. Внутрішньоігровий магазин – міститиме доступні для покупки речі для кастомізації гри: рівня та ігрового персонажа. Схему меню налаштувань зображено на рисунку 2.3.
4. Меню паузи – під час гри можна відкрити меню паузи, аби повернутися до головного меню під час гри. Схему меню налаштувань зображено на рисунку 2.4.

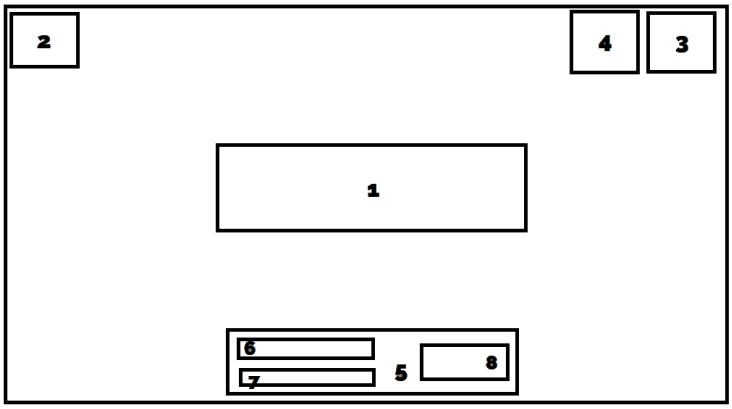

Рисунок 2.1 – Конструктивна схема головного меню

На рисунку 2.1 відповідно:

1. Кнопка «Почати гру».
2. Кнопка увімкнути/вимкнути музику.
3. Кнопка «Налаштування».
4. Кнопка доступу до внутрішньоігрового магазину.
5. Панель рекордів та валюти.
6. Напис з минулим результатом гри.
7. Напис рекордним результатом гри.
8. Напис з кількістю зібраної валюти гравця.

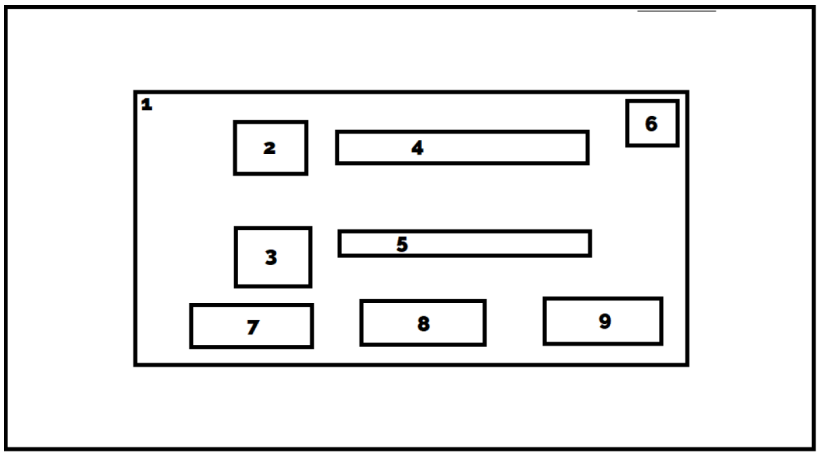

Рисунок 2.2. – Конструктивна схема меню налаштувань

На рисунку 2.2 відповідно:

1. Панель налаштувань.
2. Зображення «Звук».
3. Зображення «Музика».
4. Горизонтальний слайдер для зміни гучності звуку.
5. Горизонтальний слайдер для зміни гучності музики.
6. Кнопка виходу з меню налаштувань.
7. Кнопка для зміни часу у грі на «День».
8. Кнопка для зміни часу у грі на «Ніч».
9. Кнопка для зміни часу у грі на «Випадково».

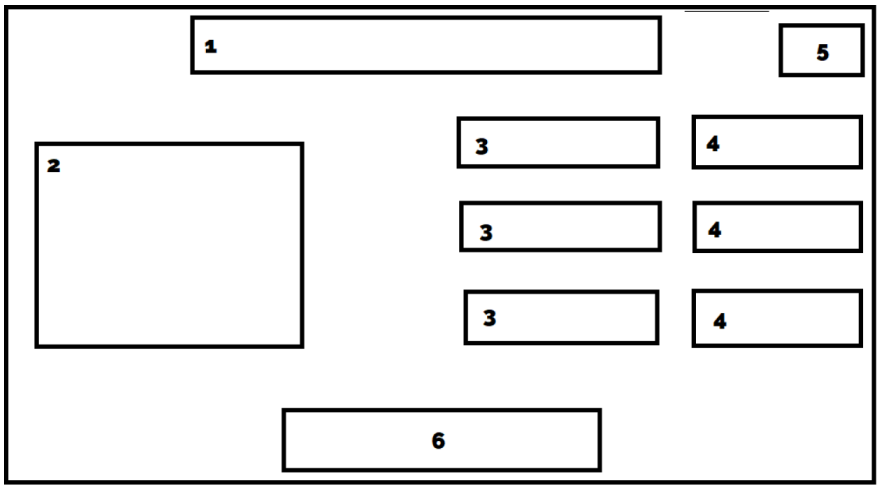

Рисунок 2.3. – Конструктивна схема внутрішньоігрового магазина

На рисунку 2.3 відповідно:

1. Напис «Натисни щоб купити».
2. Зображення наявного вигляду рівня та персонажа .
3. Елемент кастомізації рівня.
4. Елемент кастомізації персонажа.
5. Кнопка виходу з внутрішньоігрового магазину.

6. Кількість наявної у гравця валюти.

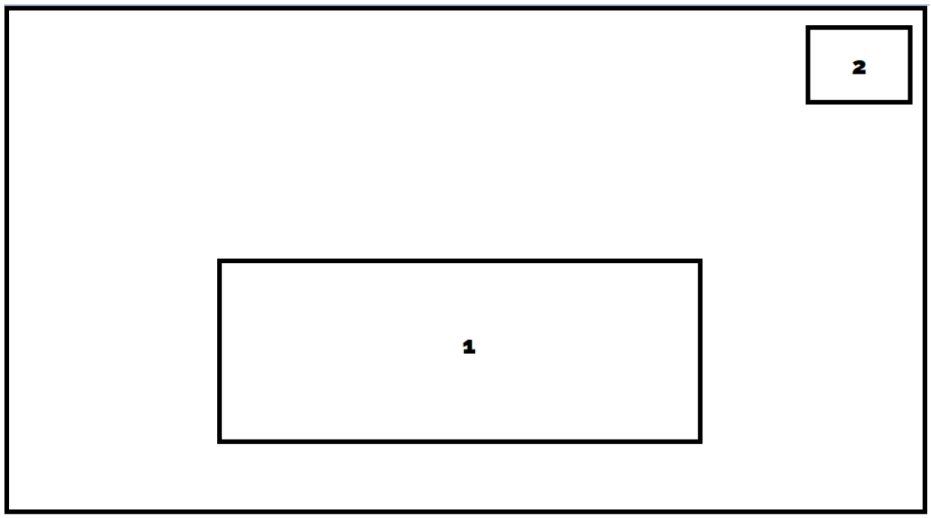

Рисунок 2.4. – Конструктивна схема меню паузи

На рисунку 2.4 відповідно:

1. Кнопка «Повернутися до головного меню».
2. Кнопка виходу з меню паузи.

2.2 Проєктування моделі реалізації ігрової системи

Перед входом на ігрову сцену задіюється модуль налаштування рівня, після чого ігрова сцена обмінюється даними з моделлю створення ігрових об’єктів та модулем обробки контролю персонажем, який в свою чергу обмінюється з модулем обробки дій гравця [7]. На рисунку 2.5 зображено модель реалізації ігрової системи.

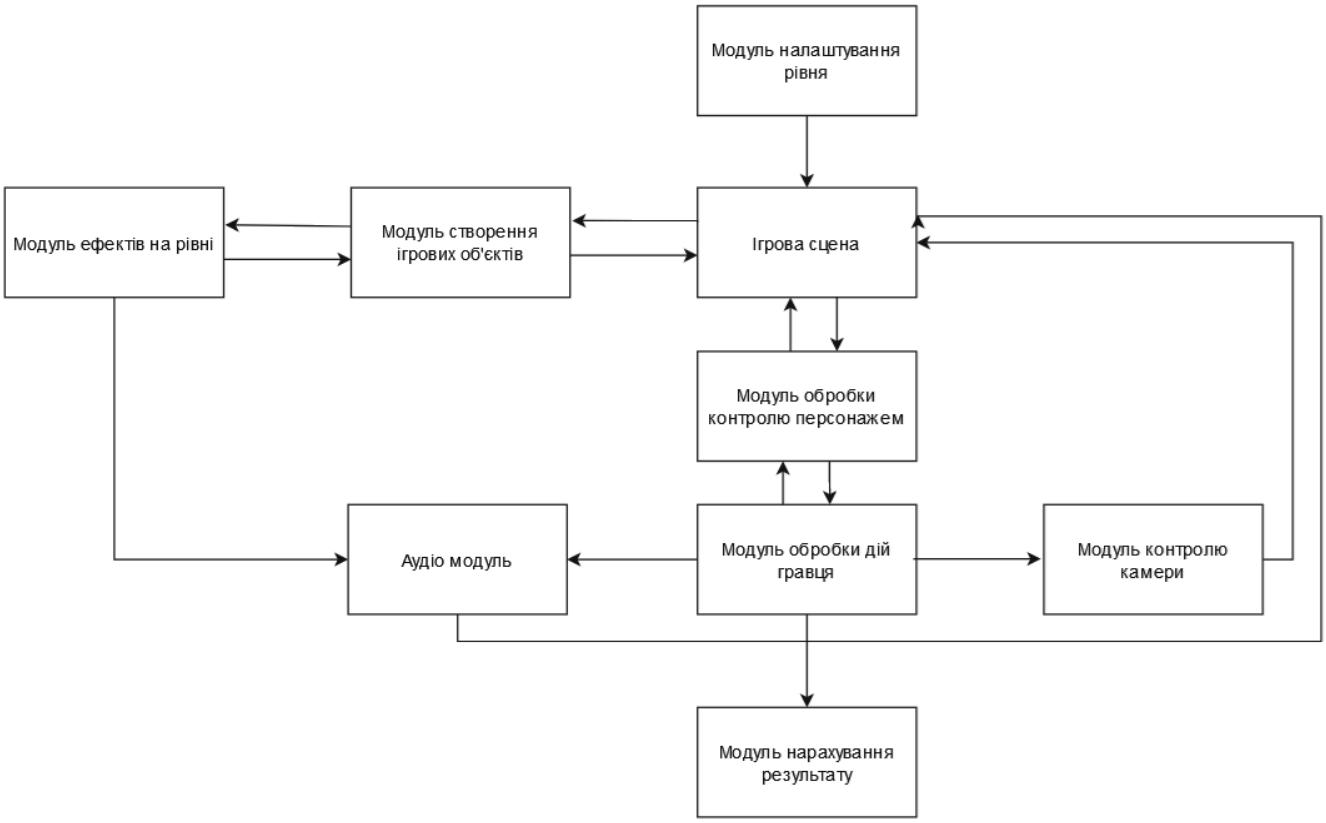

Рисунок 2.5 – Типова схема моделі реалізації ігрової системи

Діаграма ігрового циклу (game loop diagram) – це візуальне зображення основного циклу функціонування ігри. Вона відображає послідовність операцій, які відбуваються в грі від моменту початку до моменту завершення, а також взаємозв'язки між цими операціями [8].

Основна мета діаграми ігрового циклу – показати, як ігровий двигун (чи будь–яка інша програмна система, що керує грою) обробляє введення від гравця, оновлює стан гри, взаємодіє з ігровим світом та відображує оновлений стан на екрані. Ця діаграма є ключовою для розуміння роботи гри та використовується як розробниками, так і технічними архітекторами для аналізу та вдосконалення ігрового процесу [8].

Зазвичай діаграма ігрового циклу включає такі етапи:

1. Перевірка введення: Гра перевіряє введення від користувача, таке як натискання клавіш або дотик на сенсорних пристроях.

2. Оновлення стану гри: Гра оновлює стан всіх об'єктів у світі гри, таких як рух персонажів, анімація, фізика тощо.

3. Зміна стану гри: Гра перевіряє умови для зміни стану гри, такі як зіткнення об'єктів, досягнення мети тощо.

4. Відображення графіки: Гра оновлює графічний вивід на екрані, відображаючи оновлений стан гри гравцеві.

Ця діаграма є важливим інструментом для розробників і допомагає їм краще розуміти та управляти роботою гри. Діаграма зображена на рисунку 2.6

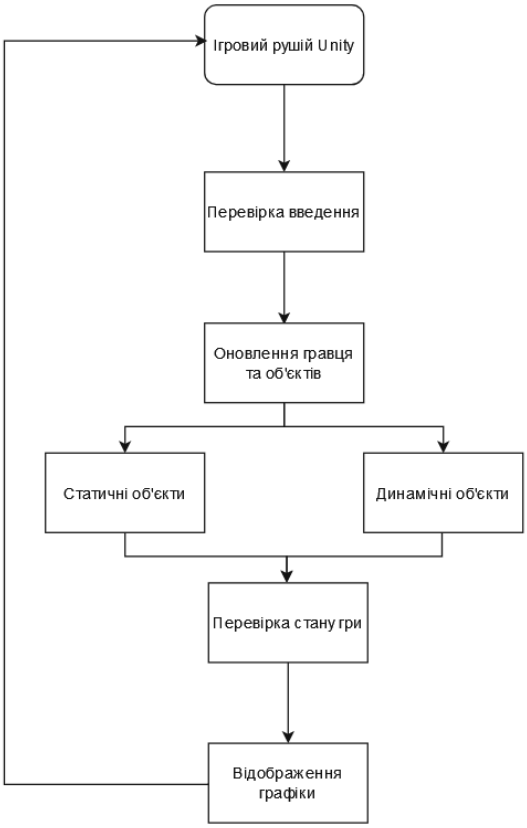

Рисунок 2.6 – Діаграма ігрового циклу

Одним із важливих компонентів гри є система збереження. Для того аби гра щоразу не починалася з чистого листа та гравцю не довелося ставити нові рекорди та збирати валюту потрібно аби ці дані були десь збережені. Популярними методами збереження є:

1. Збереження у файлах (JSON, XML, Binary)
2. Бази даних (SQL, SQLite)
3. Хмарні сервіси (PlayFab, Firebase)
4. Інструменти ігрового рушія (PlayerPrefs)

Кожен з даних методів збереження ігрових даних має свої переваги та недоліки.

1. Збереження у файлах:

Переваги:

– Легко читається та пишеться.
– Широко підтримується у багатьох мовах програмування.
– Добре підходить для збереження складних об’єктів та структур даних.

Недоліки:

– Потребує додаткових обробок для безпеки.
– Може займати більше місця на диску у порівнянні з бінарним форматом.
– При великих об’ємах даних потребує додаткових обробок.
– Після деяких обробок може стати нечитаємим для людей.
– Складніший в налагодженні та обслуговування.

2. Використання баз даних:

Переваги:

– Добре підходить для великих обсягів даних.
– Підтримує складні запити та реляційні структури.
– Висока продуктивність при роботі з великими наборами даних.
– Безпека зберігання даних.

Недоліки:

– Складність налаштування та підтримки.

– Займає більшу кількість місця у порівнянні з іншими методами збереження даних.

– Вимагає високого рівня знань розробника щоб забезпечити правильну роботу.

3. Використання хмарних сервісів:

Переваги:

– Дозволяє зберігати дані на сервері, де до них можна отримати доступ з будь-якого пристрою.

– Підтримка багатокористувацьких ігор та соціальних функцій.

– Безпека та резервне копіювання даних.

Недоліки:

– Вимагає постійного підключення до інтернету.

– Більша складність у налаштуванні та використанні.

– Може мати обмеження на безкоштовні плани.

4. Використання інструментів ігрового рушія:

Переваги:

– Простота реалізації.

– Не потребує застосування зовнішнього програмного забезпечення.

– Портативність

Недоліки:

– Низька масштабованість.

– Не підходить для великих проектів.

– Нижча безпека даних у порівнянні з іншими методами.

Усі вищенаведені способи зберігання даних були порівняні у таблиці 2.1.

Таблиця 2.1 – Порівняння методів збереження даних

Метод	Простота реалізації	Зручність використання	Масштабованість	Безпека	Портативність
PlayerPrefs	Висока	Висока	Низька	Низька	Висока
JSON/XML	Середня	Середня	Середня	Середня	Висока
SQLite	Низька	Середня	Висока	Висока	Висока
Хмарні сервіси	Низька	Середня	Висока	Висока	Висока

Оскільки проєкт, що буде розроблений у даній бакалаврській роботі не потребує збереження великої кількості збережених даних, або складних структур, внутрішній інструмент ігрового рушія PlayerPrefs є оптимальним вибором через наступні причини:

– Дуже простий у використання, не вимагає додаткових бібліотек чи налаштувань.

– Дані зберігаються у вигляді ключ-значення, що робить їх збереження та завантаення швидким та легким.

– Дані зберігаються на пристрої користувача, що робить цей метод дуже портативним і незалежним від підключення до мережі.

На рисунку 2.7 зображено діаграму механізму зберігання ігрових даних.

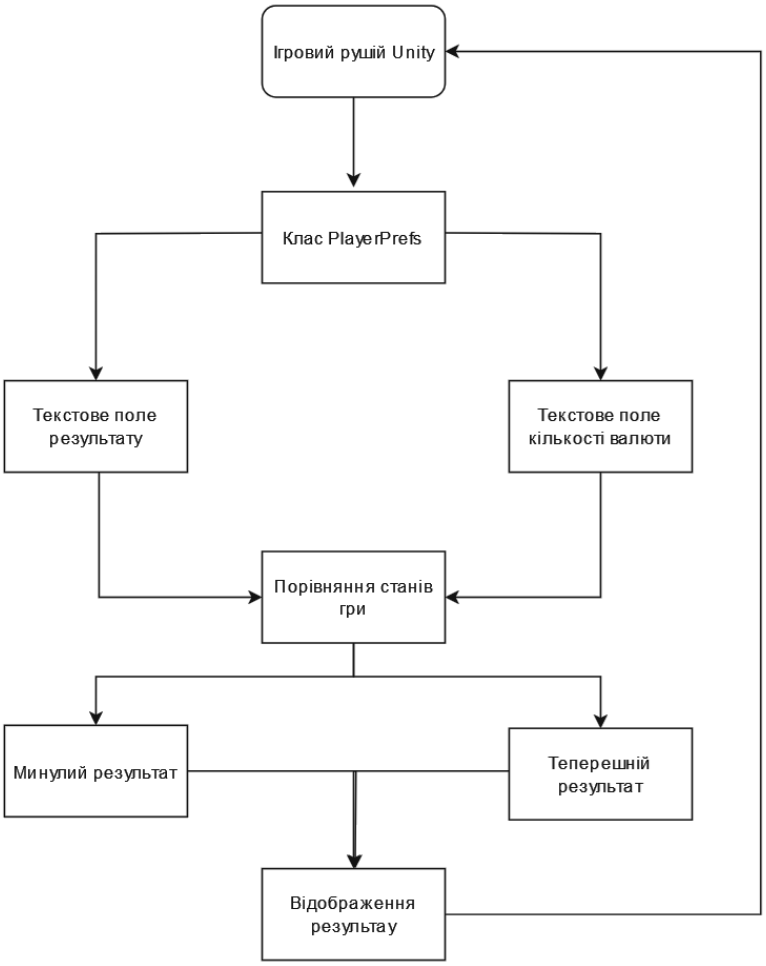

Рисунок 2.7 – Діаграма механізму зберігання ігрових даних.

2.3 Проєктування алгоритму роботи програми

Алгоритм роботи 2D відеогри жанру “runner” починається з появи гравця на рівні. Коли гравець починає рух, то перед ним процедурно генеруватимуться перешкоди та предмети які можна збирати. Перешкоди являють собою платформи якими рухається гравець, а також шипи та сфери, при контакті з якими гравець програє. Також гравець програє, якщо упав з платформи у безодню [7].

На рівні розміщені монетки у великій кількості та рідше додаткові життя, які після контакту з перешкодами дають гравцю ще один шанс продовжити гру.

Алгоритм роботи програми – це послідовність інструкцій або кроків, які необхідно виконати для досягнення певної мети або вирішення конкретного завдання. Алгоритм описує логіку дій, що виконуються програмою, включаючи умовні переходи, цикли, обробку даних та інші операції.

Основні елементи алгоритму:

– Вхідні дані: Дані, які використовуються програмою для виконання завдань.

– Послідовність дій: Чітко визначений порядок виконання інструкцій.

– Умови: Логічні перевірки, що визначають, який шлях обрати в залежності від певних умов.

– Цикли: Повторювані дії, які виконуються до досягнення певної умови.

– Вихідні дані: Результати виконання алгоритму.

Необхідність алгоритму роботи програми:

– Планування та організація Алгоритми допомагають розробникам чітко спланувати послідовність дій, які має виконати програма. Вони забезпечують структурованість та логічність процесу розробки.

– Зрозумілість та читабельність коду: Добре спроектований алгоритм робить програму більш зрозумілою та читабельною для інших розробників. Це полегшує підтримку, відлагодження та розширення програмного забезпечення.

– Ефективність: Алгоритми дозволяють оптимізувати виконання завдань, що може призвести до зменшення часу обробки даних та використання ресурсів. Вони допомагають виявити і усунути потенційні вузькі місця в продуктивності програми.

– Тестування та налагодження: Алгоритми спрощують процес тестування програм, оскільки дозволяють розробникам легко визначити, які частини коду відповідають за які дії. Це також допомагає швидко знайти та виправити помилки.

– Узгодженість та повторюваність: Алгоритми забезпечують узгодженість виконання дій, що гарантує повторювані результати за однакових умов. Це важливо

для програм, де точність і послідовність виконання завдань мають вирішальне значення.

– Навчання та документування: Алгоритми є важливими для навчання нових розробників, допомагаючи їм зрозуміти логіку і структуру програмного забезпечення. Вони також сприяють створенню якісної документації, яка може використовуватися для подальшого розвитку та підтримки програми.

Блок–схему алгоритму роботи 2D відеогри жанру “runner” зображено на рисунку 2.8.

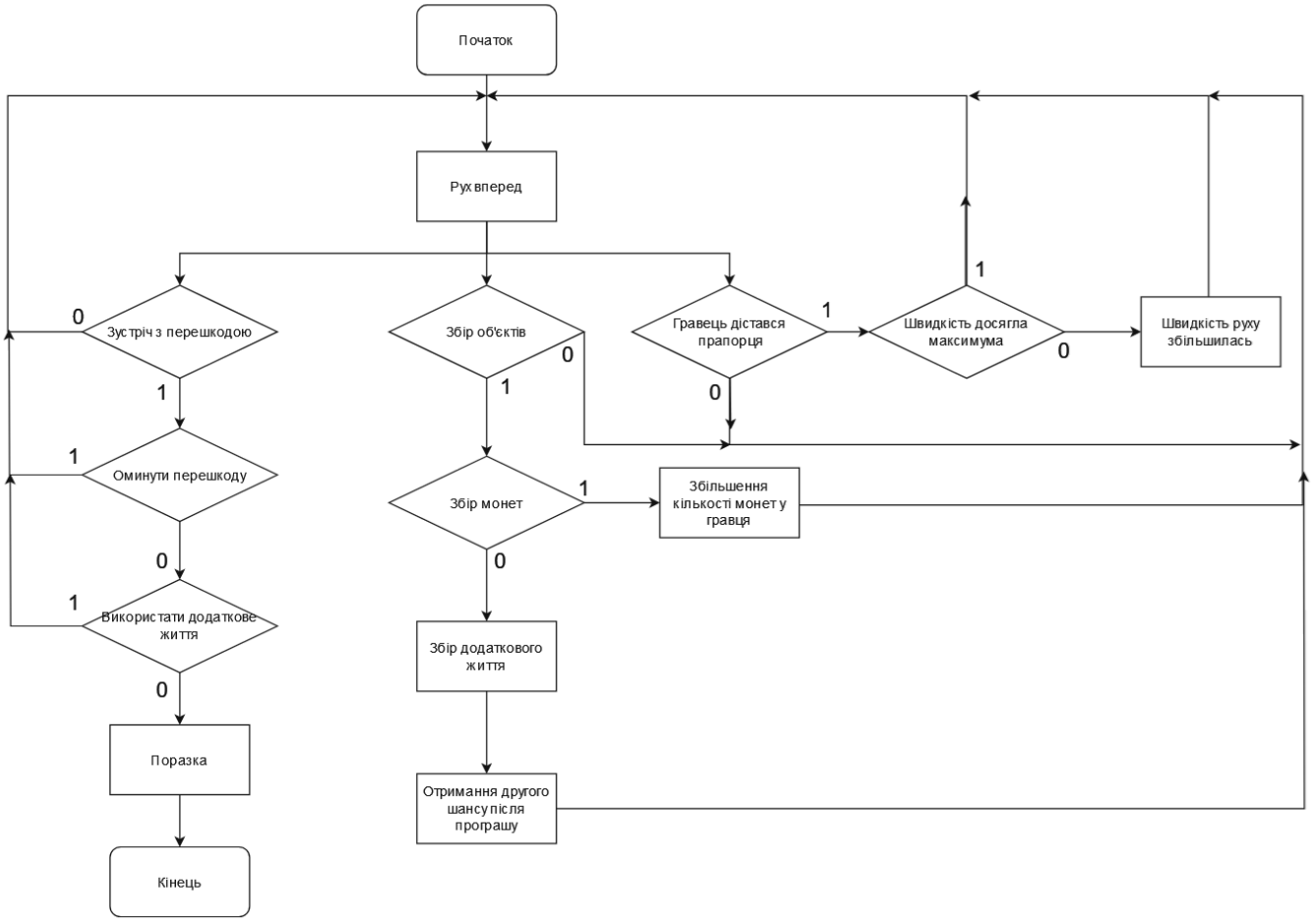

Рисунок 2.8 – Схема алгоритму роботи 2D відеогри жанру “runner”

Платформи, якими рухатиметься гравець будуть випадково генеруватися перед ним, за рахунок чого кожного разу рівень буде унікальним. Це допомагає скоротити час та ресурси на розробку унікальних процедурно генерованих рівнів, замість створення кожного рівня вручну, створюючи унікальні середовища та дизайни для них.

На початку гри генерується стартова платформа, аби гравець одразу не впав у безодню. Після того як гравець почне рух, платформи генеруватимуться за екраном перед ним, та зникатимуть за ним. Це забезпечить безшовний перехід між платформами, та ігровий досвід гравця буде кращим, бо створюється ілюзія безкінечного, випадково генерованого світу. На рисунку 2.9 зображено діаграму генерації платформ рівня.

Важливим елементом проєктування є функціональна модель персонажа. Це опис усіх функцій та можливостей якими володіє персонаж. Вона включає в себе всі можливі дії персонажа, його взаємодію з елементами гри та її середовищем. Модель також може описувати характеристики персонажа, анімації, поведінкові сценарії, та логіку обробки подій.

Елементи функціональної моделі персонажа:

1. Рух: ходьба, біг, стрибки, підкати.
2. Взаємодія з об’єктами: підбирання об’єктів, використання об’єктів.
3. Спеціальні здібності: використання магії, відновлення, тощо.
4. Взаємодія з елементами інтерфейсу: кастомізація персонажа.

Рисунок 2.9 – Діаграма генерації платформ рівня.

Основні функції функціональної моделі персонажа:

– Чітке визначення вимог: Функціональна модель допомагає чітко визначити всі вимоги до персонажа, що спрощує процес розробки.

– Планування та організація: Наявність детального опису всіх можливостей та взаємодій персонажа дозволяє розробникам планувати та організовувати роботу більш ефективно.

– Забезпечення узгодженості: Функціональна модель допомагає забезпечити узгодженість у розробці персонажа, зменшуючи ризик виникнення непередбачених проблем та конфліктів між різними аспектами гри.

– Полегшення комунікації: Детальний опис функцій та можливостей персонажа спрощує комунікацію між членами команди, дозволяючи кожному зрозуміти, як повинен працювати персонаж.

– Спрощення тестування та відлагодження: Функціональна модель дозволяє легше тестувати та відлагоджувати персонажа, оскільки всі його можливості та дії задокументовані та можуть бути перевірені на відповідність вимогам.

– Адаптивність та розширюваність: Наявність детальної моделі персонажа робить його більш адаптивним до змін і розширень. Розробники можуть легко додавати нові можливості або модифікувати існуючі, маючи чітке уявлення про те, як це вплине на загальну функціональність персонажа.

Функціональна модель персонажа зображена на рисунку 2.10

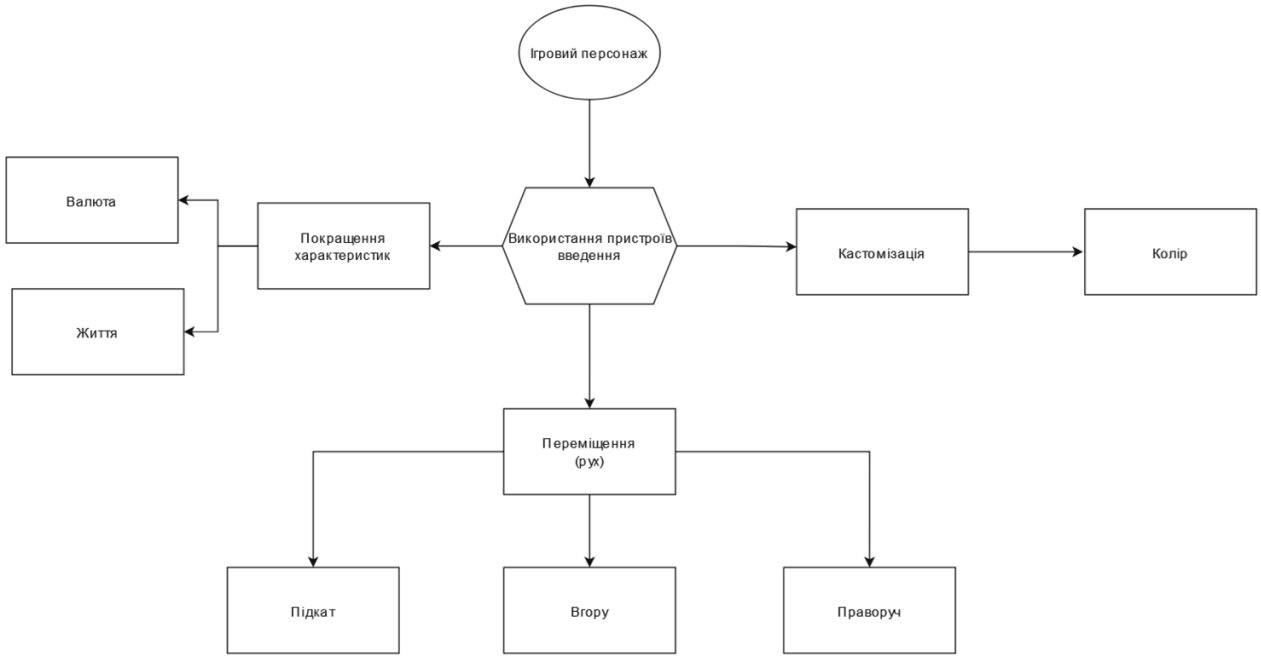

Рисунок 2.10 – Функціональна модель персонажа

На функціональній моделі персонажа показано усі можливості, що надаються гравцю у відеогрі, а саме:

1. Рух вгору – стрибок, рух вправа – біг, рух вниз – підкат.
2. Можливість змінити колір колір персонажа – кастомізація.
3. Можливість збирати монетки і заробляти валюту.
4. Можливість зібрати додаткове життя під час гри.

2.4 Висновок до розділу 2

У другому розділі дипломної роботи було проведено всебічне проектування користувацького інтерфейсу, розроблено алгоритм функціонування гри, спроектовано ігровий цикл та створено детальну модель реалізації ігрової системи для 2D гри у жанрі раннер [9].

Проектування користувацького інтерфейсу почалося з визначення ключових елементів, які забезпечують інтуїтивно зрозумілий доступ до функцій гри. На цьому етапі були створені макети інтерфейсу, які сприяють зручній та ефективній взаємодії користувача з грою.

Розробка алгоритму роботи гри включала створення механізмів генерування рівнів, обробки дій гравця та управління ігровими подіями. Було розроблено основний алгоритм, який забезпечує безперебійну роботу гри та оптимальну продуктивність завдяки ефективним методам обробки даних та управління ресурсами. Цей алгоритм враховує різноманітні ігрові сценарії, включаючи випадкові події та динамічні зміни в процесі гри [9].

Проєктування ігрового циклу охопило всі необхідні стадії від ініціалізації гри до завершення сесії. Ігровий цикл розроблено таким чином, щоб забезпечити плавність переходу між різними етапами гри та підтримувати інтерес гравця протягом всієї сесії.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ КОМП'ЮТЕРНОЇ ГРИ

3.1 Аналіз і обґрунтування вибору мов програмування

C# був обраний для розробки вашого проекту через низку переваг, які роблять його ідеальним для створення ігор в Unity. По-перше, Unity використовує C# як основну мову програмування, що забезпечує оптимальну інтеграцію з ігровим движком і доступ до потужних API, які спрощують використання різних функцій, таких як фізика, анімація та введення користувача. Це робить C# найбільш відповідним і підтримуваним варіантом для роботи в Unity.

C# має простий і зрозумілий синтаксис, що полегшує процес навчання та використання, навіть для новачків. Завдяки цьому розробники можуть швидко приступити до створення гри і зосередитись на її функціональності. Строга типізація і підтримка об'єктно-орієнтованого програмування допомагають створювати структурований, читабельний і легко підтримуваний код.

Підтримка об'єктно-орієнтованого програмування в C# дозволяє використовувати всі основні принципи ООП, такі як інкапсуляція, наслідування і поліморфізм, що сприяє створенню модульного, повторно використовуваного та масштабованого коду. У Unity кожен ігровий об'єкт складається з компонентів, і C# ідеально підходить для реалізації такої архітектури, дозволяючи легко створювати і керувати компонентами.

C# забезпечує високу продуктивність, що є критично важливим для ігрової розробки, де кожен кадр має значення. Оптимізація роботи з пам'яттю та швидке виконання коду дозволяють створювати гладкі ігрові процеси. Крім того, підтримка багатопотоковості в C# дозволяє ефективно використовувати ресурси багатоядерних процесорів, що важливо для виконання ресурсомістких завдань в іграх, таких як обробка фізики або штучного інтелекту.

Розширюваність і гнучкість C# також є важливими факторами. Завдяки використанню C# можна легко інтегрувати сторонні бібліотеки та плагіни, що розширює можливості гри та дозволяє використовувати готові рішення для типових завдань. Це дозволяє створювати розширювані та настроювані компоненти, що спрощує додавання нових функціональностей до гри без значних змін в існуючому коді.

Нарешті, C# має велику та активну спільноту розробників, що означає наявність численних ресурсів для навчання, прикладів коду, бібліотек і форумів для обміну досвідом і вирішення проблем. Існує багато документації, навчальних посібників і відеоуроків як по мові C#, так і по Unity, що значно полегшує процес навчання і розвитку проекту. Усі ці фактори роблять C# оптимальним вибором для створення сучасних та ефективних ігор в Unity.

Основні переваги мови C#:

1. Інтеграція з Unity

- Основна мова Unity: Unity використовує C# як основну мову програмування для розробки скриптів і компонентів гри. Це означає, що C# є найбільш оптимізованим і підтримуваним варіантом для роботи в Unity.

- Потужні API: Unity надає широкий набір API, написаних на C#, що дозволяє легко використовувати різні функції ігрового двигуна, такі як фізика, анімація, введення користувача і багато іншого.

2. Зручний та інтуїтивний синтаксис

- Зрозумілість: C# має простий і зрозумілий синтаксис, що робить його легким для вивчення і використання, навіть для новачків. Це дозволяє швидше приступити до розробки гри і зосередитись на її функціональності.

- Чистота коду: Завдяки строгій типізації та підтримці об’єктно-орієнтованого програмування, код на C# стає структурованим, читабельним і легко підтримуваним.

3. Об’єктно-орієнтоване програмування (ООП)

- Принципи ООП: C# підтримує всі основні принципи ООП (інкапсуляція, наслідування, поліморфізм), що дозволяє створювати модульний, повторно використовуваний та масштабований код.

- Компонентна архітектура: В Unity кожен ігровий об'єкт складається з компонентів, і C# ідеально підходить для реалізації такої архітектури, дозволяючи легко створювати і керувати компонентами.

4. Висока продуктивність

- Швидкодія: C# забезпечує високу продуктивність, що є критично важливим для реальних ігор, де кожен кадр має значення. Оптимізація роботи з пам'яттю та швидке виконання коду дозволяють створювати гладкі ігрові процеси.

- Багатопотоковість: Підтримка багатопотоковості в C# дозволяє ефективно використовувати ресурси багатоядерних процесорів, що є важливим для ресурсомістких завдань в іграх, таких як обробка фізики або штучного інтелекту.

5. Розширюваність і гнучкість

- Підтримка плагінів: Завдяки використанню C# можна легко інтегрувати сторонні бібліотеки та плагіни, що розширює можливості гри та дозволяє використовувати готові рішення для типових завдань.

- Розширювані компоненти: C# дозволяє створювати розширювані та настроювані компоненти, що спрощує додавання нових функціональностей до гри без значних змін в існуючому коді.

6. Широка підтримка та спільнота

- Активна спільнота: C# має велику та активну спільноту розробників, що означає наявність численних ресурсів для навчання, прикладів коду, бібліотек і форумів для обміну досвідом і вирішення проблем.

- Документація: Існує багато документації, навчальних посібників і відеоуроків як по мові C#, так і по Unity, що значно полегшує процес навчання і розвитку проекту.

3.2 Програмна розробка основних модулів додатку комп'ютерної гри

Основним модулем комп'ютерної гри у жанр, і "runner" з внутрішньоігровим магазином є клас Player, який відповідає за керування персонажем, та його взаємодію з ігровим світом [12].

Рух персонажа відбувається за рахунок кнопок на клавіатурі, або натисканням на екран смартфону. Реалізація руху відбувається у методах CheckInput, SlideButton, JumpButton, Jump, SetupMovement. Частини коду, що реалізовують рух гравця:

```
#region Inputs
public void SlideButton()
{
    if (isDead)
        return;

    if (rb.velocity.x != 0 && slideCooldownCounter < 0)
    {
        dustFx.Play();
        isSliding = true;
        slideTimeCounter = slideTime;
        slideCooldownCounter = slideCooldown;
    }
}

public void JumpButton()
{
    if (isSliding || isDead)
        return;

    RollAnimFinished();

    if (isGrounded)
    {
        Jump(jumpForce);
    }
    else if (canDoubleJump)
    {
```

```
            canDoubleJump = false;
            Jump(doubleJumpForce);
        }
    }

    private void Jump(float force)
    {
        dustFx.Play();
        AudioManager.instance.PlaySFX(Random.Range(1, 2));
        rb.velocity = new Vector2(rb.velocity.x, force);
    }

    private void CheckInput()
    {
        //if (Input.GetButtonDown("Fire2"))
        //    playerUnlocked = true;

        if (Input.GetButtonDown("Jump"))
            JumpButton();

        if (Input.GetKeyDown(KeyCode.LeftShift))
            SlideButton();
    }

    private void SetupMovement()
     {

         if (wallDetected)
         {
             SpeedReset();
             return;
         }

         if (isSliding)
             rb.velocity = new Vector2(slideSpeed, rb.velocity.y);
         else
             rb.velocity = new Vector2(moveSpeed, rb.velocity.y);
     }.
```

Смерть та отримана шкода персонажа відбувається після контакту персонажа з перешкодами, або падінням з рівня. Реалізовано програвання звуку після отримання шкоди, відкидання гравця, та анімацію крові [13]. Дані функції реалізовані у методах Damage, Die. Частини коду методів Damage, Die:

```
public void Damage()
 {
    bloodFx.Play();

    if (extraLife)
       Knockback();
    else
       StartCoroutine(Die());
 }

 private IEnumerator Die()
 {
    AudioManager.instance.PlaySFX(3);
    isDead = true;
    canBeKnocked = false;
    rb.velocity = knockbackDir;
    anim.SetBool("isDead", true);

    Time.timeScale = .6f;

    yield return new WaitForSeconds(1f);

    Time.timeScale = 1f;
    rb.velocity = new Vector2(0, 0);
    GameManager.instance.GameEnded();
 }.
```

Взаємодія головного персонажа з ігровим світом описана в методі CheckCollision. Метод перевіряє зо знаходиться перед гравцем, його стан, та місце розташування:

```
  private void CheckCollision()
  {
```

```
    isGrounded = Physics2D.Raycast(transform.position, Vector2.down,
groundCheckDistance, whatIsGround);
    ceillingDetected = Physics2D.Raycast(transform.position, Vector2.up,
ceillingCheckDistance, whatIsGround);
    wallDetected = Physics2D.BoxCast(wallCheck.position, wallCheckSize, 0,
Vector2.zero, 0, whatIsGround);
  }.
```

Важливою частиною гри є генерація платформи по яких рухається гравець. Методи GeneratePlatform та DeletePlatform відповідають за створення платформ перед гравцем, та видаленням платформ коли вони виходять за поле зору гравця, це потрібно для того, аби платфрми за гравцем не залишались та не завантажували пам’ять девайсу [13]. Наведено частини коду методів GeneratePlatform та DeletePlatform:

```
private void GeneratePlatform()
{
   while (Vector2.Distance(player.transform.position,nextPartPosition) <
distanceToSpawn)
   {
      Transform part = levelPart[Random.Range(0, levelPart.Length)];


      Vector2 newPosition = new Vector2(nextPartPosition.x -
part.Find("StartPoint").position.x, 0);

      Transform newPart = Instantiate(part, newPosition, transform.rotation, transform);

      nextPartPosition = newPart.Find("EndPoint").position;

   }
}

private void DeletePlatform()
{
   if (transform.childCount > 0)
   {
      Transform partToDelete = transform.GetChild(0);
```

```
        if (Vector2.Distance(player.transform.position, partToDelete.transform.position) >
distanceToDelete)
            Destroy(partToDelete.gameObject);

    }
}.
```

На платформах повинні з’являтися монетки, при взаємодії з якими у гравця збільшуватиметься кількість ігрової валюти, яку він в подальшому може витратити у внутрішньоігровому магазині. Метод генерації монеток знаходиться у методі Start класу CoinGeneration:

```
void Start()
{
    for (int i = 0; i < coinImg.Length; i++)
    {
        coinImg[i].sprite = null;
    }

    amountOfCoins = Random.Range(minCoins, maxCoins);
    int additionalOffset = amountOfCoins / 2;

    for (int i = 0; i < amountOfCoins; i++)
    {
        Vector3 offset = new Vector2(i - additionalOffset, 0);
        Instantiate(coinPrefab, transform.position + offset, Quaternion.identity,transform);
    }
}.
```

Основною функцією гри є внутрішньоігровий магазин, логіка якого реалізована у класі Shop, методах Start, PurchaseColor, EnoughMoney, Notify.

Наведено частини коду методів Start, PurchaseColor, EnoughMoney, Notify.

```
void Start()
{
    coinsText.text = PlayerPrefs.GetInt("Coins").ToString("#,#");

    for (int i = 0; i < platformColors.Length; i++)
```

```
    {
        Color color = platformColors[i].color;
        int price = platformColors[i].price;

        GameObject newButton = Instantiate(platformColorButton, platformColorParent);

        newButton.transform.GetChild(0).GetComponent<Image>().color = color;
        newButton.transform.GetChild(1).GetComponent<TextMeshProUGUI>().text =
price.ToString("#,#");

        newButton.GetComponent<Button>().onClick.AddListener(() =>
PurchaseColor(color, price, ColorType.platformColor));
    }

    for (int i = 0; i < playerColors.Length; i++)
    {
        Color color = playerColors[i].color;
        int price = playerColors[i].price;

        GameObject newButton = Instantiate(playerColorButton, playerColorParent);

        newButton.transform.GetChild(0).GetComponent<Image>().color = color;
        newButton.transform.GetChild(1).GetComponent<TextMeshProUGUI>().text =
price.ToString("#,#");    // price.ToString("#,#");;

        newButton.GetComponent<Button>().onClick.AddListener(() =>
PurchaseColor(color, price, ColorType.playerColor));
    }
}

public void PurchaseColor(Color color, int price, ColorType colorType)
{
    AudioManager.instance.PlaySFX(4);

    if (EnoughMoney(price))
    {
        if (colorType == ColorType.platformColor)
        {
            GameManager.instance.platformColor = color;
            platformDisplay.color = color;
```

```
        }
        else if (colorType == ColorType.playerColor)
        {
            GameManager.instance.player.GetComponent<SpriteRenderer>().color = color;
            GameManager.instance.SaveColor(color.r,color.g, color.b);
            playerDisplay.color = color;
        }

        StartCoroutine(Notify("Purchased successful", 1));
    }
    else
        StartCoroutine(Notify("Not enough money!", 1));

}

private bool EnoughMoney(int price)
{
    int myCoins = PlayerPrefs.GetInt("Coins");

    if (myCoins > price)
    {
        int newAmountOfCoins = myCoins - price;
        PlayerPrefs.SetInt("Coins", newAmountOfCoins);
        coinsText.text = PlayerPrefs.GetInt("Coins").ToString("#,#");
        return true;
    }
    return false;

}

IEnumerator Notify(string text, float seconds)
{
    notifyText.text = text;

    yield return new WaitForSeconds(seconds);

    notifyText.text = "Click to buy";
}.
```

Щоб забезпечити відчуття швидкості під час бігу персонажа було розроблено метод MoveBackground, аби будівлі на фоні поділені на шари, які рухаються з різною швидкістю відносно гравця. Наведено частини коду метода MoveBackground.

```
private void MoveBackground()
{
   float distanceMoved = cam.transform.position.x * (1 - parallaxEffect);
   float distanceToMove = cam.transform.position.x * parallaxEffect;

   transform.position = new Vector3(xPosition + distanceToMove, transform.position.y);

   if (distanceMoved > xPosition + length)
   {
      xPosition = xPosition + length;
   }
       }.
```

3.3 Розробка графічних матеріалів

На рівні з алгоритмами та механіками застосунку, графіка є основним компонентом взаємодії користувача з продуктом. Створення стильного, зручного та інтуїтивно зрозумілого оформлення відіграє вирішальну роль в ігровому досвіді, оскільки без цього, будь–який ігровий процес просто не матиме достатньої цікавості аби гравці поверталися у гру [11].

Розглянемо процес розробки графічних елементів для комп’ютерної гри жанру “runner” у середовищі Piskel на прикладі створення зображення головного героя.

На рисунку 3.1 наведено візуалізацію створення ігрового персонажа у середовищі Piskel. Було використано інструмент олівець, одне нажаття якого створює один піксель заданого кольору.

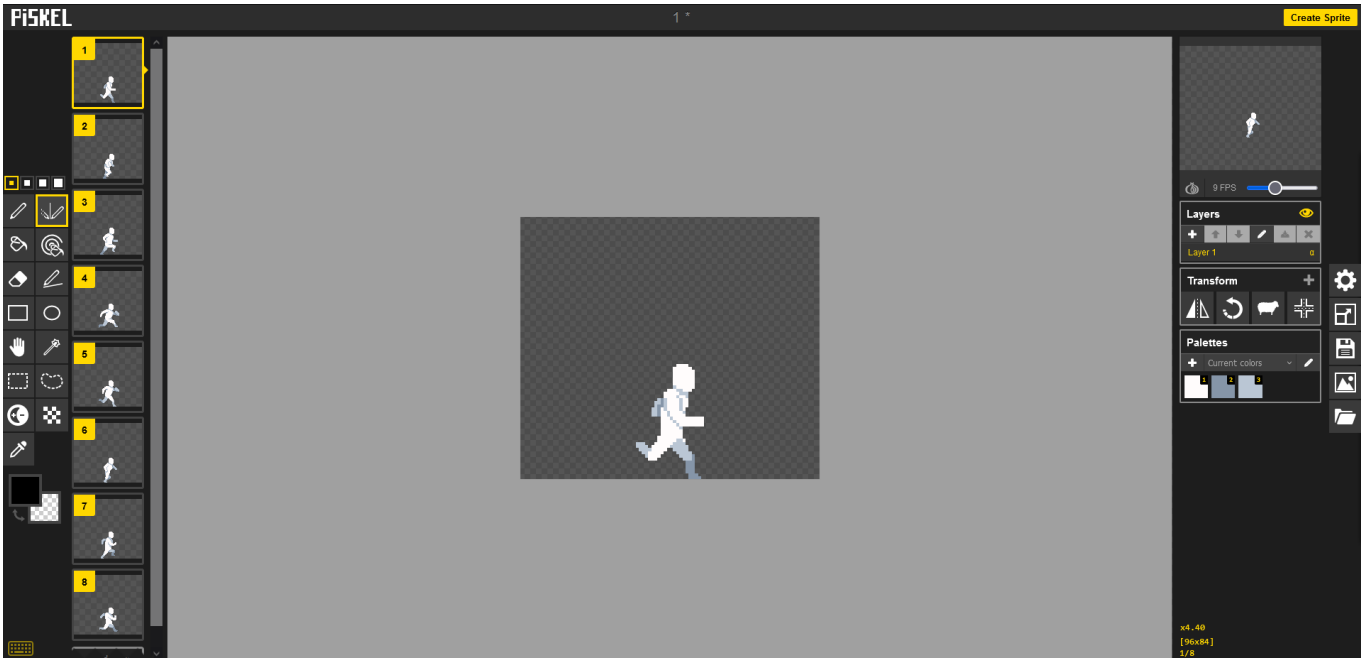

Рисунок 3.1 – Візуалізація створення ігрового персонажа у середовищі Piskel.

У середовищі графічного редагування ігрового двигуна Unity було створено фон. Щоб досягти паралакс [11] ефекту, було створено шари зображення. На рисунку 3.2 наведено зображення слоїв фону комп'ютерної гри.

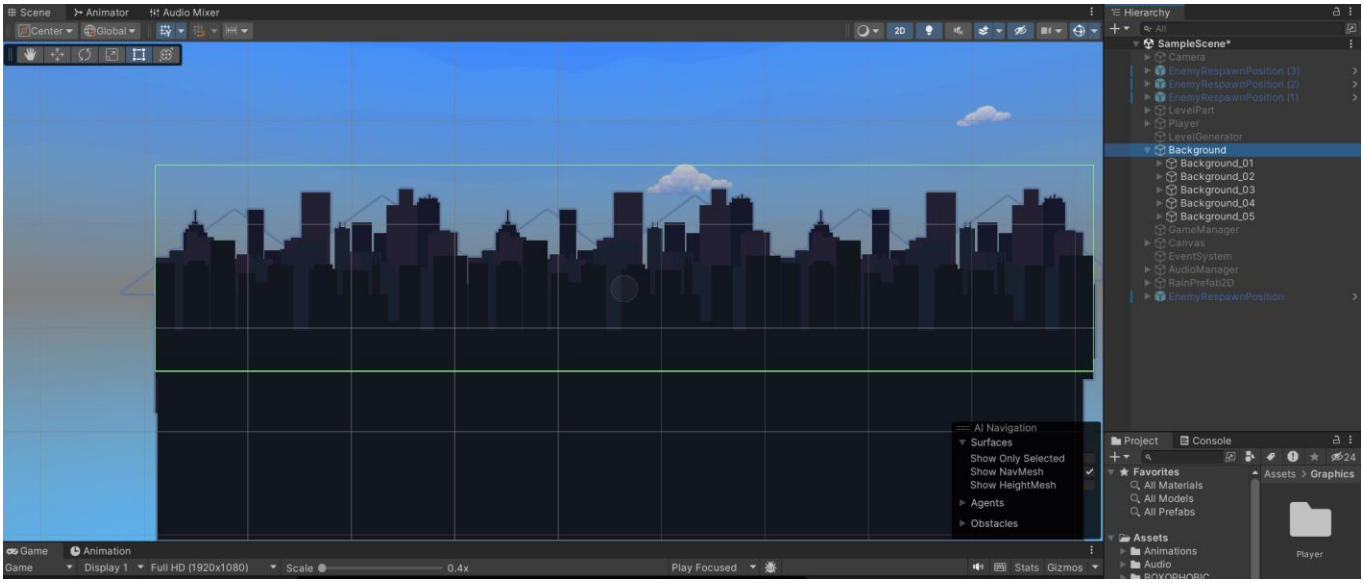

Рисунок 3.2 – Зображення шарів фону комп'ютерної гри.

Наступним графічним елементом який був створений – це зображення монеток, які гравець збиратиме під час гри. На рисунку 3.3 зображено моделі монеток, які були створені у графічному середовищі Piskel.

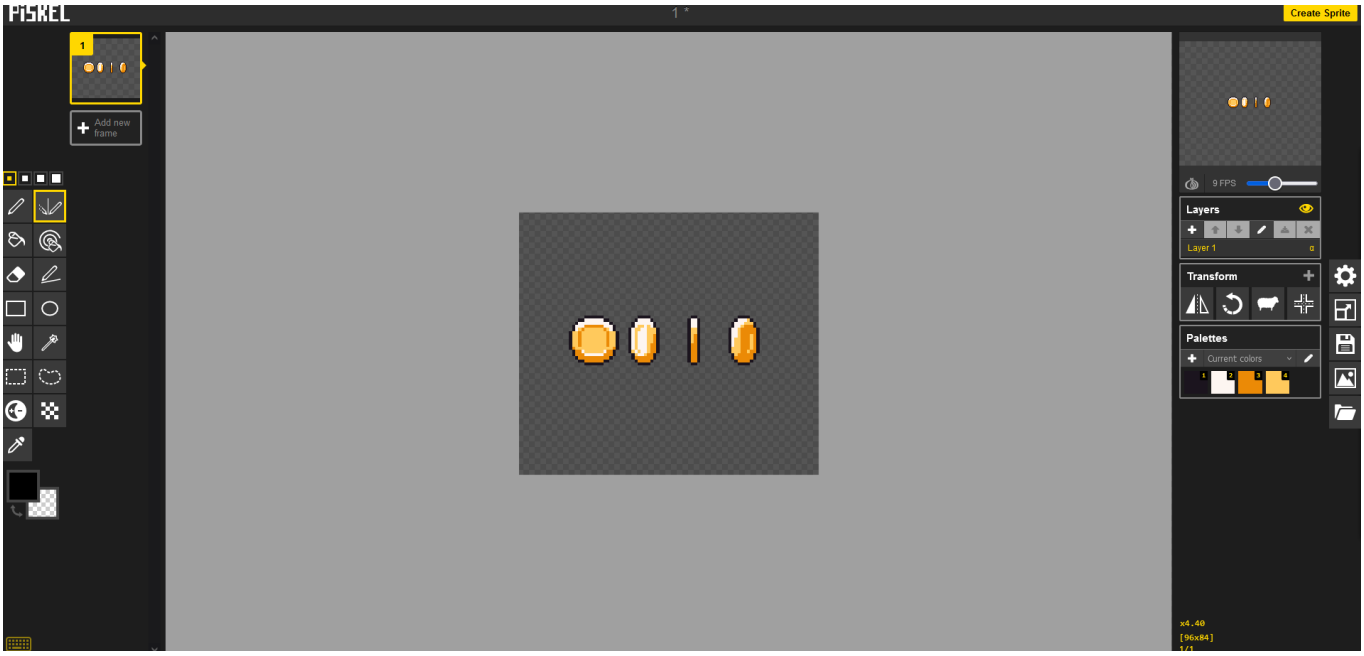

Рисунок 3.3 – Моделі монеток, які були створені у графічному середовищі Piskel.

На основі моделей користувацького інтерфейсу, які були спроектовані у розділі 2, було розроблено ігрові меню за допомогою інструментів ігрового двигуна Unity. На рисунку 3.4 зображено розроблене головне меню комп’ютерної гри [12].

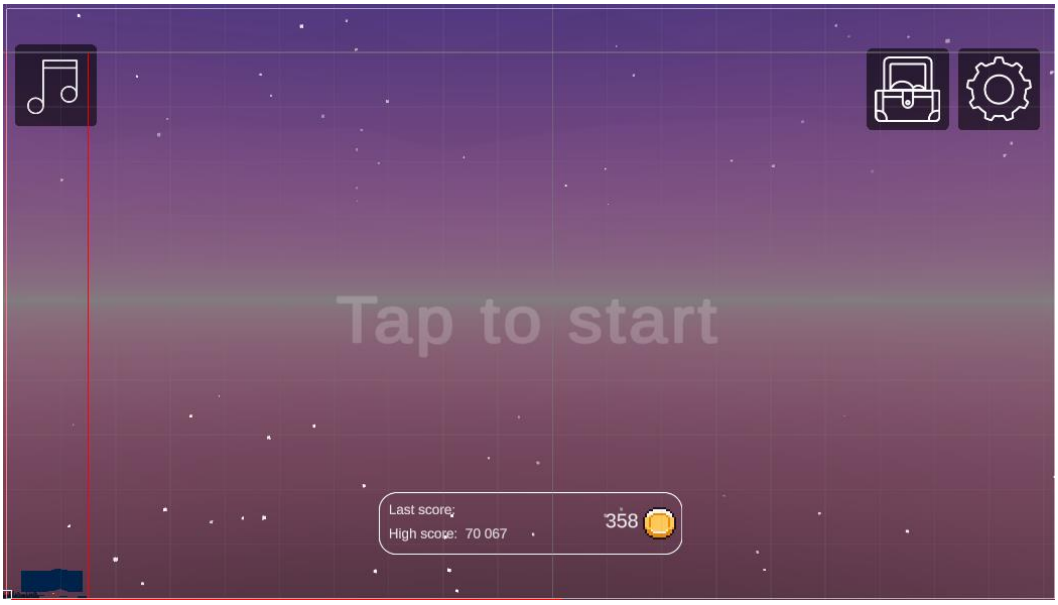

Рисунок 3.4 – Загальний вигляд інтерфейсного меню гри розроблене на основі спроектованих моделей

На рисунку 3.5 зображене меню налаштувань, розроблене на основі спроектованих у 2 розділі моделей.

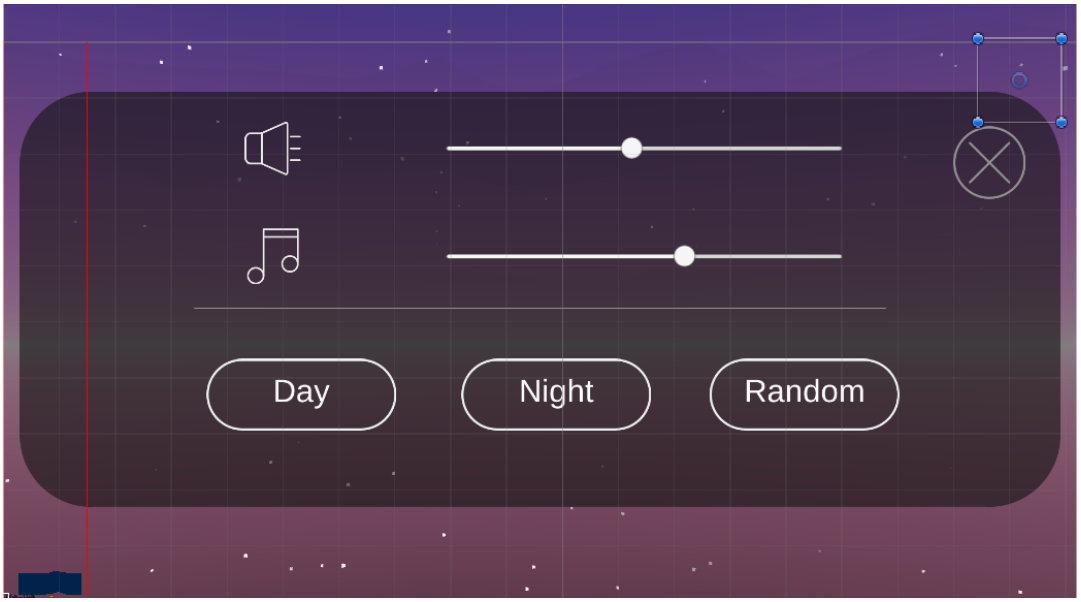

Рисунок 3.6 – Загальний вигляд інтерфейсного меню налаштувань розроблене на основі спроектованих моделей

На рисунку 3.7 зображене внутрішньоіговий магазин, розроблений на основі спроектованих у 2 розділі моделей.

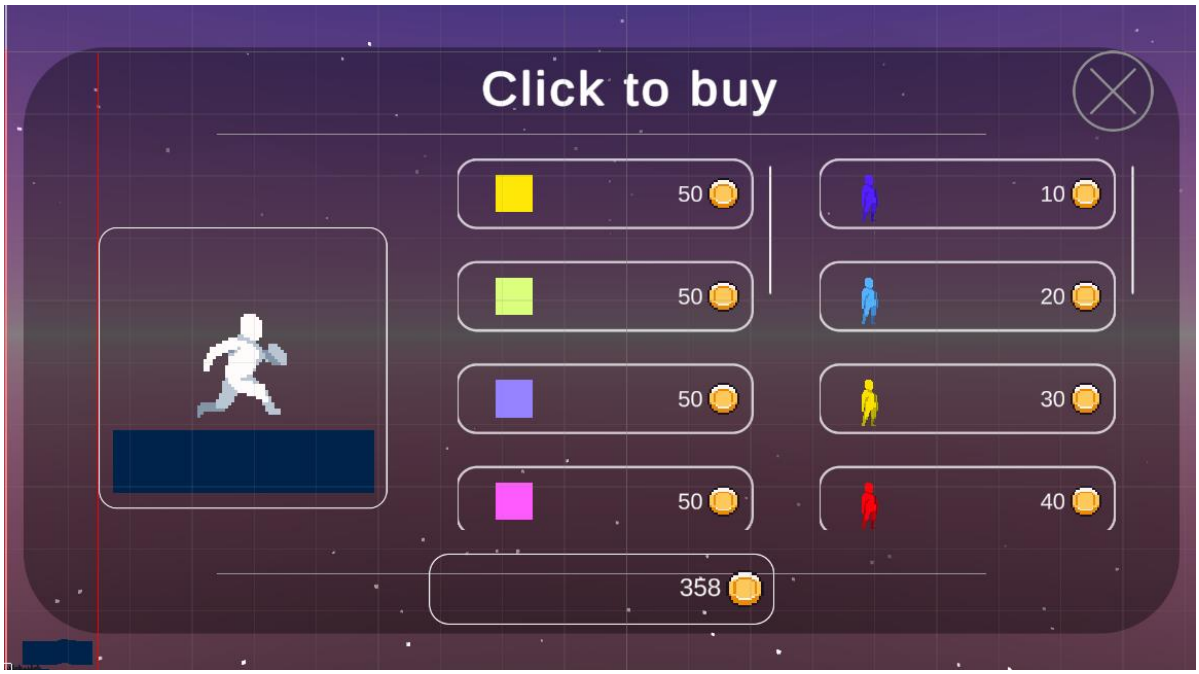

Рисунок 3.7 – Загальний вигляд інтерфейсного меню внутрішньоігрового магазина

На рисунку 3.8 зображене меню паузи, розроблене на основі спроектованих у 2 розділі моделей.

Рисунок 3.8 – Загальний вигляд інтерфейсного меню паузи

3.4 Результати тестування програмного модуля комп’ютерної гри

Першим кроком тестування комп’ютерної гри жанру “runner” внутрішньоігровим магазином є її запуск. Після запуску гравця зустрічає головне меню з кнопками «Почати гру» , «Налаштування», «Вимкнути музику» та панель найкращого результату та кількості валюти [13]. Результат запуску зображено на рисунку 3.18.

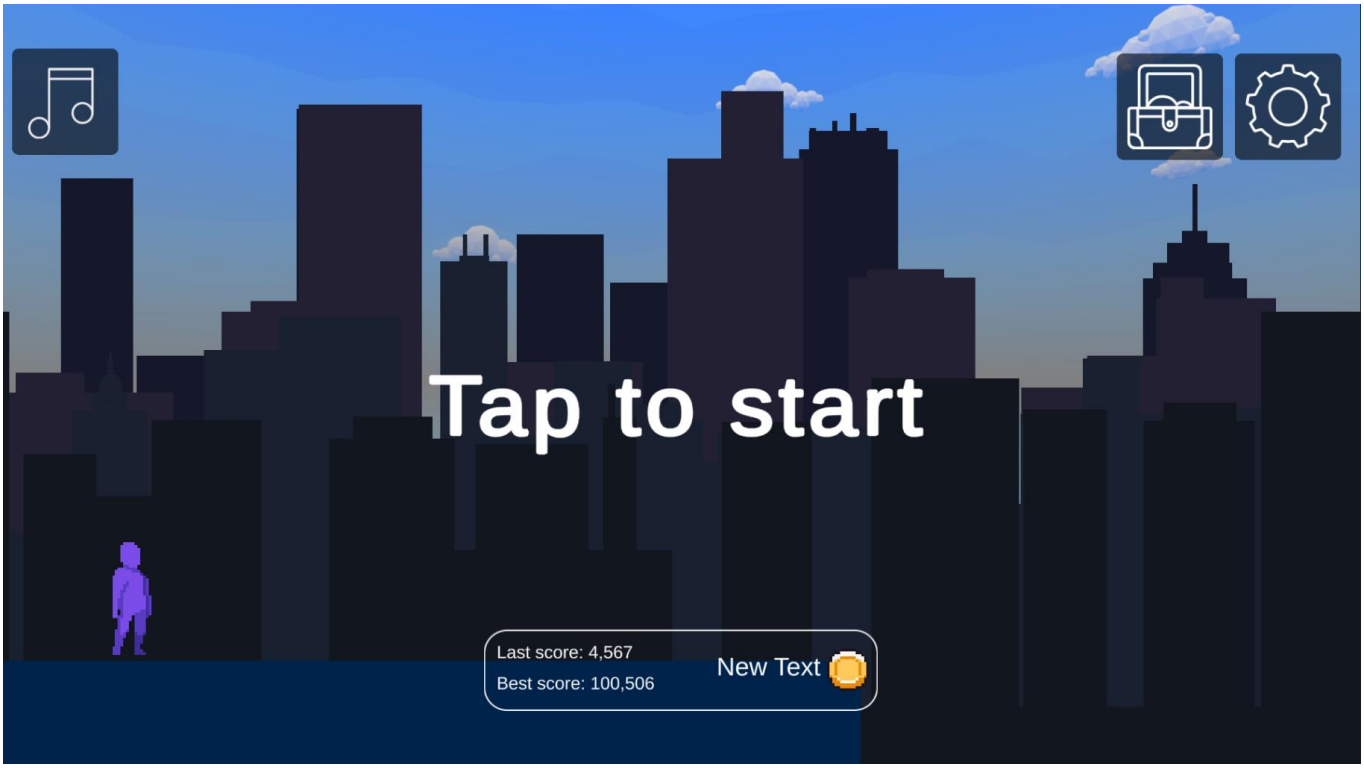

Рисунок 3.18 – Загальний вигляд інтерфейсного меню «Головне меню» комп’ютерної гри

Після запуску гри, розпочинається тестування геймплею. Для початку потрібно натиснути в будь яке місце на екрані, аби розпочалась гра, результатом чого буде меню ігрового процесу, яке включає в себе кнопку паузи, кнопку вимкнення музики та панель де відображається зароблена валюта, дистанцію яку пробіг герой та доступність підкату та наявність додаткових життів [14]. Результат початку гри зображено на рисунку 3.19

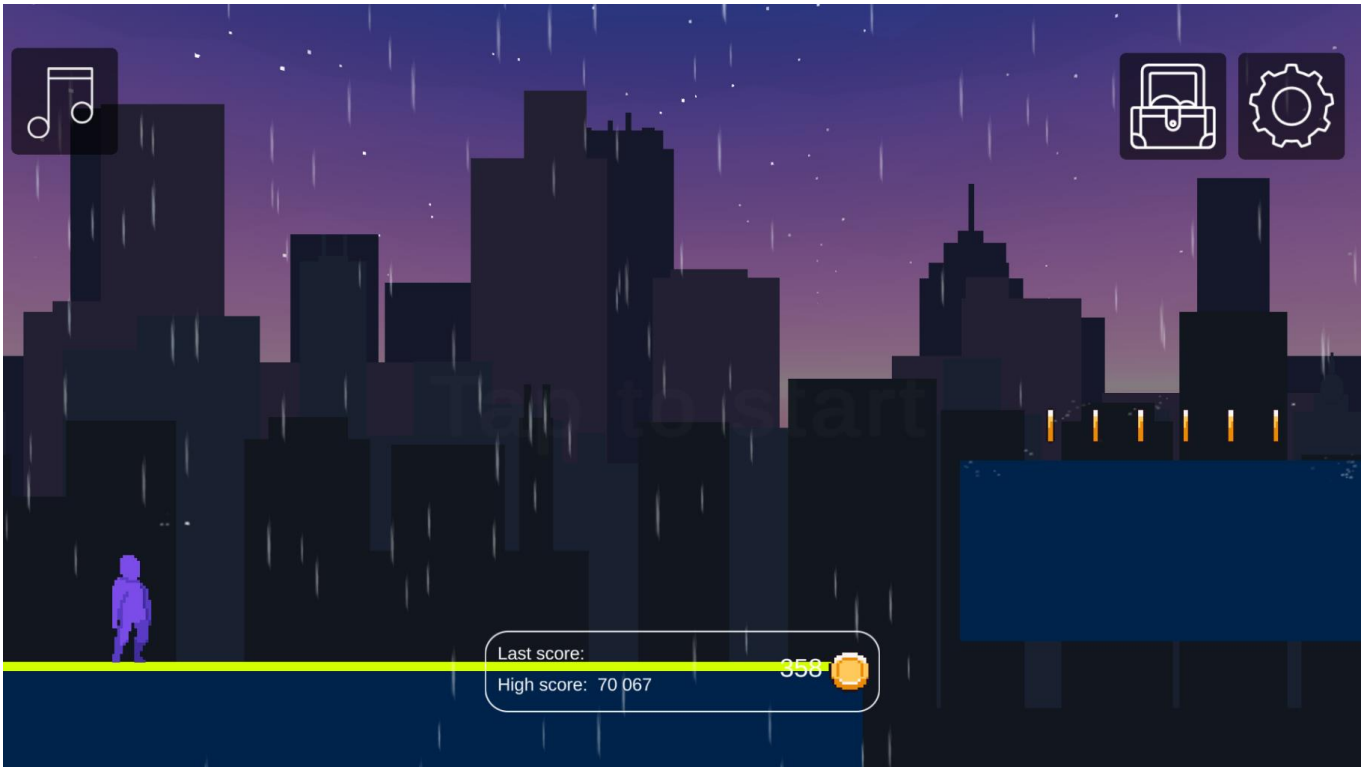

Рисунок 3.19 – Загальний вигляд інтерфейсу початку гри

Подальшим кроком буде тестування руху персонажа. Аби почати рух персонажа потрібно натиснути на екран, після початку бігу наступне натискання буде призводити до стрибка, також після натискання на кнопку підкати або кнопку Shift на клавіатурі [15]. Результат натискання на кнопки руху персонажа зображено на рисунках 3.20 – 3.23.

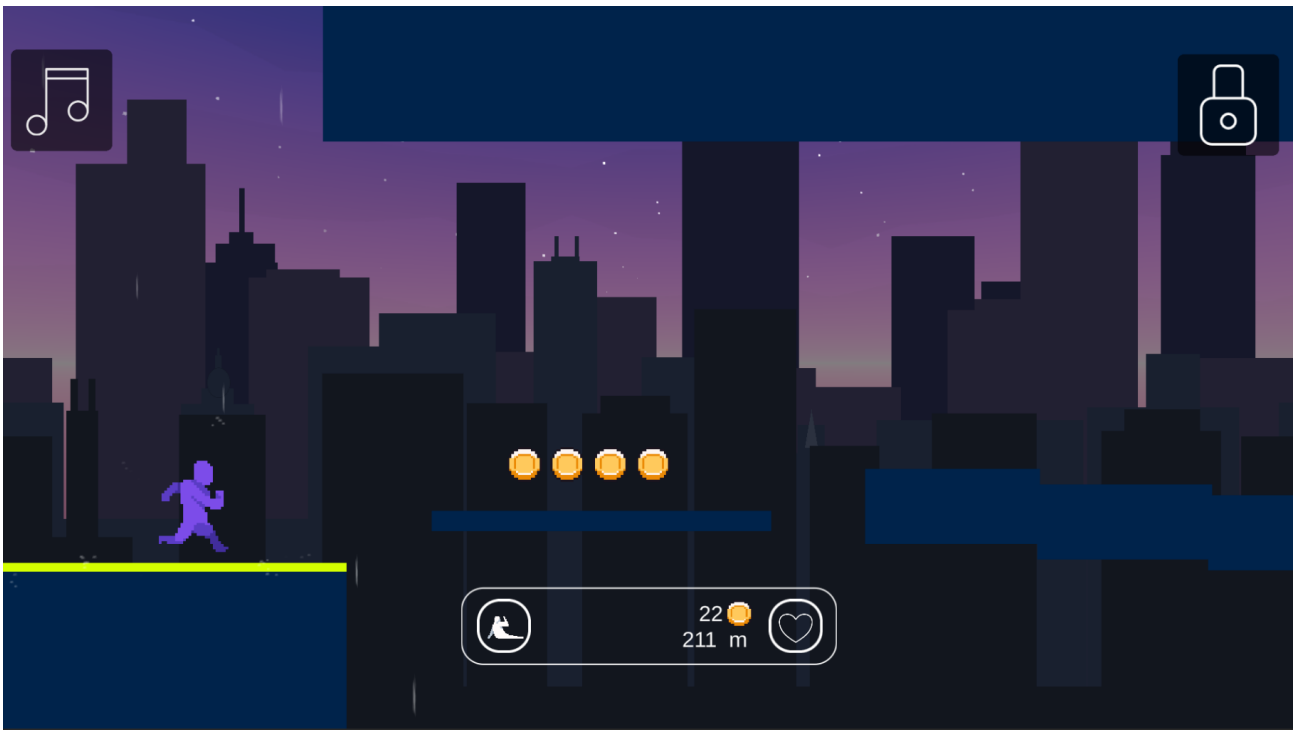

Рисунок 3.20 – Загальний вигляд інтерфейсу після натискання на екран і початку бігу

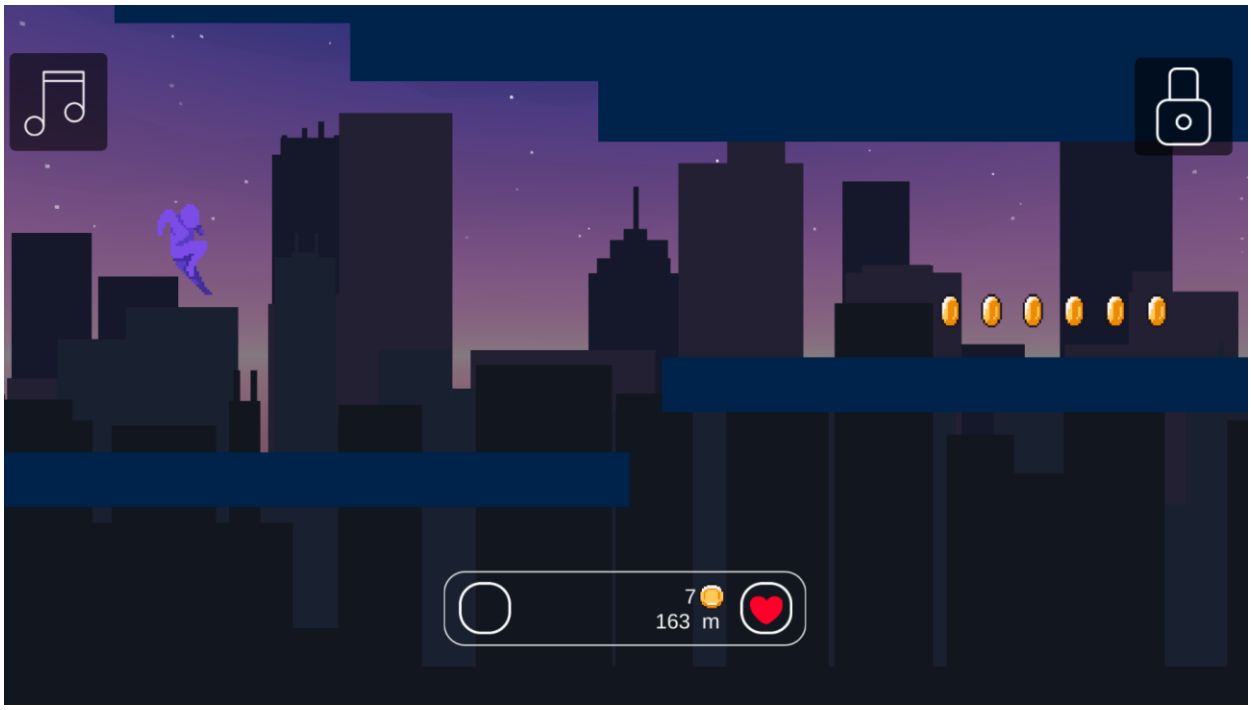

Рисунок 3.21 – Загальний вигляд інтерфейсу після натискання кнопки стрибка

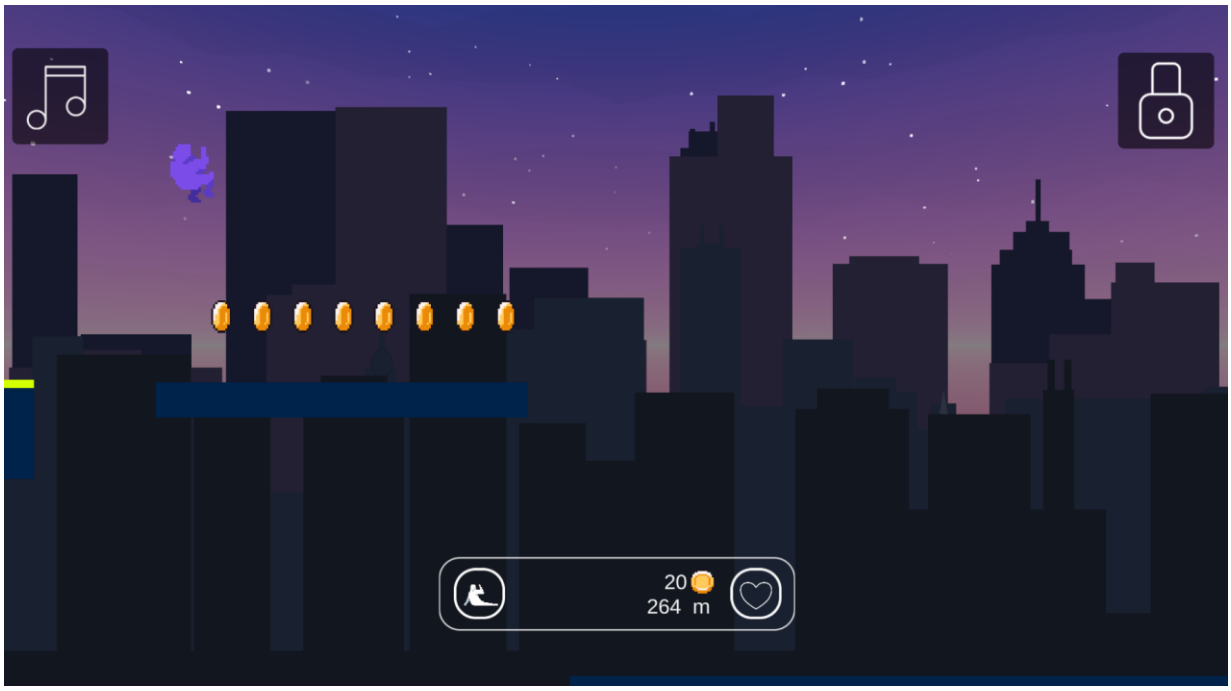

Рисунок 3.22 – Загальний вигляд інтерфейсу після натискання кнопки подвійного стрибка

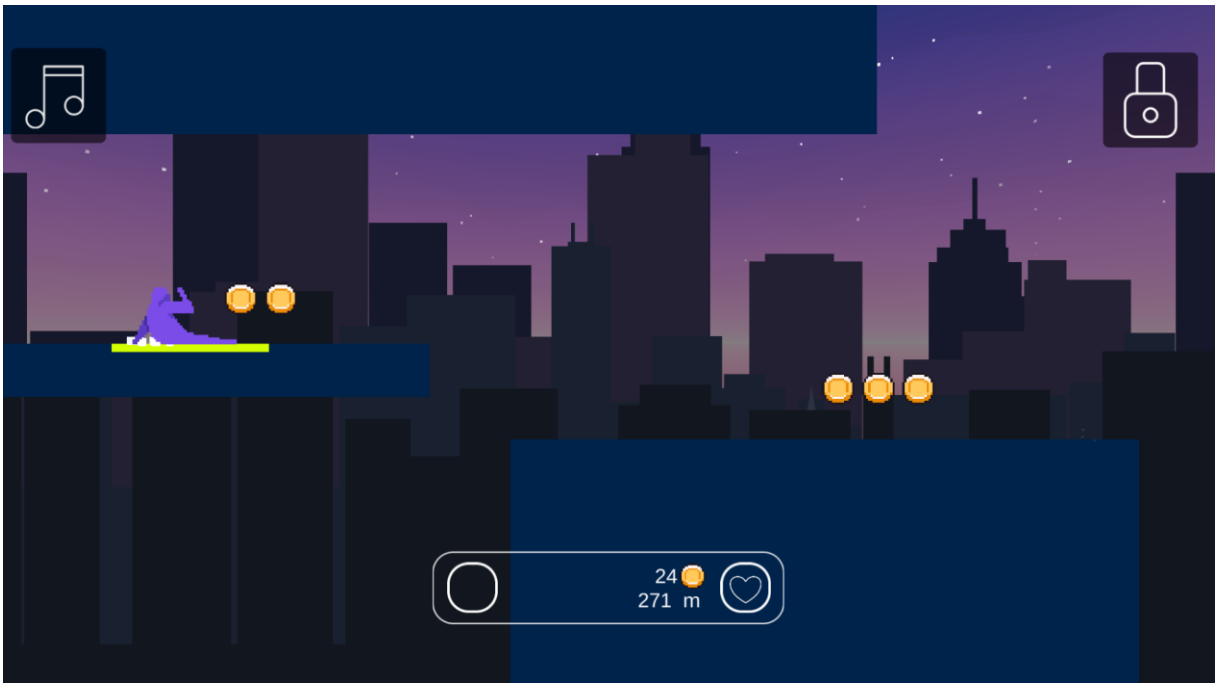

Рисунок 3.23 – Загальний вигляд інтерфейсу після натискання кнопки підкату

Гравець має змогу взаємодіяти з перешкодами, якщо він не вчасно оминає їх, то це призводить до відкидування гравця, аз умови що він має додаткове життя, або ж до смерті в інакшому випадку [15]. На рисунках 3.24 – 3.25 зображено відкидання персонажа і втрата додаткового життя, та момент смерті персонажа.

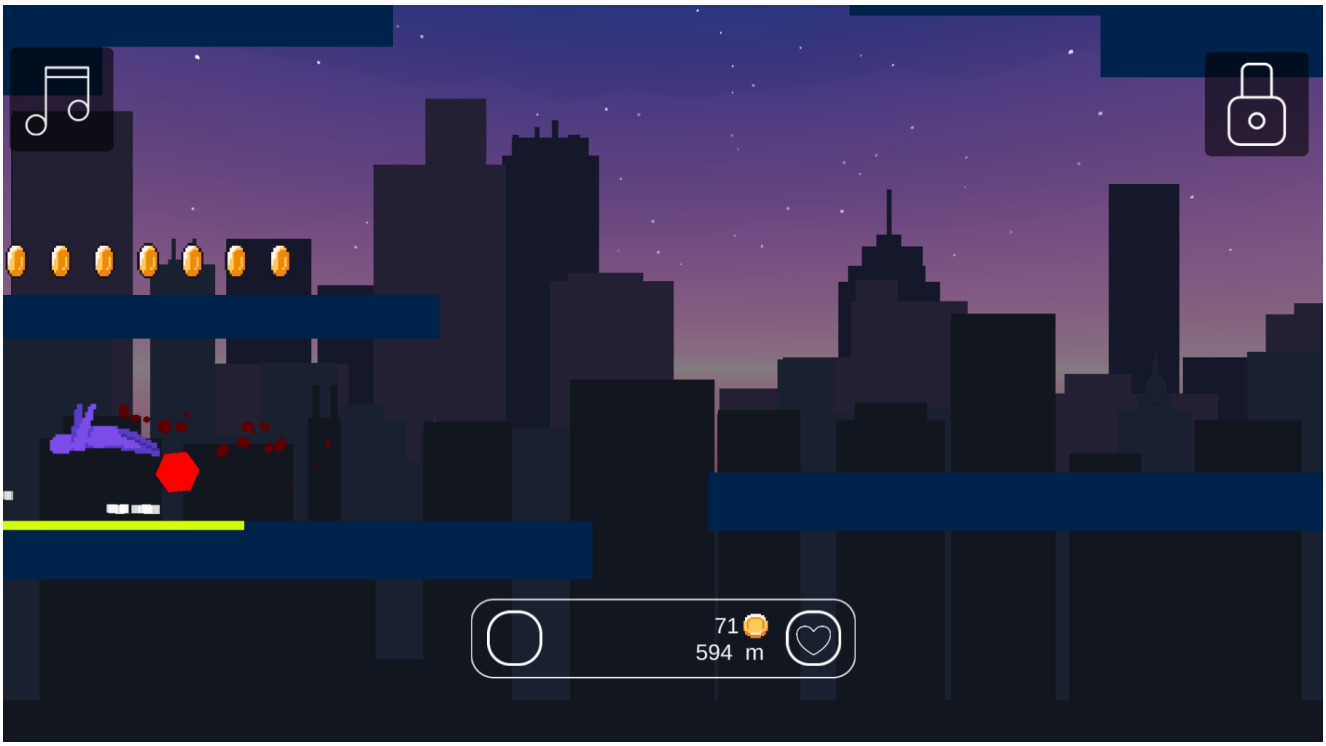

Рисунок 3.24 – Загальний вигляд інтерфейсу після смерті персонажа

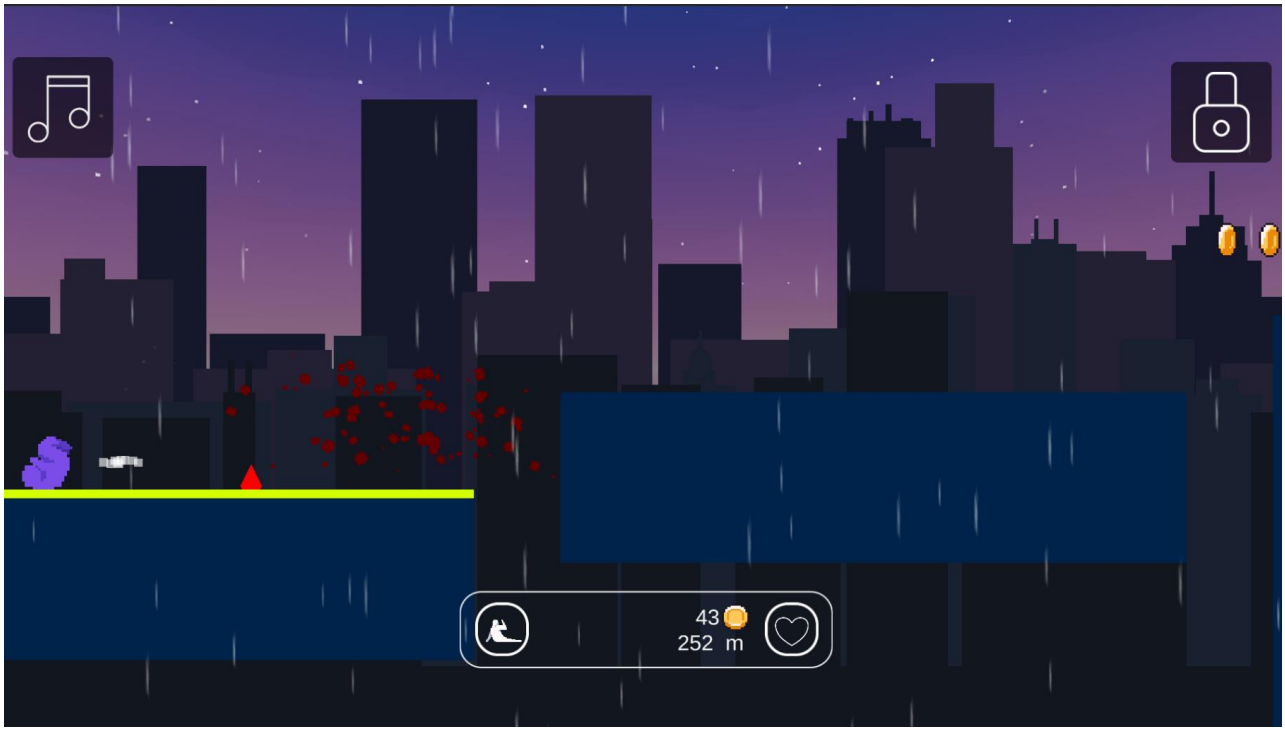

Рисунок 3.25 – Загальний вигляд інтерфейсу після відкидання персонажа та втрати додаткового життя

Також необхідно перевірити роботу паузи. Для цього натиснемо на кнопку «Пауза» у правому верхньому куті. Відкрите меню паузи зображено на рисунку 3.26

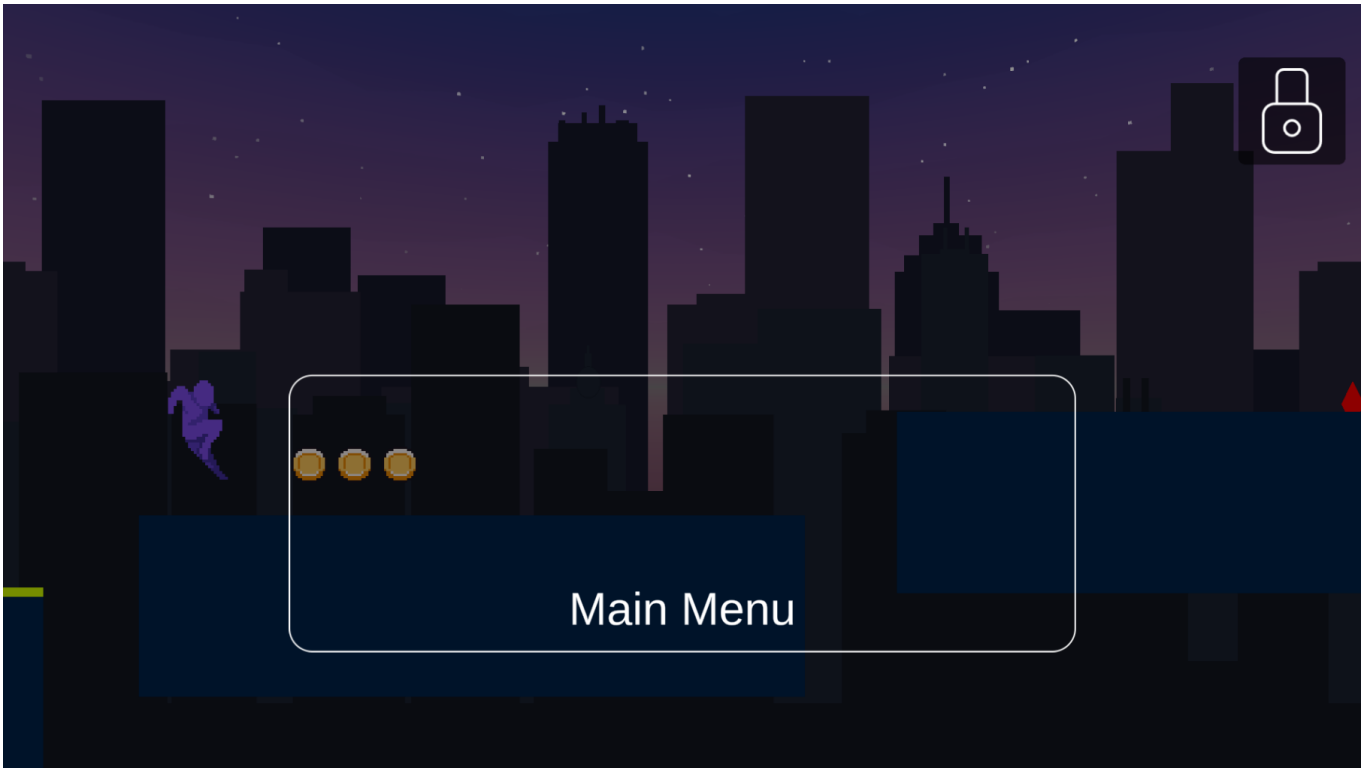

Рисунок 3.26 – Загальний вигляд інтерфейсного меню «Пауза»

Головною частиною гри є внутрішньоігровий магазин. Аби протестувати його, у головному меню потрібно натиснути на кнопку магазину, після чого відкриється вікно внутрішньоігрового магазину, де наприклад змінимо колір персонажу з фіолетового на зелений, зміну персонажу зображено на рисунках 3.26–3.27

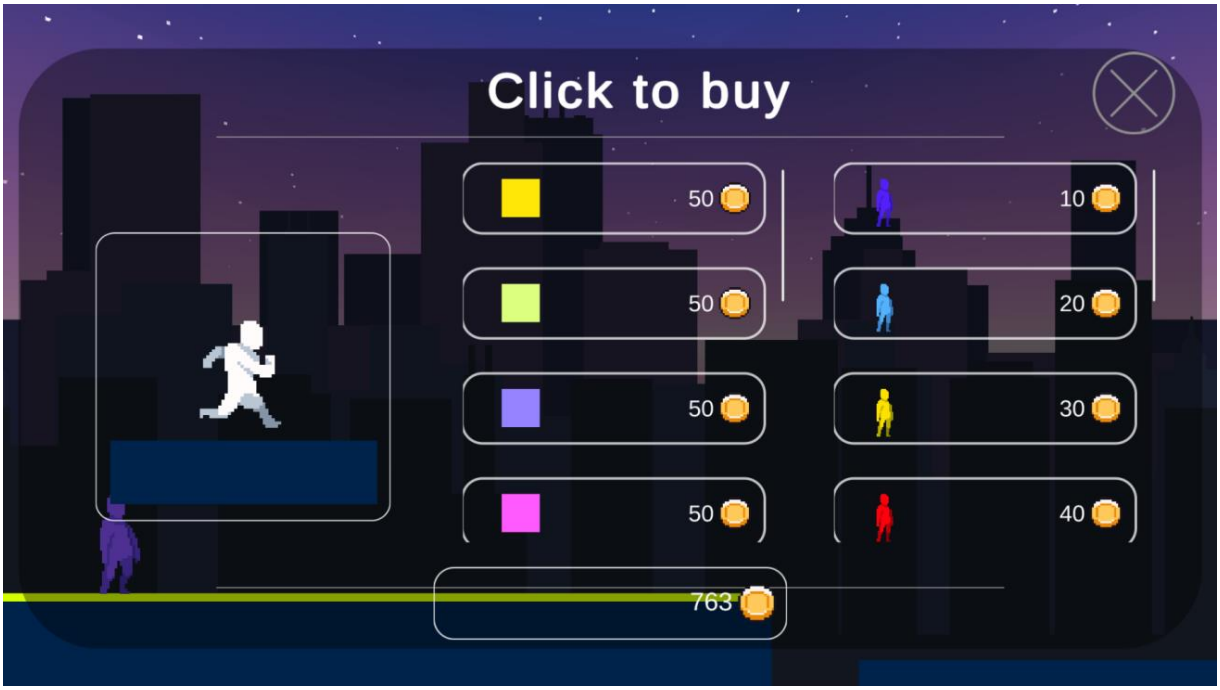

Рисунок 3.26 – Загальний вигляд інтерфейсного меню «Магазин» до покупки кастомізації персонажа

Рисунок 3.27 – Загальний вигляд інтерфейсного меню «Магазин» після покупки кастомізації

Протестуємо також меню налаштувань, для цього у головному меню натиснемо на кнопку «Налаштування» та наприклад змінимо налаштування рівня з ночі на день. Зміну зображено на рисунках 3.28 – 3.29

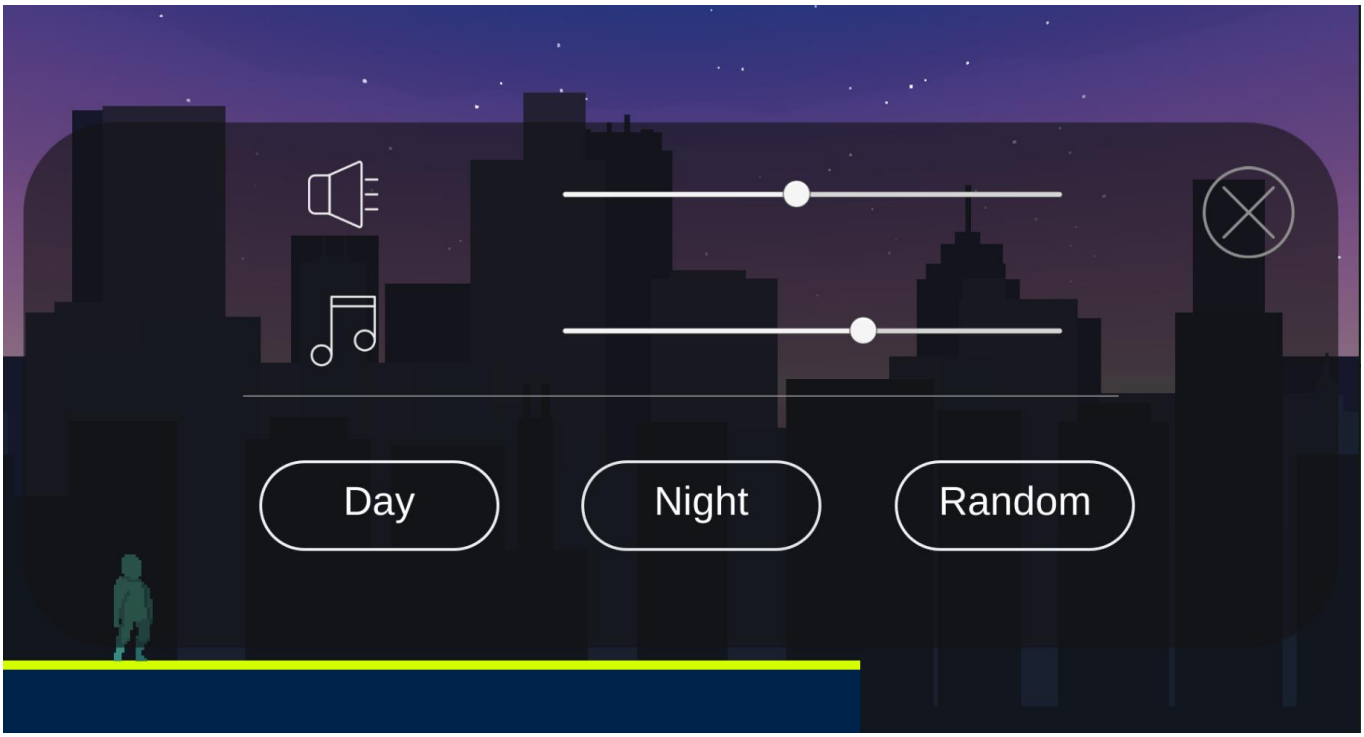

Рисунок 3.28 – Загальний вигляд інтерфейсного меню «Налаштування» до зміни налаштувань

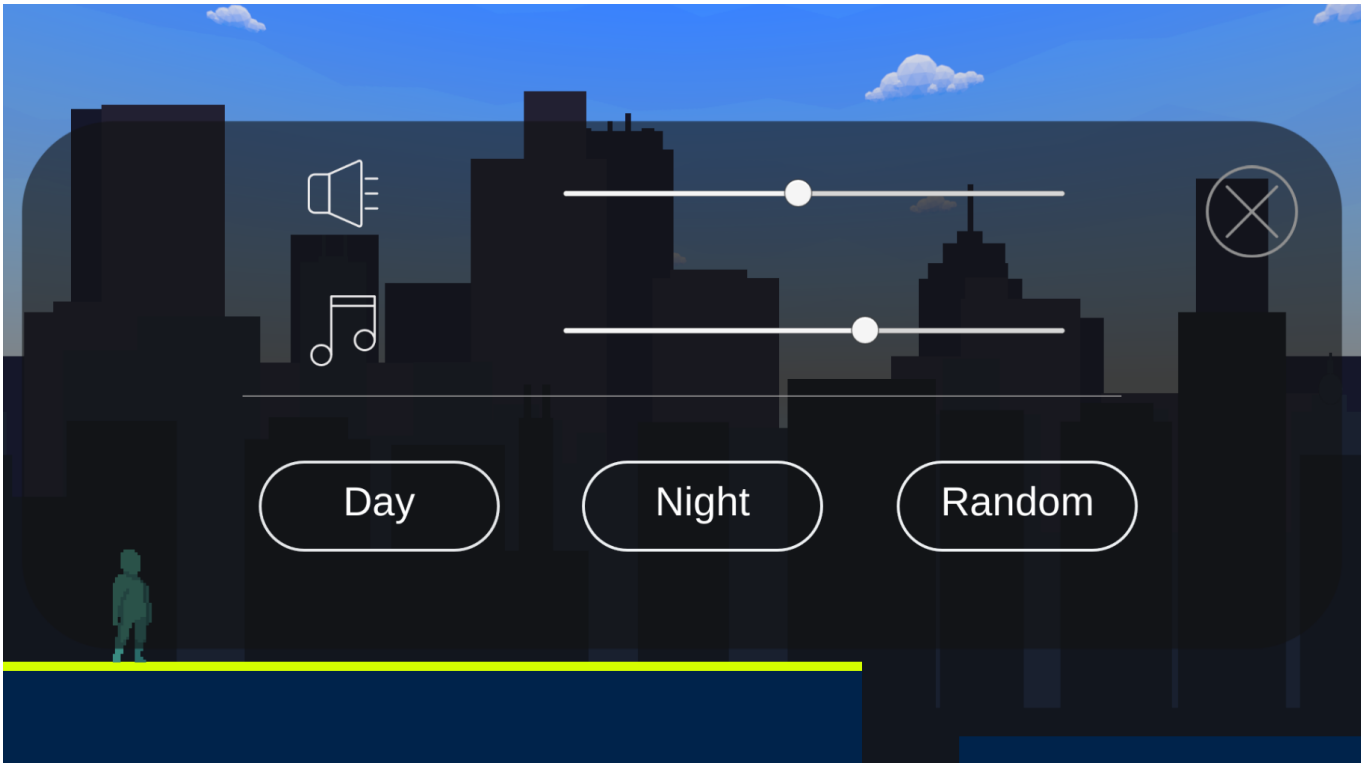

Рисунок 3.28 – Загальний вигляд інтерфейсного меню «Налаштування» після зміни налаштувань

За результатами тестування комп’ютерної гри жанру “runner” з внутрішньоігровим магазином підтверджено працездатність усіх функцій.

3.5 Висновки до розділу 3

У третьому розділі дипломної роботи було детально розглянуто процес розробки та тестування 2D гри у жанрі раннер. Цей розділ охоплює ключові етапи, які включають концептуалізацію, проектування, реалізацію та тестування гри, що дозволило створити якісний кінцевий продукт.

На етапі проектування була проведена детальна робота з вибору інструментів та технологій, необхідних для розробки гри. Було обрано ігровий рушій, який найкраще підходив для реалізації 2D графіки та забезпечував легку інтеграцію з

іншими програмними засобами. Вибір рушія визначався його функціональними можливостями, гнучкістю та зручністю використання. Окрім цього, для написання коду гри було обрано мову програмування, яка забезпечувала оптимальну продуктивність та легкість у навчанні і використанні [16].

Реалізація гри включала створення основних ігрових механік. Було розроблено систему руху персонажів, яка дозволяла гравцям легко керувати своїм персонажем у грі. Також було впроваджено систему взаємодії з об'єктами, що включала зіткнення з перешкодами, збирання бонусів та інші елементи, що роблять гру цікавою та динамічною. Окрім цього, розроблена система балів та рівнів забезпечувала мотивацію гравців до проходження гри та підвищувала реіграбельність.

Ретельне тестування забезпечило високий рівень стабільності гри та її готовність до випуску. Завдяки систематичному підходу до тестування, було виявлено та виправлено більшість помилок на ранніх етапах, що дозволило уникнути серйозних проблем на фінальному етапі розробки. Це, в свою чергу, забезпечило високий рівень готовності продукту до комерційного використання [16].

ВИСНОВКИ

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

– Проведено аналіз предметної області, методів та засобів розробки комп'ютерних ігор жанру “runner”.

– Розроблено структуру програмного забезпечення для гри жанру “runner”.

– Створено базу даних для зберігання інформації про гравців та їхній прогрес.

– Розроблено алгоритм роботи основних ігрових механік.

– Виконано програмну реалізацію гри жанру “runner”.

– Створено інструкцію користувача.

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

Також було проведено тестування розробленої гри.

Мета дослідження – створення захопливої та функціональної гри жанру раннер, досягнута завдяки реалізації ефективних ігрових механік, включаючи динамічний біг, збирання бонусів, уникнення перешкод та поступове ускладнення рівнів. Крім того, було розроблено можливості для зберігання прогресу гравця.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Schell, J. The Art of Game Design: A Book of Lenses. URL: https://www.crcpress.com/The–Art–of–Game–Design–A–Book–of–Lenses–Third–Edition/Schell/p/book/9781138632059. (дата звернення: 04.06.2024).

2. Макогончук Д.А., Іванчук Я. В. "Розробка комп'ютерної гри у жанрі "runner" з внутрішньоігровим магазином" у Матеріалах конференції «Всеукраїнська науково-технічна конференція факультету інтелектуальних інформаційних технологій та автоматизації», Вінниця, 2024. URL: https://conferences.vntu.edu.ua/index.php/all-fksa/all-fksa-2024/paper/view/20964. (дата звернення: 04.06.2024).

3. Adams, E., Rollings, A. Fundamentals of Game Design. URL: https://www.pearson.com/store/p/fundamentals–of–game–design/P100000617674. (дата звернення: 04.06.2024).

4. Novak, J. Game Development Essentials: An Introduction. URL: https://www.cengage.com/c/game–development–essentials–an–introduction–3e–novak/9781111307653. (дата звернення: 04.06.2024).

5. McGuire, M., Jenkins, O. Creating Games: Mechanics, Content, and Technology. URL: https://www.taylorandfrancis.com/books/mono/10.1201/9781439874888/creating–games–michael–mcguire. (дата звернення: 04.06.2024).

6. Nystrom, R. Game Programming Patterns. URL: https://gameprogrammingpatterns.com/. (дата звернення: 04.06.2024).

7. Rogers, S. Level Up! The Guide to Great Video Game Design. URL: https://www.wiley.com/en–us/Level+Up%21%3A+The+Guide+to+Great+Video+Game+Design–p–9781118877167. (дата звернення: 04.06.2024).

8. Totten, C. W. An Architectural Approach to Level Design. URL: https://www.crcpress.com/An–Architectural–Approach–to–Level–Design/Totten/p/book/9781498707215. (дата звернення: 04.06.2024).

9. Salen, K., Zimmerman, E. Rules of Play: Game Design Fundamentals. URL: https://mitpress.mit.edu/books/rules–play. (дата звернення: 04.06.2024).

10. Fullerton, T. Game Design Workshop: A Playcentric Approach to Creating Innovative Games. URL: https://www.crcpress.com/Game–Design–Workshop–A–Playcentric–Approach–to–Creating–Innovative–Games/Fullerton/p/book/9781138098770. (дата звернення: 04.06.2024).

11. Rabin, S. Introduction to Game Development. URL: https://www.charlesriver.com/titles/introduction–to–game–development/9781584503794. (дата звернення: 04.06.2024).

12. Rollings, A., Adams, E. Andrew Rollings and Ernest Adams on Game Design. URL: https://www.peachpit.com/store/andrew–rollings–and–ernest–adams–on–game–design–9781592730018. (дата звернення: 04.06.2024).

13. Rogers, S. Swipe This!: The Guide to Great Touchscreen Game Design. URL: https://www.wiley.com/en–us/Swipe+This%21%3A+The+Guide+to+Great+Touchscreen+Game+Design–p–9781118852386. (дата звернення: 04.06.2024).

14. Isbister, K. How Games Move Us: Emotion by Design. URL: https://mitpress.mit.edu/books/how–games–move–us. (дата звернення: 04.06.2024).

15. Hunicke, R., LeBlanc, M., Zubek, R. MDA: A Formal Approach to Game Design and Game Research. URL: https://www.cs.northwestern.edu/~hunicke/MDA.pdf. (дата звернення: 04.06.2024).

16. Jason, G. Developing 2D Games with Unity: Independent, [Електронний ресурс]. URL: https://www.springer.com/gp/book/9781484237716. (дата звернення: 04.06.2024).

ДОДАТОК А

ПРОТОКОЛ ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ НА НАЯВНІСТЬ ТЕКСТОВИХ ЗАПОЗИЧЕНЬ

Назва роботи: Комп'ютерна гра жанру runner із внутрішньо-ігровим магазином

Тип роботи: бакалаврська дипломна робота
(БДР, МКР)

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

Показники звіту подібності Unicheck

Оригінальність 98,2% Схожість 1,8%

Аналіз звіту подібності (відмітити потрібне):

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

Особа, відповідальна за перевірку Озеранський В.С.

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

Автор роботи Макогончук Д.А.

Керівник роботи Іванчук Я.В.

ДОДАТОК Б

Інструкція користувача

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

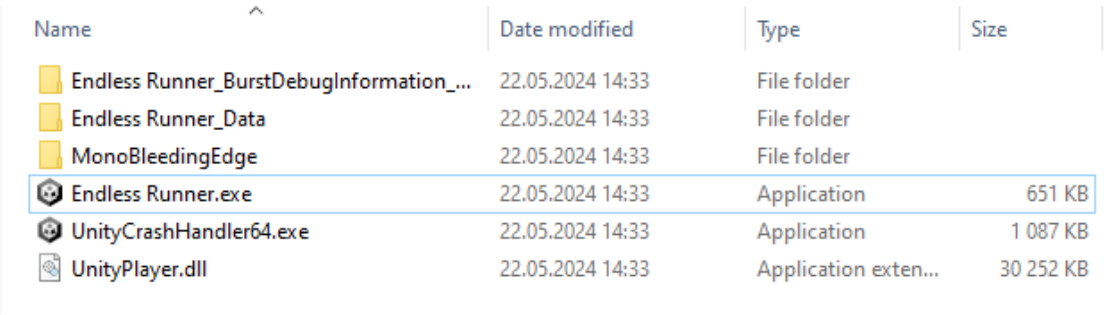

Рисунок Б.1 – Зображення файла запуску

3. Переглядаємо головне меню гри, за готовності починаємо гру

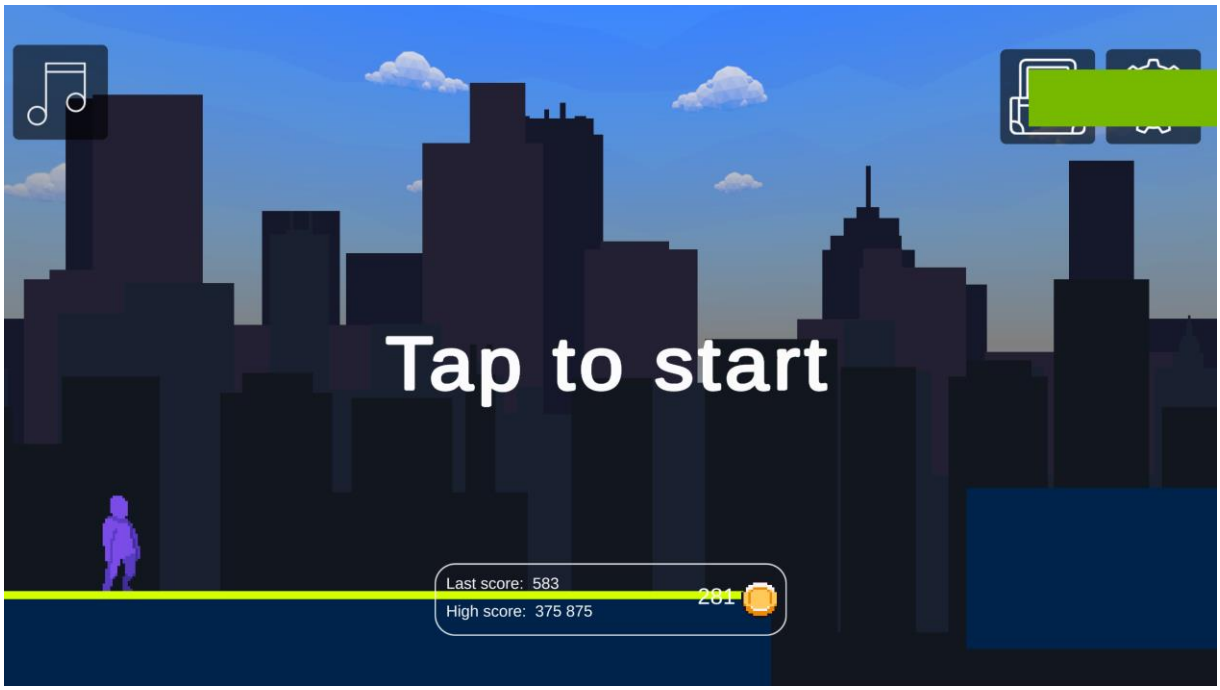

Рисунок Б.2 – Загальний вигляд меню програмного модуля

4. Після натискання на екран почніть гру

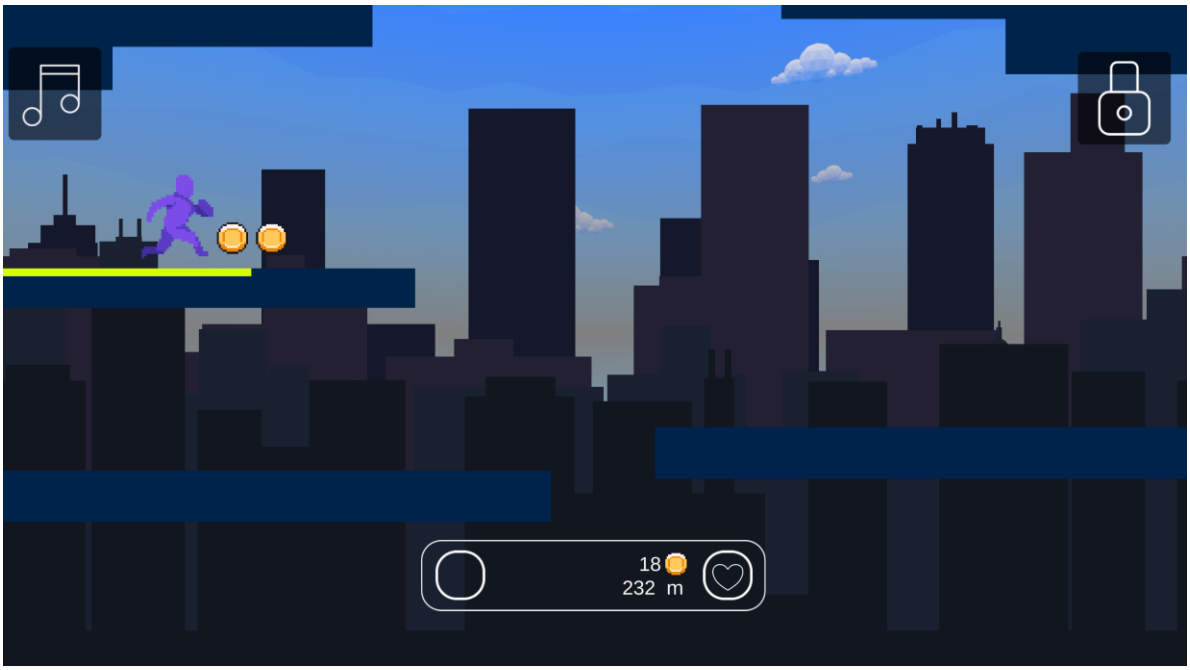

Рисунок Б.3 – Загальний вигляд програмного модуля

ДОДАТОК В

Лістинг програми

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

public class Player : MonoBehaviour
{

    private Rigidbody2D rb;
    private Animator anim;
    private SpriteRenderer sr;

    private bool isDead;
    [HideInInspector] public bool playerUnlocked;
    [HideInInspector] public bool extraLife;

    [Header("VFX")]
    [SerializeField] private ParticleSystem dustFx;
    [SerializeField] private ParticleSystem bloodFx;

    [Header("Knockback info")]
    [SerializeField] private Vector2 knockbackDir;
    private bool isKnocked;
    private bool canBeKnocked = true;

    [Header("Move info")]
    [SerializeField] private float speedToSurvive = 18;
    [SerializeField] private float moveSpeed;
    [SerializeField] private float maxSpeed;
    [SerializeField] private float speedMultiplier;
    private float defaultSpeed;
    [Space]
    [SerializeField] private float milestoneIncreaser;
    private float defaultMilestoneIncrease;
    private float speedMilestone;

    private bool readyToLand;

    [Header("Jump info")]
    [SerializeField] private float jumpForce;
    [SerializeField] private float doubleJumpForce;
    private bool canDoubleJump;



    [Header("Slide info")]
```

```
[SerializeField] private float slideSpeed;
[SerializeField] private float slideTime;
[SerializeField] private float slideCooldown;
[HideInInspector] public float slideCooldownCounter;
private float slideTimeCounter;
private bool isSliding;

[Header("Collision info")]
[SerializeField] private float groundCheckDistance;
[SerializeField] private float ceillingCheckDistance;
[SerializeField] private LayerMask whatIsGround;
[SerializeField] private Transform wallCheck;
[SerializeField] private Vector2 wallCheckSize;
private bool isGrounded;
private bool wallDetected;
private bool ceillingDetected;
[HideInInspector] public bool ledgeDetected;

[Header("Ledge info")]
[SerializeField] private Vector2 offset1; // offset for position before climb
[SerializeField] private Vector2 offset2; // offset for position AFTER climb

private Vector2 climbBegunPosition;
private Vector2 climbOverPosition;

private bool canGrabLedge = true;
private bool canClimb;




void Start()
{
    rb = GetComponent<Rigidbody2D>();
    anim = GetComponent<Animator>();
    sr = GetComponent<SpriteRenderer>();



    speedMilestone = milestoneIncreaser;
    defaultSpeed = moveSpeed;
    defaultMilestoneIncrease = milestoneIncreaser;
}

void Update()
{
    CheckCollision();
    AnimatorControllers();

    slideTimeCounter -= Time.deltaTime;
    slideCooldownCounter -= Time.deltaTime;
```

```
    extraLife = moveSpeed >= speedToSurvive;

    if (isDead)
        return;

    if (isKnocked)
        return;

    if (playerUnlocked)
        SetupMovement();

    if (isGrounded)
    {
        Time.timeScale = 1;
        canDoubleJump = true;
    }

    SpeedController();

    CheckForLanding();
    CheckForLedge();
    CheckForSlideCancel();
    CheckInput();


}

private void CheckForLanding()
{
    if (rb.velocity.y < –5 && !isGrounded)
        readyToLand = true;

    if (readyToLand && isGrounded)
    {
        dustFx.Play();
        readyToLand = false;
    }
}

public void Damage()
{
    bloodFx.Play();

    if (extraLife)
        Knockback();
    else
        StartCoroutine(Die());
}

private IEnumerator Die()
```

```
{
    AudioManager.instance.PlaySFX(3);
    isDead = true;
    canBeKnocked = false;
    rb.velocity = knockbackDir;
    anim.SetBool("isDead", true);

    Time.timeScale = .6f;

    yield return new WaitForSeconds(1f);

    Time.timeScale = 1f;
    rb.velocity = new Vector2(0, 0);
    GameManager.instance.GameEnded();
}


#region Knockback
private IEnumerator Invincibility()
{
    Color originalColor = sr.color;
    Color darkenColor = new Color(sr.color.r, sr.color.g, sr.color.b, .5f);

    canBeKnocked = false;
    sr.color = darkenColor;
    yield return new WaitForSeconds(.1f);

    sr.color = originalColor;
    yield return new WaitForSeconds(.1f);

    sr.color = darkenColor;
    yield return new WaitForSeconds(.15f);

    sr.color = originalColor;
    yield return new WaitForSeconds(.15f);

    sr.color = darkenColor;
    yield return new WaitForSeconds(.25f);

    sr.color = originalColor;
    yield return new WaitForSeconds(.25f);

    sr.color = darkenColor;
    yield return new WaitForSeconds(.3f);

    sr.color = originalColor;
    yield return new WaitForSeconds(.35f);

    sr.color = darkenColor;
    yield return new WaitForSeconds(.4f);
```

```
    sr.color = originalColor;
    canBeKnocked = true;
}

private void Knockback()
{
    if (!canBeKnocked)
        return;

    SpeedReset();
    StartCoroutine(Invincibility());
    isKnocked = true;
    rb.velocity = knockbackDir;
}

private void CancelKnockback() => isKnocked = false;

#endregion

#region SpeedControll
private void SpeedReset()
{
    if (isSliding)
        return;

    moveSpeed = defaultSpeed;
    milestoneIncreaser = defaultMilestoneIncrease;
}

private void SpeedController()
{
    if (moveSpeed == maxSpeed)
        return;


    if (transform.position.x > speedMilestone)
    {
        speedMilestone = speedMilestone + milestoneIncreaser;

        moveSpeed = moveSpeed * speedMultiplier;
        milestoneIncreaser = milestoneIncreaser * speedMultiplier;

        if (moveSpeed > maxSpeed)
            moveSpeed = maxSpeed;
    }
}
#endregion

#region Ledge Climb Region

private void CheckForLedge()
```

```
{
    if (ledgeDetected && canGrabLedge)
    {
        canGrabLedge = false;
        rb.gravityScale = 0;

        Vector2 ledgePosition = GetComponentInChildren<LedgeDetection>().transform.position;

        climbBegunPosition = ledgePosition + offset1;
        climbOverPosition = ledgePosition + offset2;

        canClimb = true;
    }

    if (canClimb)
        transform.position = climbBegunPosition;
}

private void LedgeClimbOver()
{

    canClimb = false;
    rb.gravityScale = 5;
    transform.position = climbOverPosition;
    Invoke("AllowLedgeGrab", .1f);
}

private void AllowLedgeGrab() => canGrabLedge = true;


#endregion

private void CheckForSlideCancel()
{
    if (slideTimeCounter < 0 && !ceillingDetected)
        isSliding = false;
}
private void SetupMovement()
{

    if (wallDetected)
    {
        SpeedReset();
        return;
    }

    if (isSliding)
        rb.velocity = new Vector2(slideSpeed, rb.velocity.y);
    else
        rb.velocity = new Vector2(moveSpeed, rb.velocity.y);
}
```

```
#region Inputs
public void SlideButton()
{
    if (isDead)
        return;

    if (rb.velocity.x != 0 && slideCooldownCounter < 0)
    {
        dustFx.Play();
        isSliding = true;
        slideTimeCounter = slideTime;
        slideCooldownCounter = slideCooldown;
    }
}

public void JumpButton()
{
    if (isSliding || isDead)
        return;

    RollAnimFinished();

    if (isGrounded)
    {
        Jump(jumpForce);
    }
    else if (canDoubleJump)
    {
        canDoubleJump = false;
        Jump(doubleJumpForce);
    }
}

private void Jump(float force)
{
    dustFx.Play();
    AudioManager.instance.PlaySFX(Random.Range(1, 2));
    rb.velocity = new Vector2(rb.velocity.x, force);
}

private void CheckInput()
{
    //if (Input.GetButtonDown("Fire2"))
    //    playerUnlocked = true;

    if (Input.GetButtonDown("Jump"))
        JumpButton();

    if (Input.GetKeyDown(KeyCode.LeftShift))
```

```
            SlideButton();
        }
        #endregion
        #region Animations
        private void AnimatorControllers()
        {
            anim.SetFloat("xVelocity", rb.velocity.x);
            anim.SetFloat("yVelocity", rb.velocity.y);

            anim.SetBool("canDoubleJump", canDoubleJump);
            anim.SetBool("isGrounded", isGrounded);
            anim.SetBool("isSliding", isSliding);
            anim.SetBool("canClimb", canClimb);
            anim.SetBool("isKnocked", isKnocked);

            if (rb.velocity.y < –20)
                anim.SetBool("canRoll", true);

        }

        private void RollAnimFinished() => anim.SetBool("canRoll", false);

        #endregion

        private void CheckCollision()
        {
            isGrounded = Physics2D.Raycast(transform.position, Vector2.down, groundCheckDistance, whatIsGround);
            ceillingDetected = Physics2D.Raycast(transform.position, Vector2.up, ceillingCheckDistance,
whatIsGround);
            wallDetected = Physics2D.BoxCast(wallCheck.position, wallCheckSize, 0, Vector2.zero, 0, whatIsGround);
        }
        private void OnDrawGizmos()
        {
            Gizmos.DrawLine(transform.position, new Vector2(transform.position.x, transform.position.y –
groundCheckDistance));
            Gizmos.DrawLine(transform.position, new Vector2(transform.position.x, transform.position.y +
ceillingCheckDistance));
            Gizmos.DrawWireCube(wallCheck.position, wallCheckSize);
        }
}


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

public class AudioManager : MonoBehaviour
{
    public static AudioManager instance;

    [SerializeField] private AudioSource[] sfx;
```

```
    [SerializeField] private AudioSource[] bgm;

    private void Awake() => instance = this;


    private int bgmIndex;
    private void Update()
    {
        if (!bgm[bgmIndex].isPlaying)
            PlayRandomBGM();
    }
    public void PlayRandomBGM()
    {
        bgmIndex = Random.Range(0, bgm.Length);

        PlayBGM(bgmIndex);
    }

    public void PlaySFX(int index)
    {
        if (index < sfx.Length)
        {
            sfx[index].pitch = Random.Range(.85f, 1.1f);
            sfx[index].Play();
        }
    }

    public void StopSFX(int index)
    {
        sfx[index].Stop();
    }

    public void PlayBGM(int index)
    {
        for (int i = 0; i < bgm.Length; i++)
        {
            bgm[index].Stop();
        }

        bgm[index].Play();
    }

    public void StopBGM()
    {
        for (int i = 0; i < bgm.Length; i++)
        {
            bgm[i].Stop();
        }
    }

}
```

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

public class Coin : MonoBehaviour
{
    private void OnTriggerEnter2D(Collider2D collision)
    {
        if (collision.GetComponent<Enemy>() != null)
            Destroy(gameObject);


        if (collision.GetComponent<Player>() != null)
        {
            AudioManager.instance.PlaySFX(0);
            GameManager.instance.coins++;
            Destroy(gameObject);
        }
    }
}


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

public class CoinGenerator : MonoBehaviour
{
    private int amountOfCoins;
    [SerializeField] private GameObject coinPrefab;
    [SerializeField] private int minCoins;
    [SerializeField] private int maxCoins;

    [SerializeField] private SpriteRenderer[] coinImg;

    void Start()
    {
        for (int i = 0; i < coinImg.Length; i++)
        {
            coinImg[i].sprite = null;
        }

        amountOfCoins = Random.Range(minCoins, maxCoins);
        int additionalOffset = amountOfCoins / 2;

        for (int i = 0; i < amountOfCoins; i++)
        {
            Vector3 offset = new Vector2(i – additionalOffset, 0);
            Instantiate(coinPrefab, transform.position + offset, Quaternion.identity,transform);
```

ДОДАТОК Г

Графічна частина

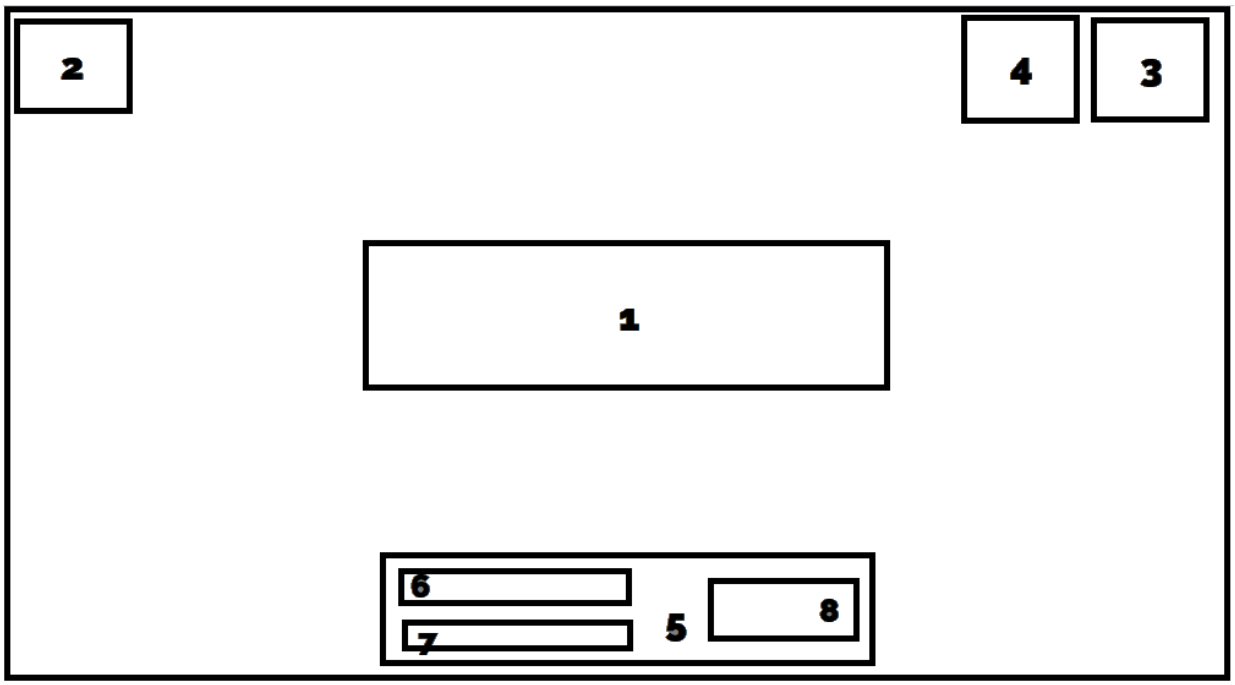

Рисунок Г.1 – Конструктивна схема головного меню

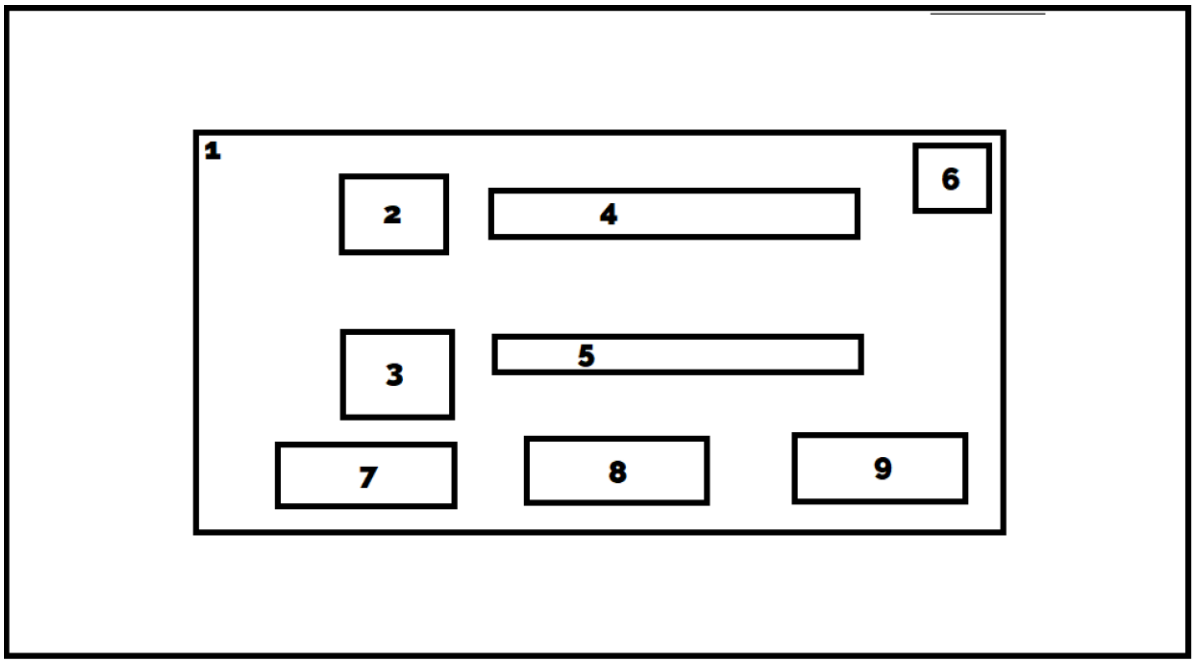

Рисунок Г.2 – Конструктивна схема меню налаштувань

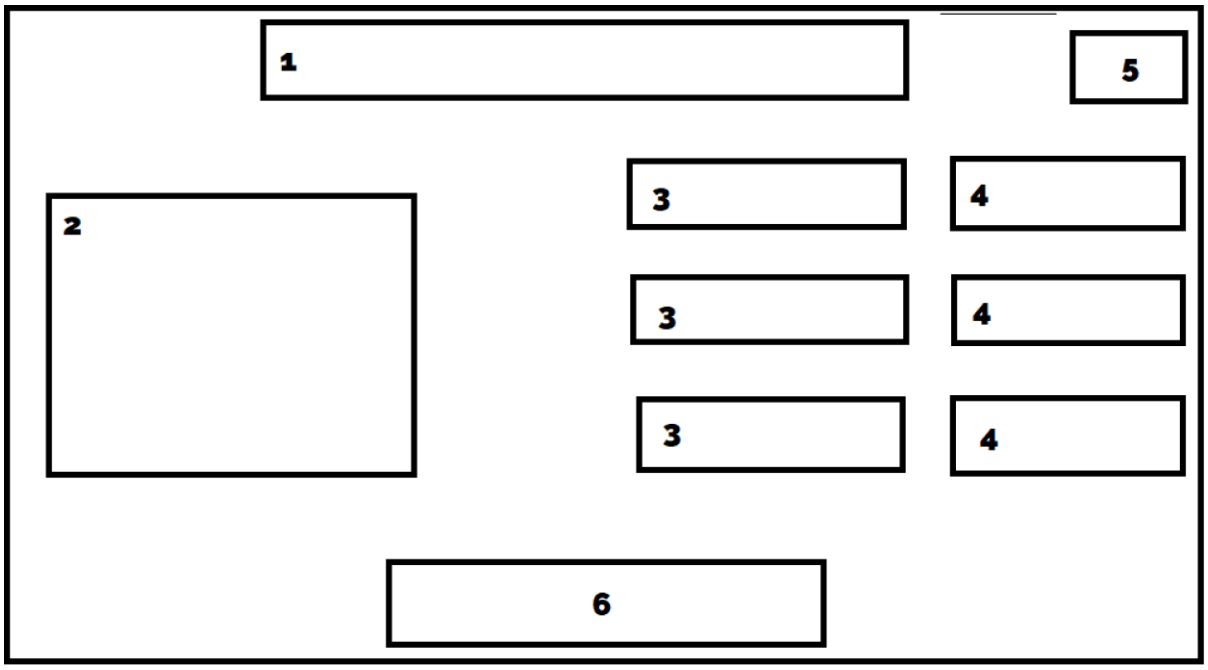

Рисунок Г.3 – Конструктивна схема внутрішньоігрового магазина

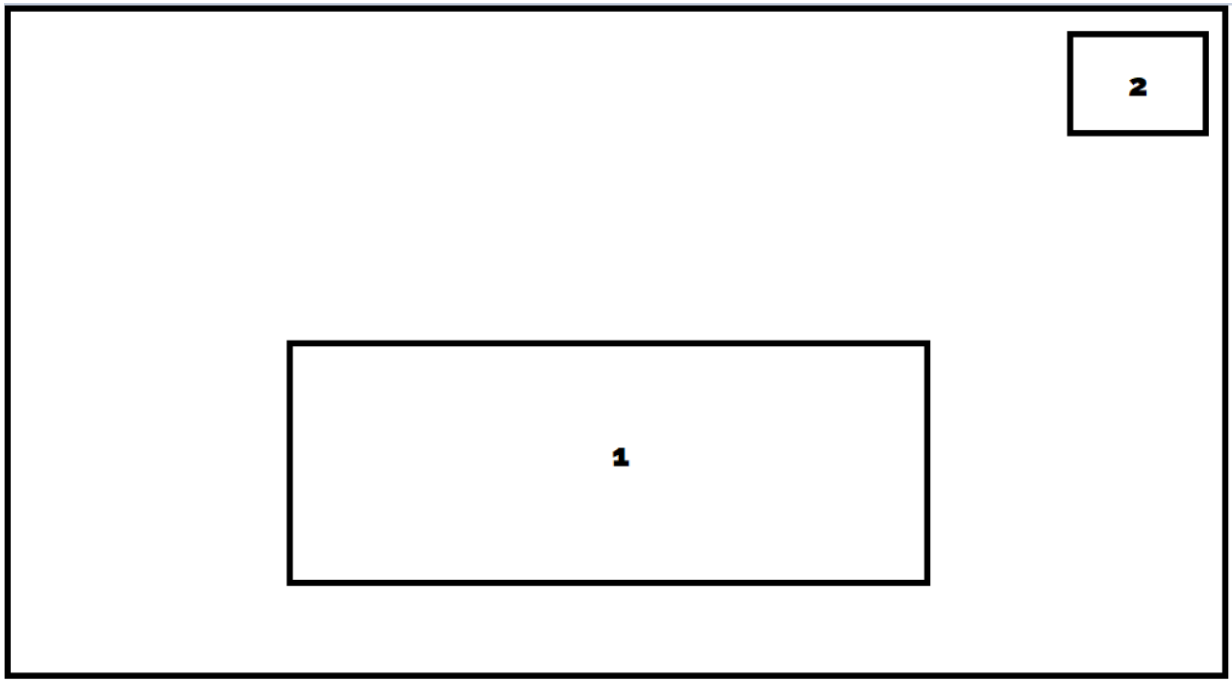

Рисунок Г.4 – Конструктивна схема меню паузи

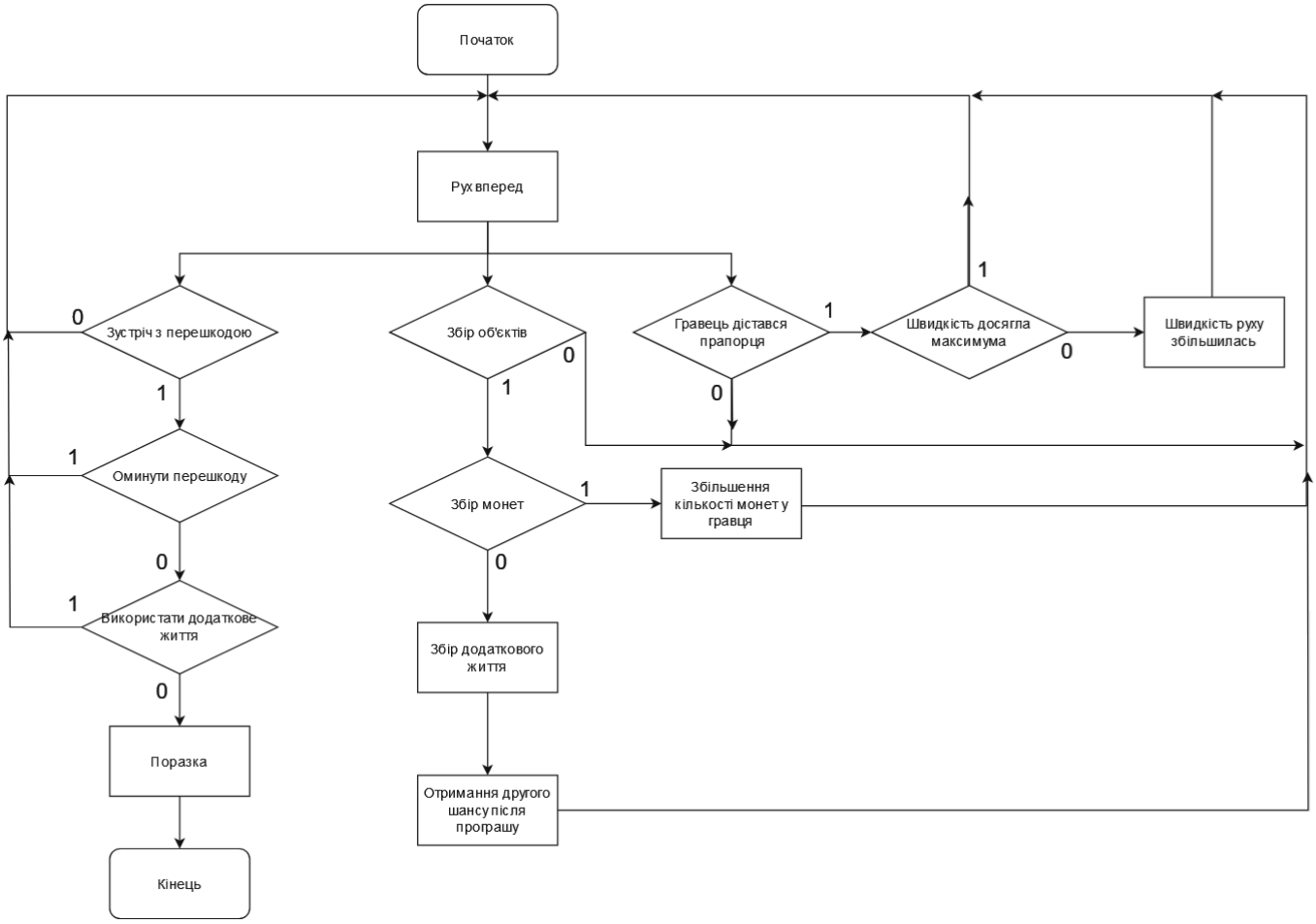

Рисунок Г.5 – Схема алгоритму роботи 2D відеогри жанру “runner”

Рисунок Г.6 – Діаграма генерації платформ рівня.

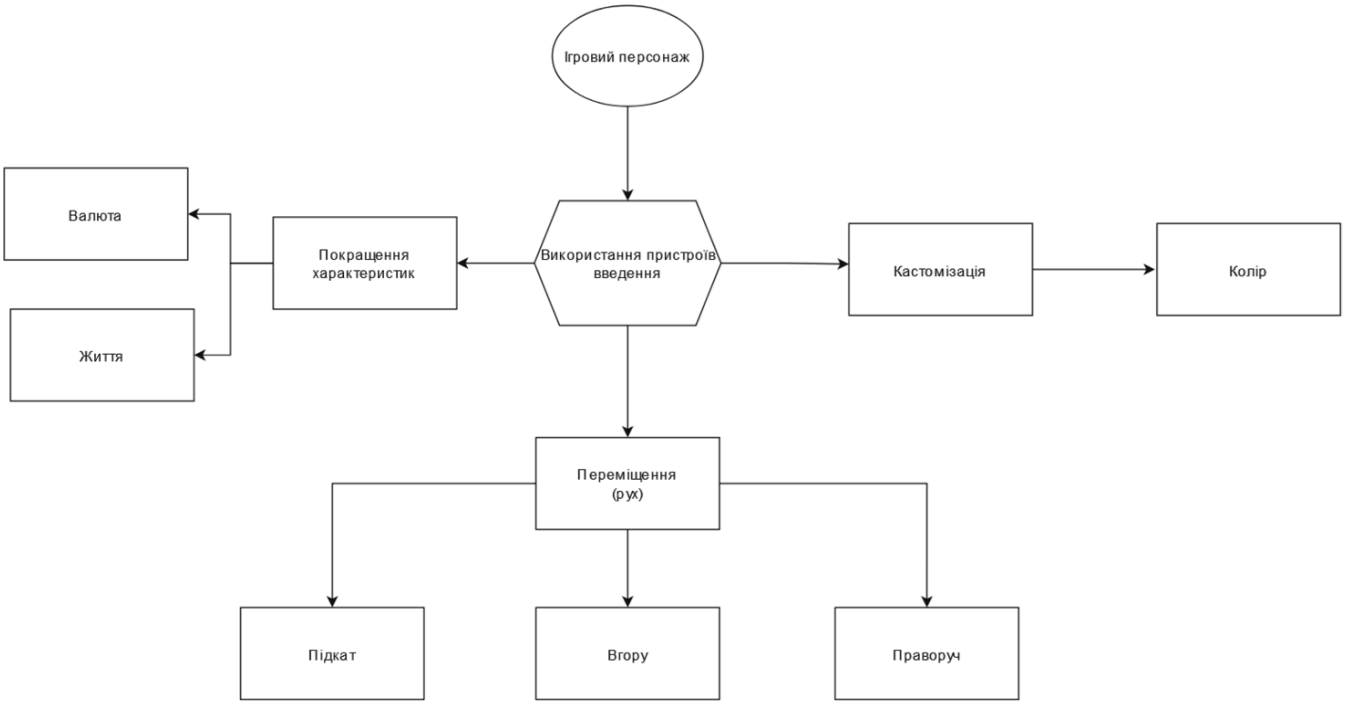

Рисунок Г.7– Типова схема функціональної моделі персонажа комп’ютерної гри

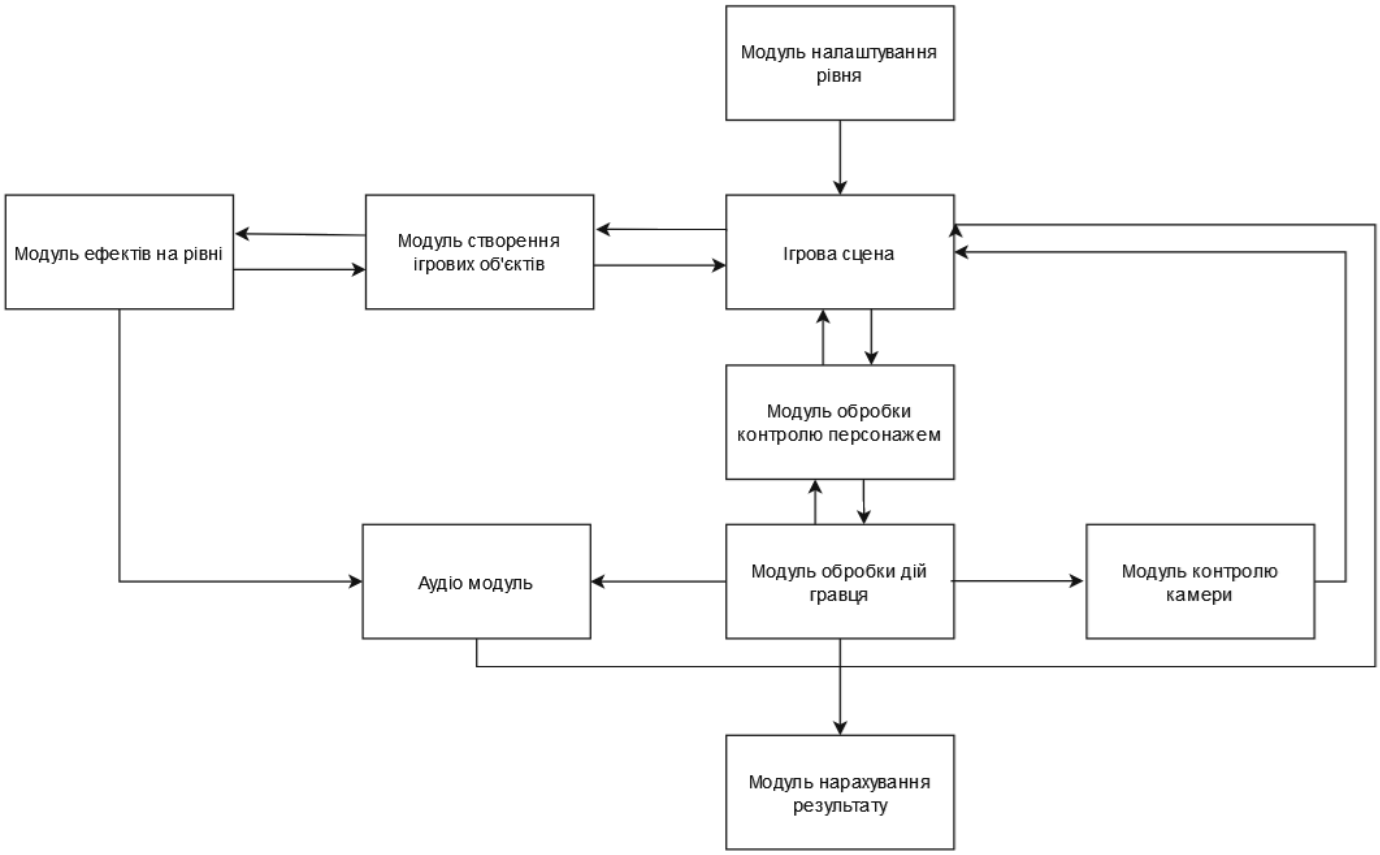

Рисунок Г.8 – Типова схема моделі реалізації ігрової системи

Рисунок Г.9 – Діаграма ігрового циклу

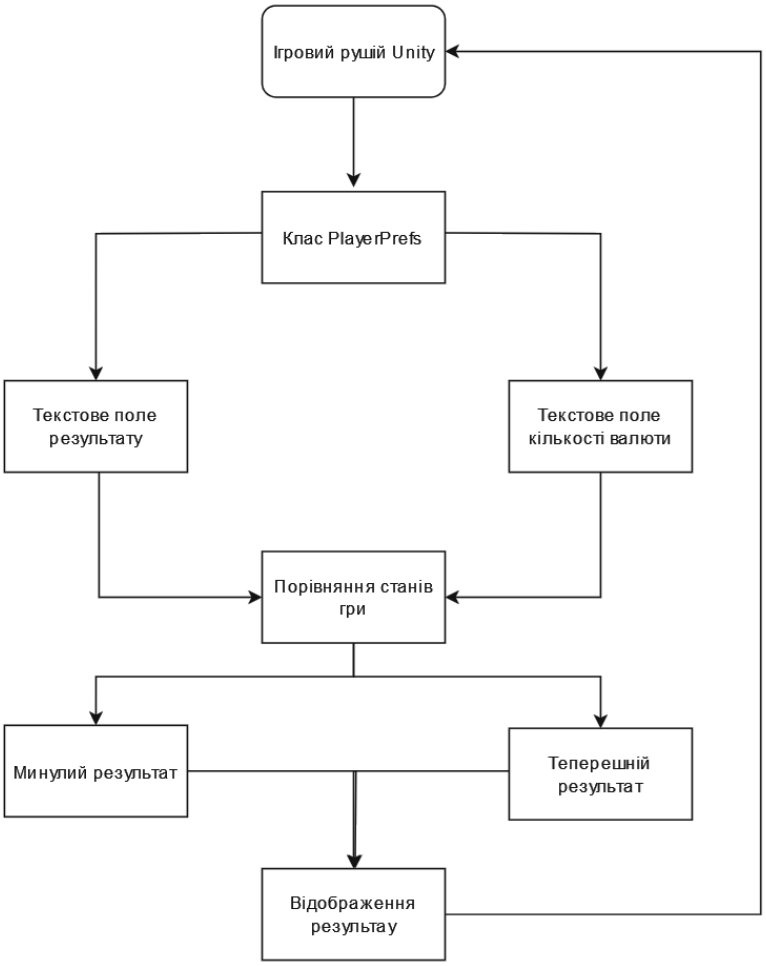

Рисунок Г.10 – Діаграма механізму зберігання ігрових даних.

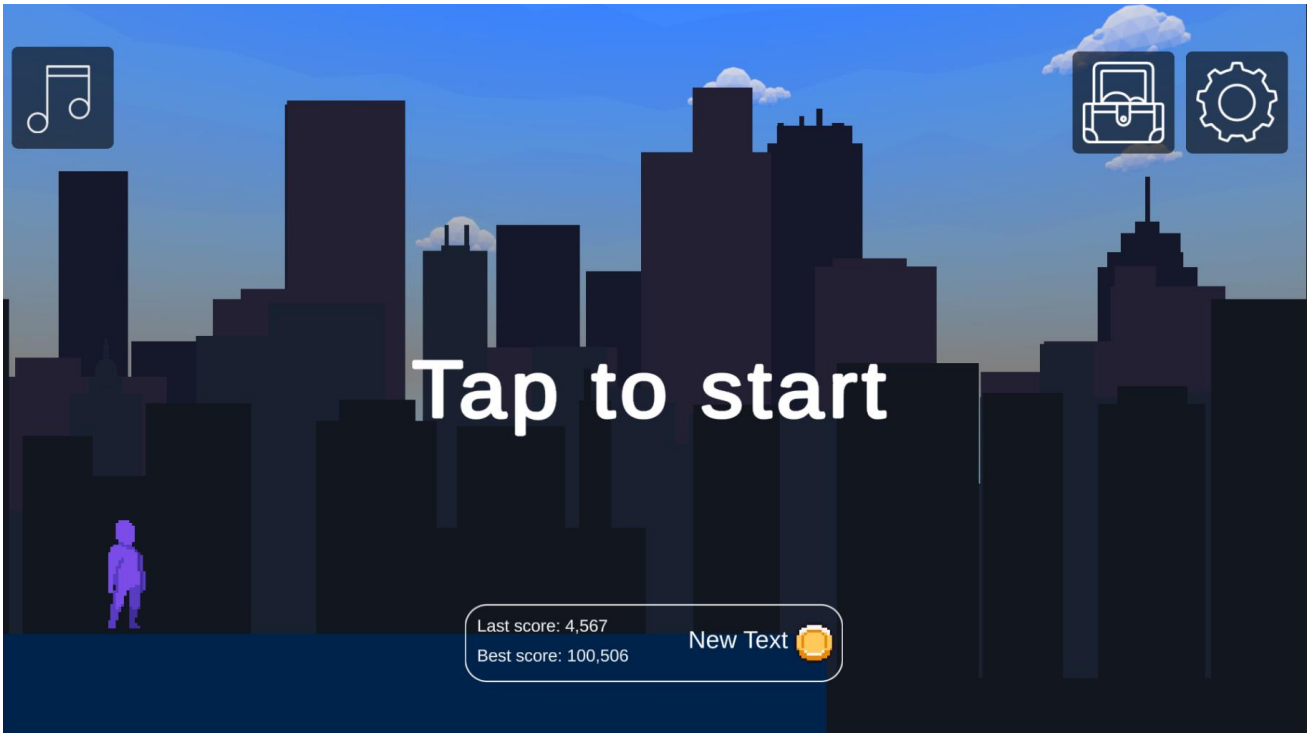

Рисунок Г.11 – Загальний вигляд інтерфейсного меню «Головне меню» комп’ютерної гри

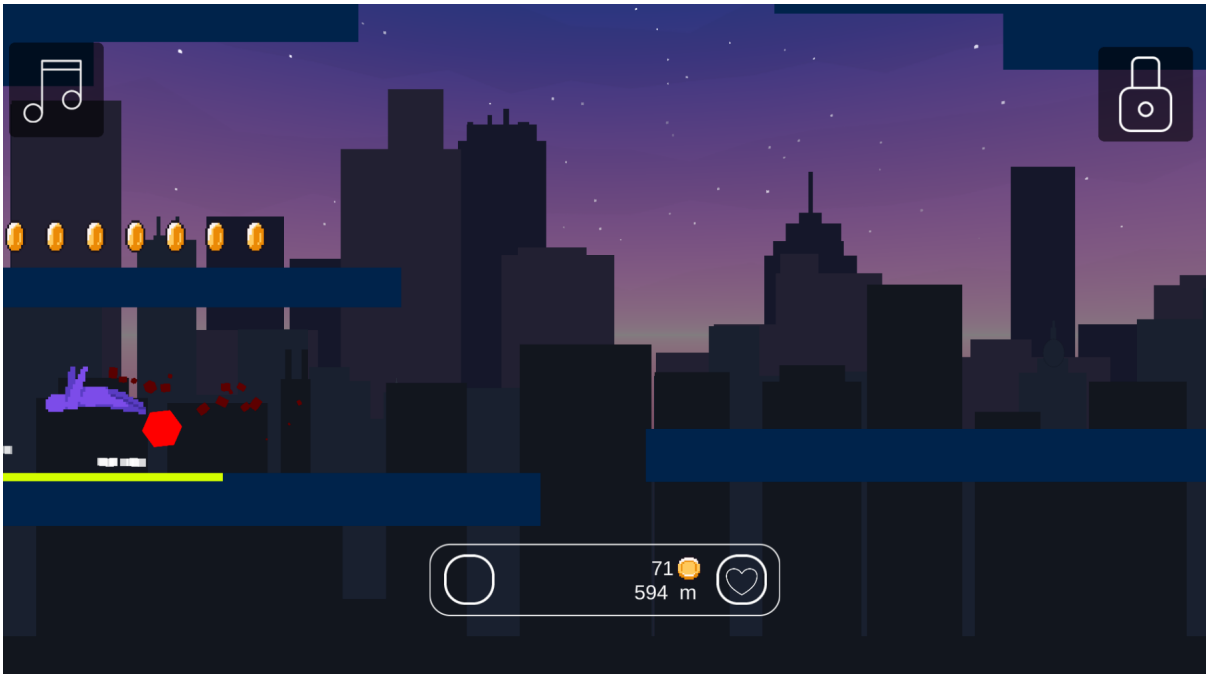

Рисунок Г.12 – Загальний вигляд інтерфейсу після смерті персонажа

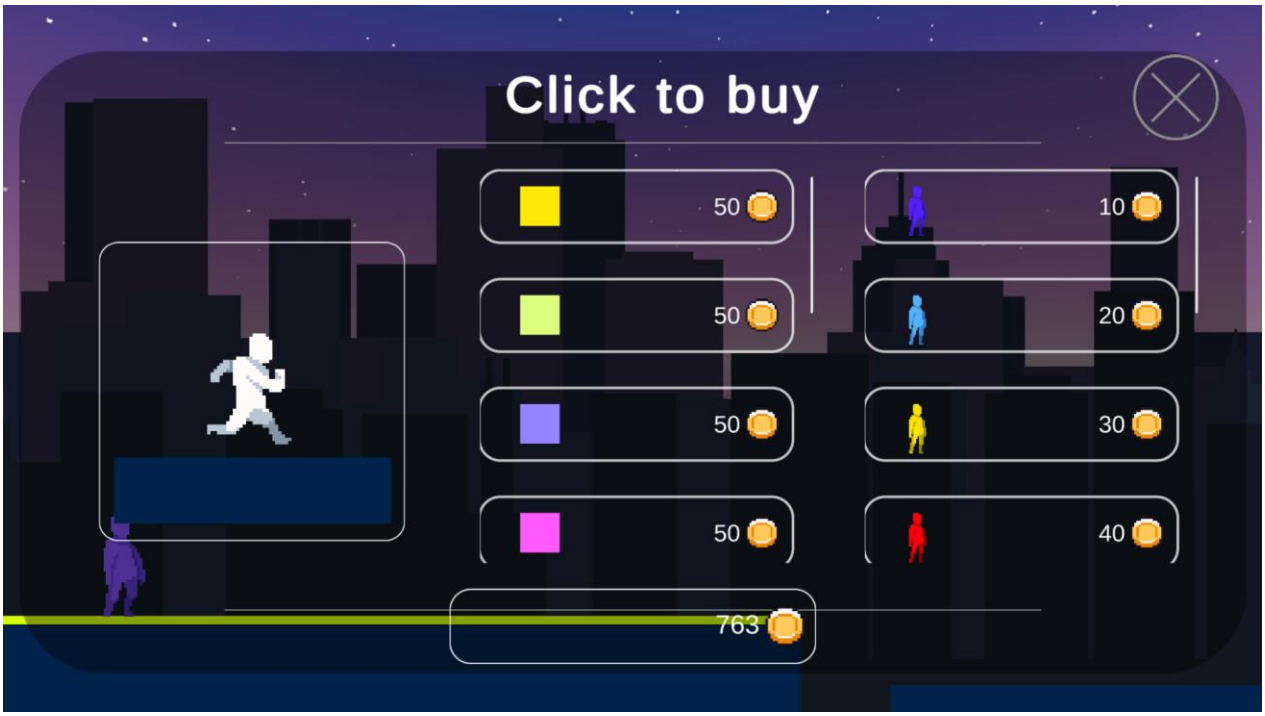

Рисунок Г.13 – Загальний вигляд інтерфейсного меню «Магазин» покупки кастомізації

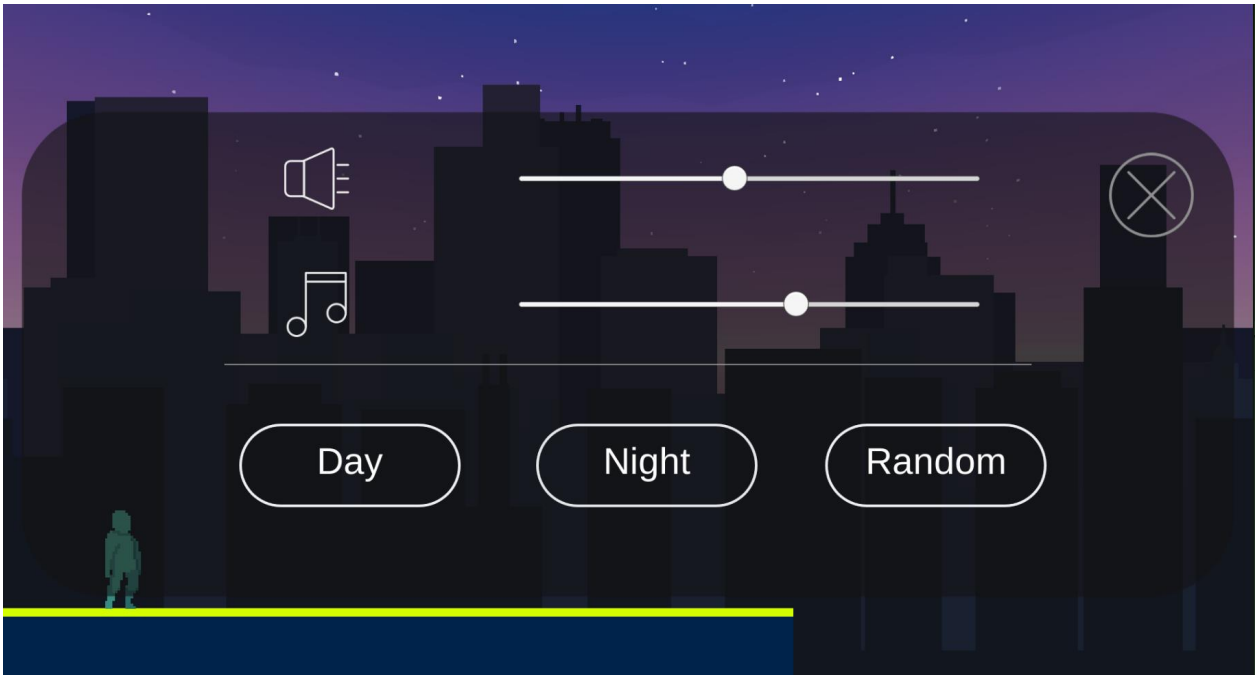

Рисунок Г.14 – Загальний вигляд інтерфейсного меню «Налаштування»