ДВНЗ «ДОНЕЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ»

Факультет комп'ютерно-інформаційних технологій та автоматизації

Кафедра прикладної математики і інформатики

«До захисту допущено»

Завідувачка кафедри ПМІ

_____Ольга ДМИТРІЄВА

«___»_____2022 р.

Випускна кваліфікаційна робота

бакалавра

(освітній ступень)

на тему

«Розробка однокористувацької гри-платформера для комп'ютерних пристроїв на базі Windows»

зі спецчастиною

«Розробка сценарію, логіки, інтерфейсу та програмних модулів ігрового додатку засобами C#, Unity 3D»

Виконав: студент 4 курсу, групи ІПЗ-18

Спеціальності 121 Інженерія програмного забезпечення

Цибулько О.І. _____

(прізвище та ініціали) (підпис)

Керівник каф. ПМІ, д.т.н., проф. Ольга ДМИТРІЄВА _____

(посада, науковий ступінь, вчене звання, ім'я та прізвище) (підпис)

Рецензент зав. каф. АТ, д.т.н., доц. Іван ЛАКТІОНОВ _____

(посада, науковий ступінь, вчене звання, ім'я та прізвище) (підпис)

Нормоконтроль:

_____ Ірина НАЗАРОВА

(підпис)

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.

Дата 7.06.2022

Студент _____

(підпис)

Луцьк – 2022 р.

ДВНЗ «ДОНЕЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ»

Факультет комп'ютерно-інформаційних технологій та автоматизації

Кафедра прикладної математики та інформатики

Освітньо-кваліфікаційний рівень (освітній ступінь) бакалавр

Напрям підготовки (спеціальність) 121 Інженерія програмного забезпечення
(шифр і назва)

ЗАТВЕРДЖУЮ:
Завідувачка кафедри ПМІ
Ольга ДМИТРІЄВА
/_____/
"____" _____2022 р.

ЗАВДАННЯ
НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТУ

Олексій Цибулько

1. Тема проєкту (роботи): Розробка однокористувацької гри-платформера для комп'ютерних пристроїв на базі Windows.
Спецчастина: Розробка сценарію, логіки, інтерфейсу та програмних модулів ігрового додатку засобами C#, Unity 3D.
Керівник проєкту (роботи) Ольга ДМИТРІЄВА, д.т.н, проф., зав. каф. ПМІ
(ім'я, прізвище, науковий ступінь, вчене звання, посада)
затверджені наказом від "06" травня 2022 року № 180

2. Строк подання студентом проєкту (роботи)_____

3. Вихідні дані до проєкту (роботи) результати науково-дослідної роботи, матеріали переддипломної практики.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1) проведення аналізу предметної області;
2) проведення аналізу додатків цього жанру;
3) визначення основних механік гри;
4) визначення аудіо-візуального стилю;
5) реалізація основних механік та створення сцен та сценаріїв;
6) розробка користувацького інтерфейсу;
7) провести тестування програми.

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

1) блок-схеми розроблених алгоритмів поведінки ворогів;
2) діаграма варіантів використання;
3) екранні форми.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Підпис, дата	Підпис, дата

7. Дата видачі завдання 6 квітня 2022

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломної роботи	Строк виконання етапів роботи	Примітка
1.	Об'єктний аналіз предметної області і постановка завдання	10.04.2022	
2.	Створення наборів спрайтів для побудови сцен	13.04.2022	
3.	Розробка механік пересування	20.04.2022	
4.	Побудова сцен	05.05.2022	
5.	Розробка унікальних механік	21.05.2022	
6.	Розробка розділів меню	24.05.2022	
7.	Тестування програми й аналіз	28.05.2022	
8.	Оформлення пояснювальної записки	01.06.2022	
9.	Підготовка слайдів презентації	05.06.2022	
10.	Підготовка демонстраційної версії розробленого програмного продукту	10.06.2022	
11.	Написання доповіді	12.06.2022	

Студент Олексій Цибулько
(підпис) (ім'я, прізвище)

Керівниця роботи Ольга ДМИТРІЄВА
(підпис) (ім'я, прізвище)

АНОТАЦІЯ

Цибулько Олексій Ігорович. Однокористувацька гра-платформер для комп'ютерних пристроїв на базі Windows / Випускна кваліфікаційна робота на здобуття освітньо-кваліфікаційного рівня «бакалавр» за спеціальністю 121 Інженерія програмного забезпечення. – ДВНЗ ДонНТУ, Покровськ, Луцьк, 2022.

Об’єктом дослідження є процес проєктування та розробки комп’ютерного ігрового додатку однокористувацької гри-платформера для комп’ютерних пристроїв на базі Windows.

Предметом дослідження є основні алгоритми і методи розробки відеоігор в 2D- і 3D- середовищах та ігровий рушій Unity.

Метою роботи є розробка сценарію та логіки програмного додатку однокористувацької гри-платформера для комп'ютерних пристроїв на базі Windows засобами C# та Unity.

Практичне значення розробки комп’ютерного додатку полягає в створенні гри в популярному жанрі з новим поглядом на деякі його аспекти, зацікавлення цільової аудиторії в своєму проєкті та надання їй якісного продукту для приємного проведення часу.

Методи досліджень базуються на використанні основних положень теорії алгоритмів, проєктування програмного забезпечення, об’єктно-орієнтованого програмування.

Ключові слова: гра-платформер, логіка, сценарій, програмний додаток, Unity, мова C#, фреймворк, операційна система Windows

ЗМІСТ

ВСТУП 8
1 АНАЛІЗ СУЧАСНИХ ПІДХОДІВ ДО ПРОЄКТУВАННЯ ІГОР-ПЛАТФОРМЕРІВ ДЛЯ КОМП'ЮТЕРНИХ ПРИСТРОЇВ 9
1.1 Обґрунтування вибору теми проєктування, її актуальності 9
1.2 Мета і завдання проєктування 9
1.3 Методи дослідження 9
1.4 Обґрунтування обраних методів 10
1.5 Системний аналіз предметної області 10
1.5.1 Графічний дизайн 10
1.5.2 Аудіо дизайн 14
1.5.3 Ігровий дизайн 14
1.6 Дослідження методів розробки та проєктування 17
1.6.1 Об’єктно-орієнтоване програмування 17
1.6.2 Патерн проєктування «Одинак» 18
1.6.3 Патерн проєктування «Пул об’єктів» 19
1.6.4 Мова моделювання UML 20
1.7 Основні завдання роботи 20
1.8 Висновки за розділом 21
2 ПРОЄКТУВАННЯ ОДНОКОРИСТУВАЦЬКОЇ ГРИ-ПЛАТФОРМЕРА 22
2.1 Визначення функцій програмного засобу 22
2.2 Проєктування системи 22
2.3 Проєктування людино-машинного інтерфейсу 27
2.4 Засоби розробки 30
2.4.1 Використаний ігровий рушій 30
2.4.2 Обґрунтування вибору середовища розробки та мови програмування 31
2.4.3 Визначення додаткових інструментів розробки 32
2.5 Формування концепт-документу проєкту 33
2.6 Висновки за розділом 35
3 РОЗРОБКА ПРОГРАМНОГО ДОДАТКУ ОДНОКОРИСТУВАЦЬКОЇ ГРИ-ПЛАТФОРМЕРА 36

3.1 Підготовка матеріалів для додатку 36
3.2 Побудова ігрового рівня 38
3.2.1 Налаштування об’єкта камера 38
3.2.2 Налаштування об’єкта полотно 40
3.2.3 Створення ігрового простору 41
3.2.4 Реалізація керування головним героєм 53
3.3 Створення головного меню 55
3.4 Висновки за розділом 56
4 ОПИС РОБОТИ ПРОГРАМНОЇ СИСТЕМИ 57
4.1 Меню програми 57
4.2 Структура рівнів 61
4.2.1 Перший рівень 61
4.2.2 Другий рівень 63
4.2.3 Третій рівень 65
4.3 Висновки за розділом 68
5 ТЕСТУВАННЯ ПРОГРАМНОЇ СИСТЕМИ 69
5.1 Тестування розробленої системи 69
5.1.1 Тестування компонентів 70
5.1.2 Інтеграційне тестування 70
5.1.3 Системне тестування 71
5.1.4 Альфа-тестування 73
5.1.5 Бета-тестування 74
5.2 Висновки за розділом 76
ВИСНОВКИ 77
ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ 79
Додаток А Зауваження нормоконтролера 82
Додаток Б Графічна частина 84
Додаток В Графічні елементи 94
Додаток Г.1 Лістинг класу «FinishMenu.cs» 99
Додаток Г.2 Лістинг класу «MainMenu.cs» 101
Додаток Г.3Лістинг класу «PauseMenu.cs» 103
Додаток Г.4 Лістинг класу «InitAudio.cs» 105
Додаток Г.5 Лістинг класу «InitQuality.cs» 107
Додаток Г.6 Лістинг класу «SetAudio.cs» 109

Додаток Г.7 Лістинг класу «SetQuality.cs» 111
Додаток Г.8 Лістинг класу «BossArm.cs» 113
Додаток Г.9 Лістинг класу «BossEye.cs» 115
Додаток Г.10 Лістинг класу «Bullet.cs» 117
Додаток Г.11 Лістинг класу «BulletFire.cs» 119
Додаток Г.12 Лістинг класу «BulletPool.cs» 122
Додаток Г.13 Лістинг класу «ButtonNew.cs» 124
Додаток Г.14 Лістинг класу «CameraControl.cs» 126
Додаток Г.15 Лістинг класу «Door.cs» 128
Додаток Г.16 Лістинг класу «Entity.cs» 130
Додаток Г.17 Лістинг класу «FocusOnTarget.cs» 132
Додаток Г.18 Лістинг класу «Follow.cs» 134
Додаток Г.19 Лістинг класу «MoveRespawn.cs» 136
Додаток Г.20 Лістинг класу «PlateNew.cs» 138
Додаток Г.21 Лістинг класу «PlatformMove.cs» 140
Додаток Г.22 Лістинг класу «PlatformToStart.cs» 143
Додаток Г.23 Лістинг класу «Player.cs» 145
Додаток Г.24 Лістинг класу «Sounds.cs» 148
Додаток Г.25 Лістинг класу «SpeedBoost.cs» 150
Додаток Г.26 Лістинг класу «Stoper.cs» 152

ВСТУП

Сьогодні персональний комп’ютер є в кожної людини. Він став частиною повсякденного життя як інструмент для роботи чи відпочинку. Одним з відносно нових видів відпочинку на теперішній час виступають комп’ютерні ігри.

Сучасні комп’ютерні ігри є видом інтерактивних розваг які захоплюють всю увагу користувача. Останнім часом вони набирають все більшої популярності. Відеоігри бувають як простими з візуальної та геймплейної точки зору, так і складними системами з надзвичайною кількістю можливостей.

В наслідок популярності комп’ютерних ігор, виникло чи не мало доступних інструментів для їх створення та модифікації. Кожен може себе спробувати себе в геймдеві, оскільки сучасний інструментарій дозволяє розробляти відеоігри самотужки.

Тож темою кваліфікаційної роботи є розробка однокористувацької гри-платформера для комп'ютерних пристроїв на базі Windows.

Об’єкт дослідження – процес проєктування та розробки ігор для комп'ютерних пристроїв.

Мета дослідження – розробка однокористувацької гри-платформера.

1 АНАЛІЗ СУЧАСНИХ ПІДХОДІВ ДО ПРОЄКТУВАННЯ ІГОР-ПЛАТФОРМЕРІВ ДЛЯ КОМП'ЮТЕРНИХ ПРИСТРОЇВ

1.1 Обґрунтування вибору теми проєктування, її актуальності

Тема «Розробка однокористувацької гри-платформера для комп'ютерних пристроїв на базі Windows» обрана через те, що зараз комп’ютерна ігрова індустрія розвивається найактивніше за весь період свого існування, тому є безліч можливостей для працевлаштування в даній сфері, від інді-розробника до члена команди розробників ААА-проєкту [1].

1.2 Мета і завдання проєктування

Метою роботи є розробка програмного додатку однокористувацької гри-платформера для комп'ютерних пристроїв на базі Windows за допомогою засобів C# [2] та Unity [3].

1.3 Методи дослідження

Насамперед було використано емпіричний метод дослідження – спостереження. Цим методом було досліджено загальну структуру програмного застосунку та його складові частини. Далі, за допомогою порівняльного аналізу, було визначено оптимальні методи реалізації, необхідних в даному випадку, структурних елементів. В завершення, за допомогою експериментального методу, було перероблено обрані методи реалізації структурних елементів під власні потреби та, за необхідністю, додано власні й видалено зайві.

1.4 Обґрунтування обраних методів

За допомогою спостереження було підчерпнуто інформацію з досвідчених джерел, що запобігає витраченню часу на проєктування базових структур. Порівняльний аналіз використано оскільки в даній сфері немає єдиного правильного рішення всіх задач, тож необхідно аналізувати декілька джерел інформації та обирати найбільш доцільне. Експериментальним методом допрацьовується або редагується те чи інше рішення бо кожен проєкт є унікальним та потребує деяких змін для покращення єдності та ефективності системи.

1.5 Системний аналіз предметної області

В ігровій індустрії існують та здобувають популярність досить прості проєкти, відносно реалізованих в них можливостей. Їх класифікують як аркади – ігри з навмисне примітивним ігровим процесом. Аркадні ігри не відрізняються високою інтерактивністю проте зачіпають креативним підходом до простих механік. Цей ігровий жанр є чудовим вибором для починаючого інді-розробника – людини чи невеличної групи осіб які працюють над власним проєктом, зазвичай без зовнішнього фінансування. В аркаді можна легко реалізувати власні ігрові механіки не сильно відволікаючись на пропрацьований сюжет чи розкішний візуальний стиль. Тут головним є сам ігровий процес.

1.5.1 Графічний дизайн

Досить популярне просте графічне оформлення у стилі мінімалізму, наприклад такі проєкти як «Geometry Dash» [4], «Journey» [5], «Thomas Was Alone» [6] (рис. 1.1 - 1.3) та інші, або ж у стилі піксель-арт – форма цифрового живопису, створеного на комп’ютері за допомогою растрового графічного редактора, де зображення редагується на рівні пікселів. Прикладом інді-проєктів з використанням піксельної графіки є «Undertale» [7], «Hyper Light

Drifter» [8], «Celeste» [9] (рис. 1.4 - 1.6) та інші. Такі стилі здобули свою популярність через простоту їх освоєння.

Рисунок 1.1 – Приклад мінімалізму в «Geometry Dash» [4]

Рисунок 1.2 - Приклад мінімалізму в «Journey» [5]

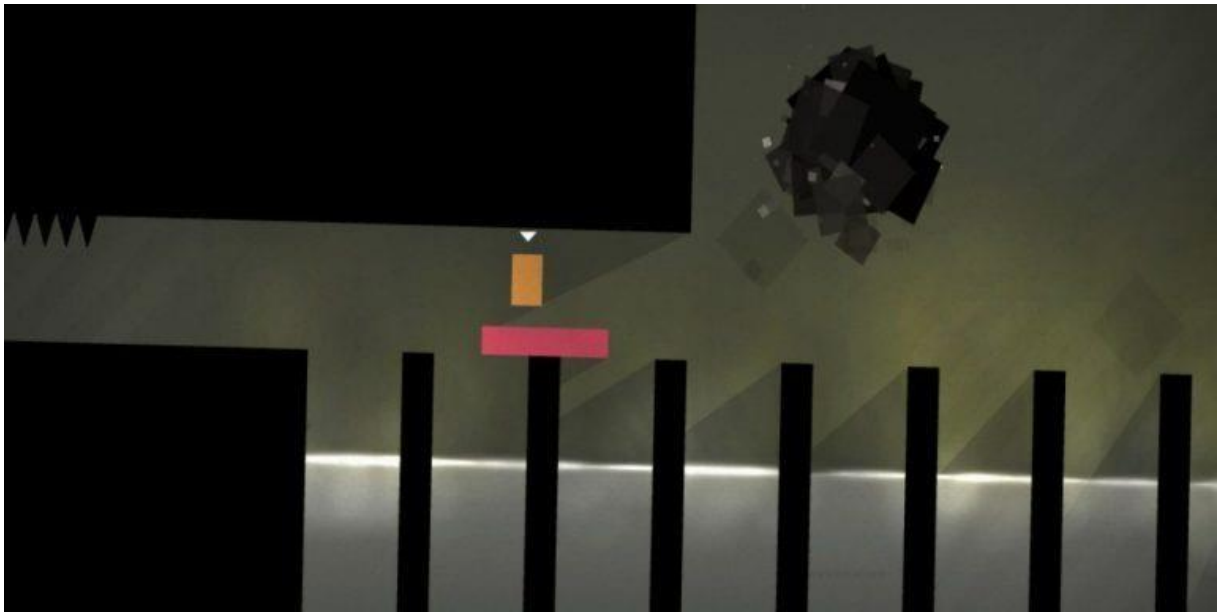


Рисунок 1.3 - Приклад мінімалізму в «Thomas Was Alone» [6]

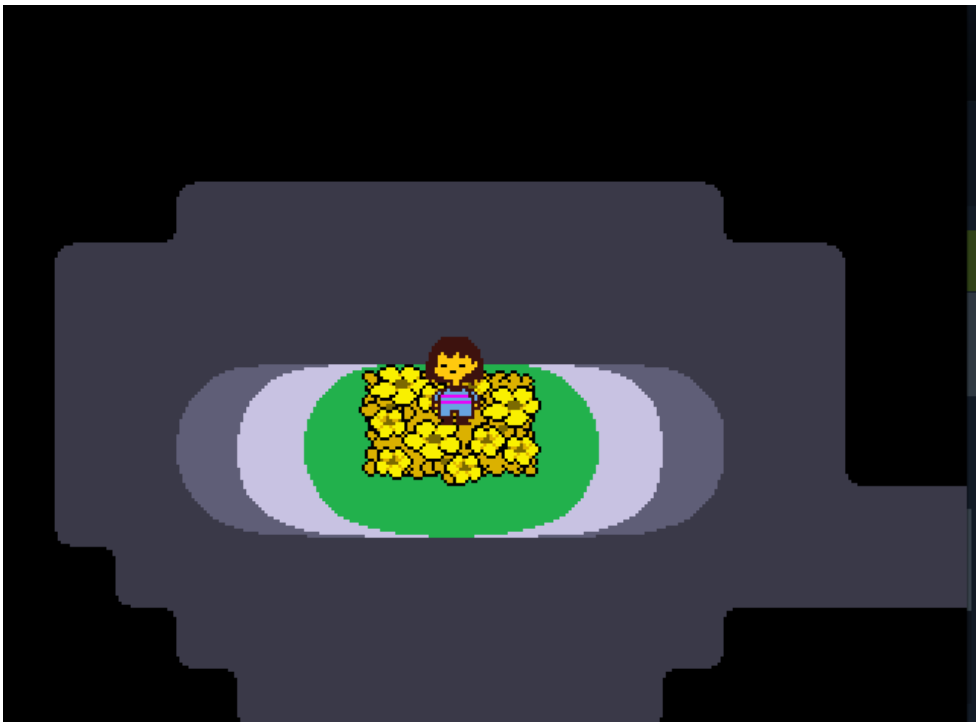


Рисунок 1.4 – Приклад піксельної графіки в «Undertale» [7]

Рисунок 1.5 - Приклад піксельної графіки в «Hyper Light Drifter» [8]

Рисунок 1.6 - Приклад піксельної графіки в «Celeste» [9]

1.5.2 Аудіо дизайн

Значну роль також грає аудіо частина проєкту. Через музики задається необхідний темп та настрій гри. Звуковими ефектами, гравцю подається необхідна інформація, чи то звук успішної взаємодії з об’єктом, чи то поточний стан ігрового персонажа. Часом аудіо частина є наскільки важливою, що коло неї будується весь проєкт. Проєкти даного жанру називають ритм-іграми або музикальними іграми, а вся суть зводиться до виконання гравцем певних дій в такт музики. В приклад можна привести раніше зазначену гру «Geometry Dash» [6].

1.5.3 Ігровий дизайн

Найважливішою частиною розробки гри є ігровий дизайн – процес створення наповнення самої гри. Робота з геймдизайном може виконуватися як за допомогою спеціального документу – дизайн-документу, так і існувати просто в голові розробника. У дизайн-документі описуються правила та особливості гри, що допомагає сформувати загальне уявлення про проєкт до початку його розробки.

Ігровий дизайн поділяється на наступні типи:

– системний дизайн – створення правил та відповідних залежностей;

– контент-дизайн – створення персонажів, предметів, завдань та загадок;

– дизайн рівнів – створення рівнів гри, що включає в себе ландшафт на рівні та розміщення об’єктів на ньому;

– дизайн світу – продумування тематики гри на загальному рівні, зв’язування локацій між собою та відведення їм певної мети.

Розглянемо презентацію та наведені в ній тезиси «Методологія розробки ігор з позиції гейм-дизайна» [10] та спробуємо визначити її актуальність для інді-розробника.

Тож основними етапами в розробці ігрового дизайну є:

– концепт;

- прототип;
- вертикальний зріз;
- виробництво контенту;
- закритий бета-тест та відкритий бета-тест;
- реліз.

Перш за все необхідно скласти концепт-документ**.** Метою даного кроку є визначення того чим є гра: платформа розповсюдження, жанр/тематика, цільова аудиторія, цільовий ринок, основні механіка гри, строки, технічні аспекти, оцінка ризиків. Даний пункт буде корисний будь-якому розробнику, оскільки маючи готовий план дій на папері легше вести подальшу розробку не відволікаючись на сумнівні зміни.

Створення прототипу. Мета даного кроку – оцінити наскільки цікавою буде гра. Основним принципом є робота тільки над ключовими елементами, незважаючи на баги рівня Critical – порушення логіки програми але з можливістю її запуска, проте без багів рівня Blocker – порушення логіки програми без можливості її запуску. Даний пункт буде корисний будь-якому розробнику, оскільки потрібно знати на самих ранніх етапах розробки чи є цікавим проєкт.

Вертикальний зріз. Мета даного кроку – довести, що робота йде в правильному напряму. Головними цілями етапу є:

- доведення одного ключового елементу гри до фінального стану;
- розкриття центрального ігрового стилю;
- розробка базових елементів;
- мінімальна наповненість контентом.

Даний пункт буде корисний будь-якому розробнику, оскільки він дозволяє подивитися на основні аспекти гри до того як буде виконана робота з наповненням ігрового простору додатковими елементами.

Закритий бета-тест. Мета даного кроку – демонстрація перед людьми, пошук гейм-дизайнерських помилок, усунення багів рівня Critical. Головними цілями є:

– у грі опрацьовано 30% контенту;
– присутні всі головні елементи;
– логіка гри близька до фінальної;

Даний пункт буде корисний будь-якому розробнику, оскільки він дозволяє отримати відгуки від людей непричасних до проєкту, оцінка яких є більш об’єктивною ніж самого розробника.

Відкритий бета-тест. Мета даного кроку – тест гри на широкій аудиторії, оптимізація навантажень, підготовка до запуску. Головними цілями є:

– у грі опрацьовано 80% контенту;
– підключена монетизація, гра приймає платежі;
– гра готова до великого потоку трафіку;
– функціонує інфраструктура продукту.

Цей пункт є суперечливим по декільком причинам. З позиції інді-розробника, відкритий бета-тест може затягнутися на завжди, індустрія знає випадки коли ігри не виходять з відкритого бета-тесту роками через постійні бажання додати нові функцій та вдосконалювати старі, що відштовхне потенційних покупців які хочуть отримати завершений продукт. Також суперечливим є пункт про підключення монетизації на цьому етапі, остільки в покупців може виникнути погане передчуття через те, що гра ще в розробці але уже вимагає гроші у гравців. Тож щоб не ризикувати власним авторитетом, краще здержати себе від довготривалого відкритого бета-тесту та відмовитися від монетизації на цьому етапі розробки.

Реліз. Мета даного кроку – отримання прибутку. Стан проєкту на момент настання цього етапу:

– налагоджена дистрибуція ігрового продукту;
– дотримуються маркетингові та фінансові плани;

– зібрана статистика з відкритого бета-тесту та ведуться роботи над покращенням;

Даний пункт буде корисний будь-якому розробнику, оскільки будь-який проєкт повинен виходити на продаж в своєму найкращому вигляді, за час зазначений графіком.

Проаналізувавши презентацію та наведені в ній тезиси «Методологія розробки ігор з позиції гейм-дизайну» [4] можна зробити висновок про те. що загалом ця методологія підходить для інді-розробника, з урахуванням невеликих корективів в плані проведення відкритого бета-тесту.

1.6 Дослідження методів розробки та проєктування

При розробці програми важливим фактором є цілісне бачення її структури. Методології розробки програмного забезпечення пропонують сукупність методів зі спільним філософським підходом, які використовують на різних етапах життєвого циклу програми. Патерни проєктування надають рішення для часто повторюваних проблем при проєктування архітектури програми. Це загальна концепція, яку потрібно адаптувати під конкретну проблему.

1.6.1 Об’єктно-орієнтоване програмування

Об’єктно-орієнтоване програмування (ООП) [11-12] – це програмування, сфокусоване на даних, при чому дані та поведінка нерозривно зв’язані. Дані та поведінка разом представляють собою клас, а об’єкти являються екземплярами класів, а самі класи утворюють ієрархію успадкування.

Методи – підпрограми класу, які можуть оперувати його об’єктами. Загальна структура класу зображена на рисунку 1.7.

Концепції об’єктно-орієнтоване програмування:

– абстракція – формальний вигляд предмета, сформований у вигляді класу;

– інкапсуляція – механізм мови програмування, який дозволяє зв’язувати данні та методи, що працюють з цими даними в єдиний об’єкт;

– спадкування – можливість спадкування даних та функціональності від абстрактного типу даних до деякого існуючого типу;

– поліморфізм – можливість об’єктів з однаковою специфікацією мати різну реалізацію.

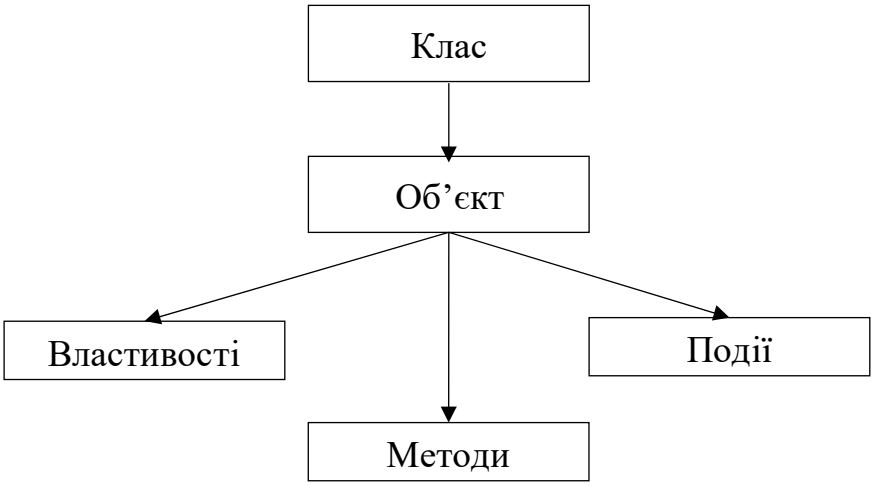


Рисунок 1.7 – Структура класу в ООП

1.6.2 Патерн проєктування «Одинак»

Одинак [13] – це патерн проєктування, який гарантує, що в класі є лише один екземпляр та надає до нього глобальну точку доступу.

Його проблемою є те, що патерн вирішую дві проблеми, порушуючи принцип єдиної відповідальності – принцип об’єктно-орієнтованого програмування, визначаючий, що кожен об’єкт повинен мати єдину відповідальність та ця відповідальність повинна бути інкапсульована в класі.

По-перше, «Одинак» гарантує наявність одного єдиного екземпляра класу, що корисно для доступу до загального ресурсу чи унікального об’єкту. По-друге, він представляє глобальну точку доступу до класу, через яку можна як отримати дані так і змінити їх. Реалізація патерна проєктування «Одинак» полягає в приховування конструктора за замовчуванням та в створенні публічного статичного методу, який буде грати роль конструктора та

контролювати життєвий цикл об’єкта. Структуру патерна зображено на рисунку 1.8.

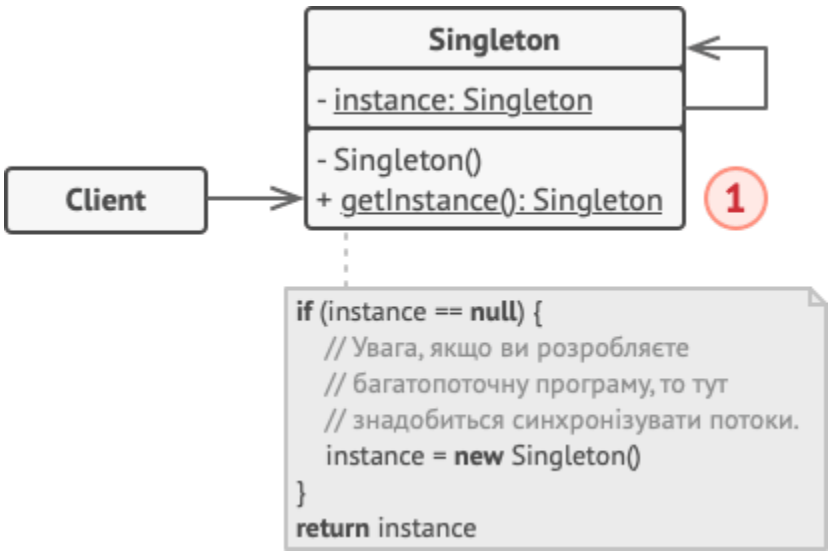


Рисунок 1.8 – Діаграма класів патерна проєктування «Одинак»

1.6.3 Патерн проєктування «Пул об’єктів»

Патерн «Пул об’єктів» [14-15] доцільно використовувати коли об’єкти в системі створюються часто та число створюваних об’єктів в одиницю часу обмежено. Всі багаторазово використовувані об’єкти, вільні в деякий проміжок часу, зберігаються в одному і тому ж пулі об’єктів. Тож ними можна керувати на основі єдиної політики.

Клієнт запитує в пулі вже створений екземпляр. Пул об'єктів може бути зростаючим, коли при відсутності вільних створюються нові об'єкти, або з обмеженням кількості створюваних об'єктів.

Клас пулу об’єктів зазвичай проєктується за допомогою патерна «Одинак».

Основна ідея парена «Пул об’єктів» - уникнути створення нових екземплярів класу в разі можливості їх повторного використання.

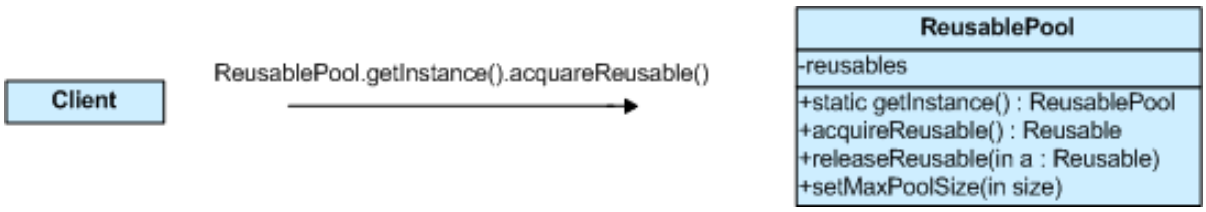


Рисунок 1.8 – Діаграма класів патерна проєктування «Пул об’єктів»

1.6.4 Мова моделювання UML

UML (Unified Modeling Language) [15-17] – мова графічного опису для об’єктного моделювання, що забезпечує стандартизований спосіб візуалізації дизайну системи.

Плюси використання UML моделювання:

– можливість розглянути задачу з різних точок зору;

– спрощене розуміння програмної системи та задач, необхідних для її реалізації;

– простота читання діаграм, при умові ознайомленості із їх синтаксисом.

Мінуси використання UML моделювання:

– трата часу на моделювання незначних частин системи;

– необхідність знання різноманітних діаграм та анотацій;

– надмірність мови;

– неточна семантика.

1.7 Основні завдання роботи

Завдання, виконання яких необхідне для досягнення поставленої мети:

1) проведення критичного аналізу предметної області;

2) проведення порівняльного аналізу сучасних технологій розробки ігрових додатків відповідного жанру;

3) розробка сценарію та обґрунтування логіки ігрового додатку, визначення основних механік гри та аудіо-візуального стилю;

4) проєктування програмної системи для реалізації розробленого сценарію;

5) програмна реалізація основних механік та створення сцен та сценаріїв;

6) розробка користувацького інтерфейсу;

7) проведення тестування розробленої програмної системи.

1.8 Висновки за розділом

У даному розділі було доведено актуальність теми, проведено аналіз предметної області, виконана постановка завдання. проведено аналіз проєктів від інших розробників, розбито їх на категорії. Сформовано ключові складові відеоігор. Проаналізовано основні етапи розробки проєкту та розглянуто презентацію з рекомендаціями від досвідченого розробника на відповідну тему.

Досліджено методи розробки та проєктування програмного забезпечення. Розглянуто об’єктно-орієнтований підхід до створення програмних засобів. Проаналізовано найбільш доцільний поведінковий патерн «Одинак» та мову графічного опису для об’єктного моделювання UML. Сформовано основні завдання роботи.

2 ПРОЄКТУВАННЯ ОДНОКОРИСТУВАЦЬКОЇ ГРИ-ПЛАТФОРМЕРА

2.1 Визначення функцій програмного засобу

Розроблюваний програмний продукт повинен реалізовувати людино-машинний інтерфейс для функціонального та інтуїтивно зрозумілого використання. Для досягнення даної мети необхідно реалізувати функції, наведені в наступному списку.

Графічний функціонал:

– вибір якості графіки.

Звуковий функціонал:

– регулювання гучності музики;

– регулювання гучності звуків.

Внутрішньоігровий функціонал:

– звукові ефекти;

– реалізація пересування;

– можливість переходу в головне меню в будь-який момент.

Головне меню:

– меню налаштувань;

– вибір рівня для проходження.

2.2 Проєктування системи

В якості засобу для проєктування обрано редактор Draw.io [11]– це онлайн-платформа з відкритим кодом, випущена в 2013 році, де люди можуть створювати схеми. Цей веб-сайт підтримує роботу в режимі реального часу, коли він підключений до облікового запису Google. Його головна перевага - безкоштовність. За користування ресурсів не стягується плата, що робить його ще більш доступним. Крім того, для повноцінної роботи не потрібно проходити реєстрацію і проходити процес авторизації на сайті. При вході на

головну сторінку потрібно лише вибрати шлях для збереження проєкту. Кінцеві результати можна зберегти на віддаленому сховищі – «хмарі» («Google Drive», «Dropbox», «OneDrive»), на ресурсі «GitHub», на жорсткому диску пристрою «Device» або безпосередньо в середу для управління розробкою веб-додатків і програм «Trello». Інтерфейс програми зручний та зрозумілий навіть з першого погляду, також вигідно виділяється підтримка української мови. На рисунку 2.1 зображено інтерфейс Draw.io.

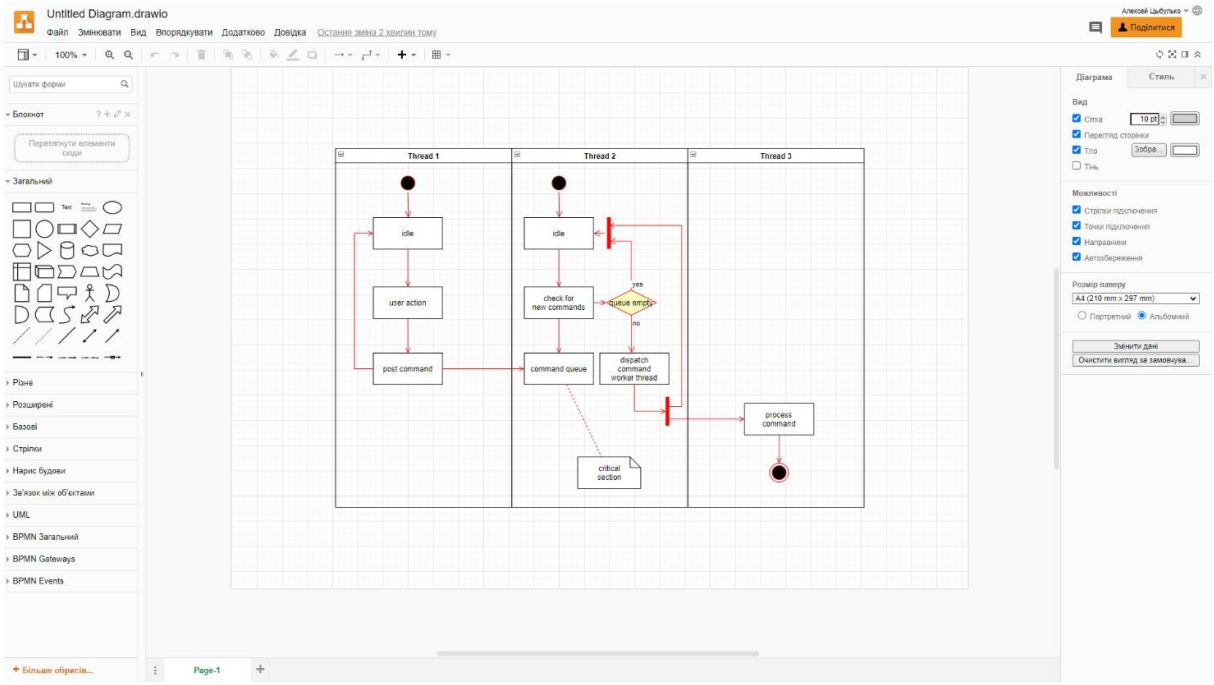


Рисунок 2.1 – Інтерфейс Draw.io

На побудованій діаграмі варіантів використання для користувача відображено всі можливі дії актора від натискання на ярлик до завершення ігрової сесії. Для відображення на діаграмі варіанту використання, використано фігуру еліпс, актора – фігура «чоловічка». Також, для відображення відношень між елементами діаграми, використано відношення асоціації, включення, розширення, узагальнення. На рисунку 2.2 зображено цю діаграму.

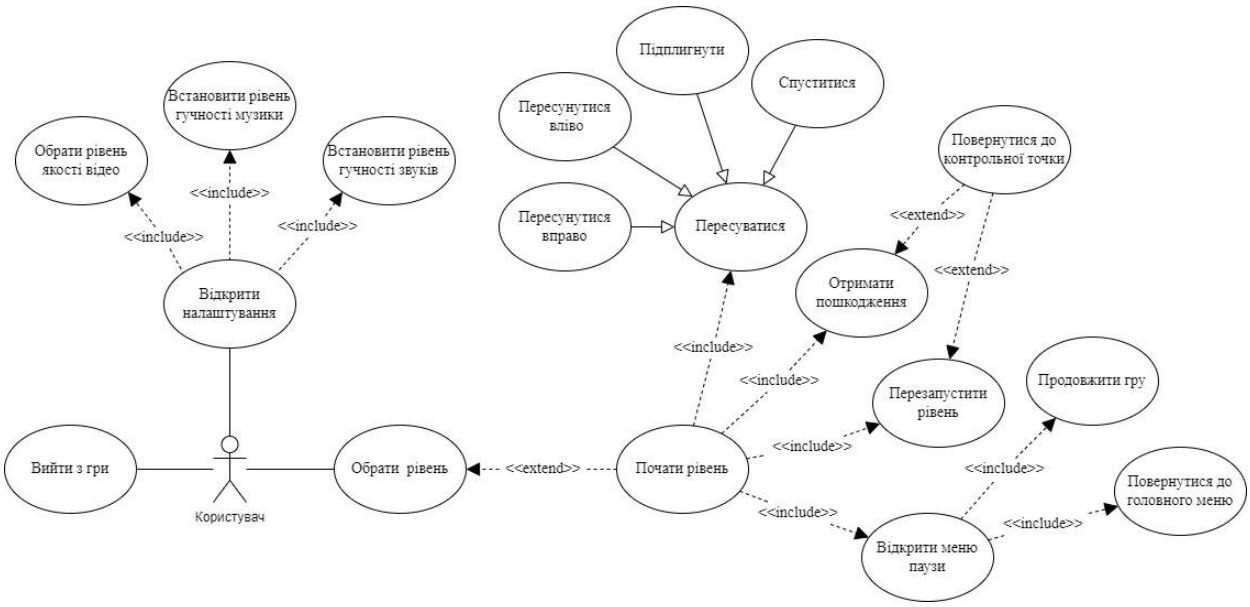


Рисунок 2.2 – Діаграма варіантів використання для користувача

На наступній діаграмі діяльності (рис. 2.3) відображено поведінку користувача в головному меню додатка.

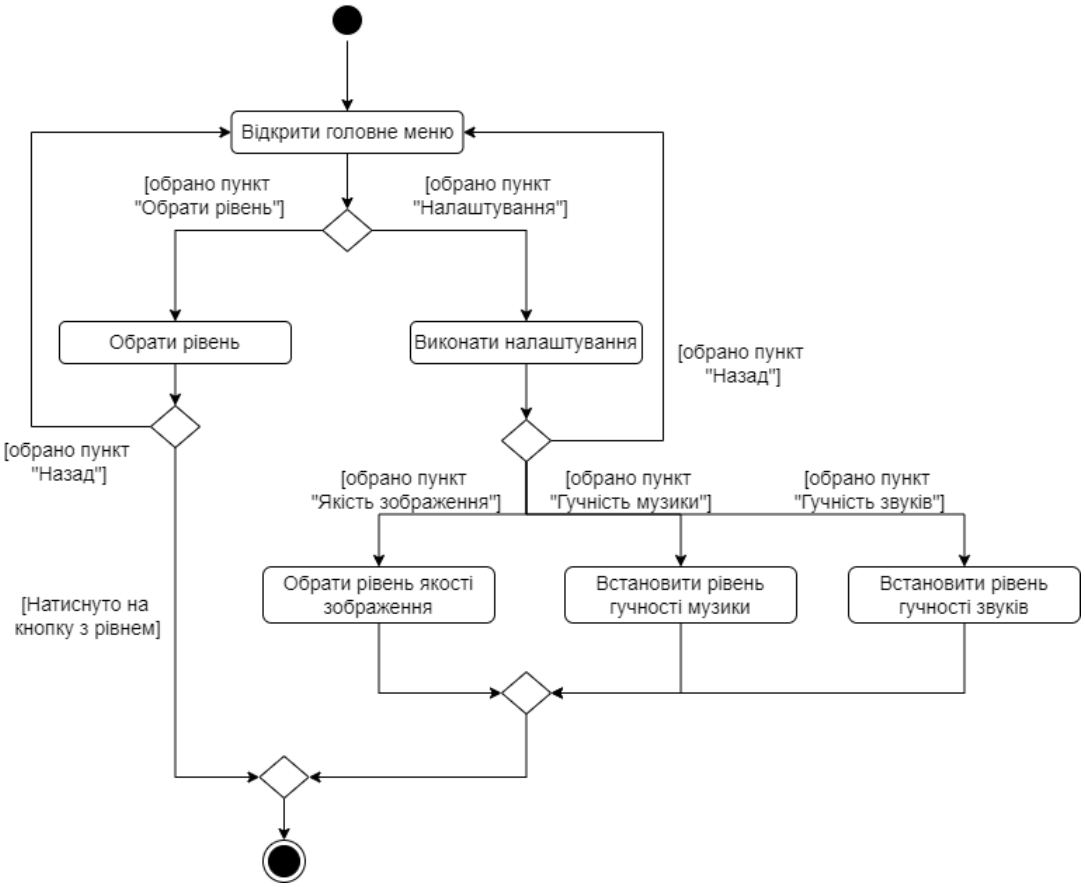


Рисунок 2.3 – Діаграма діяльності користувача в головному меню

На наступній діаграмі діяльності (рис. 2.4) відображено поведінку користувача на ігровому рівні.

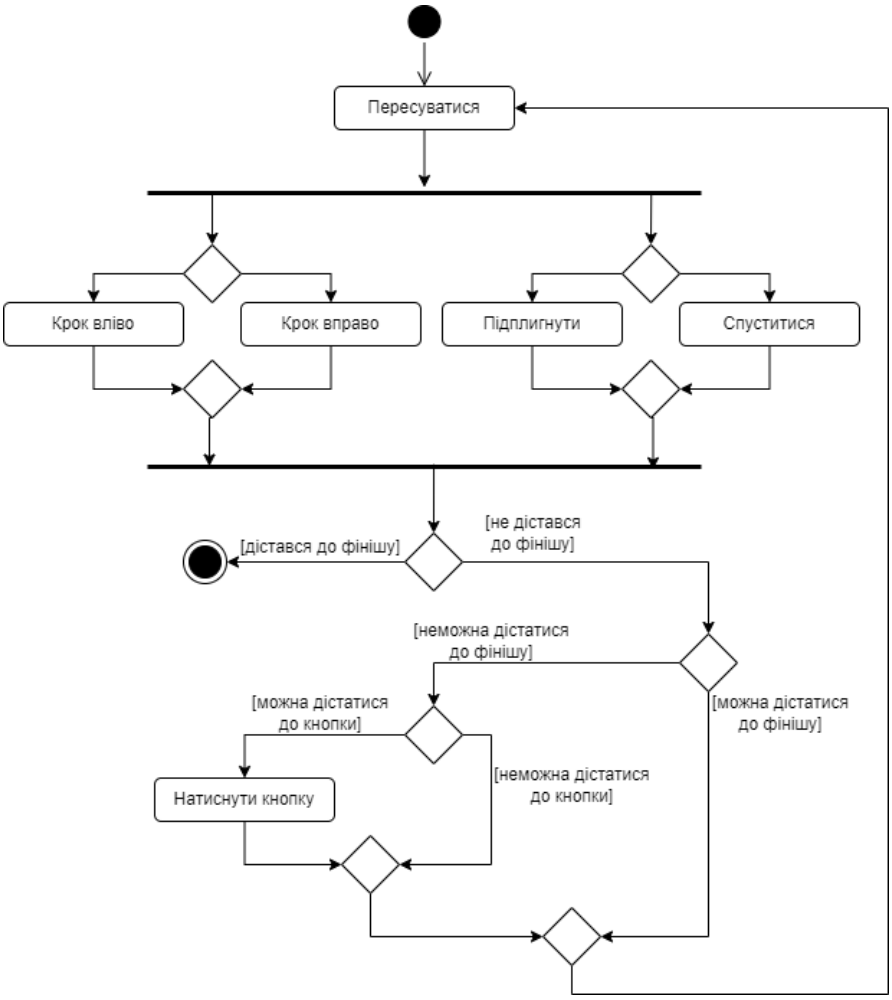


Рисунок 2.4 – Діаграма діяльності користувача на ігровому рівні

На наступній діаграмі послідовності (рис. 2.5) відображено процес успішного вибору ігрового рівня та об’єкти, які беруть безпосередню участь у взаємодії.

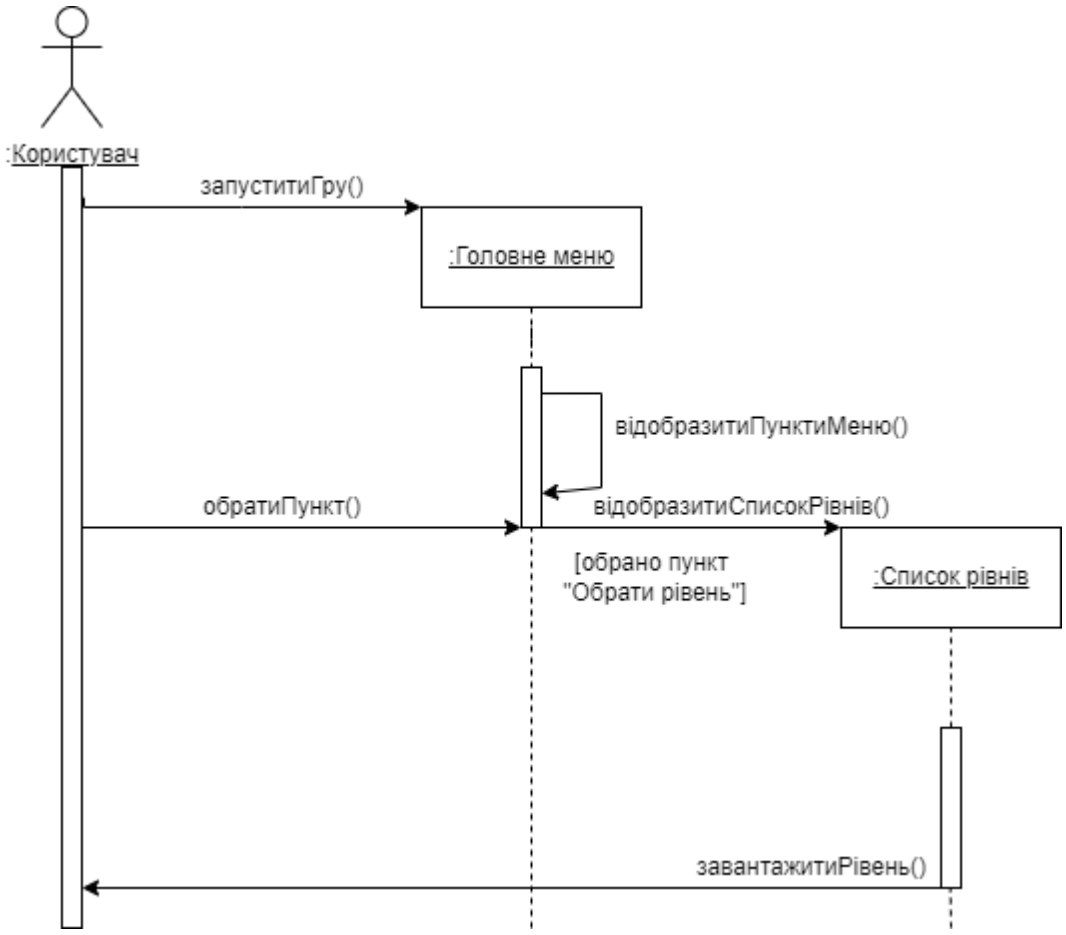


Рисунок 2.5 – Діаграма послідовності вибору ігрового рівня

В програмному застосунку реалізовано збереження користувацьких налаштувань та можливість їх завантаження під час запуску додатку. Якщо додаток запускається вперше, будуть застосовані налаштування за замовчуванням. Діаграма станів роботи даного модуля зображена на рисунку 2.6.

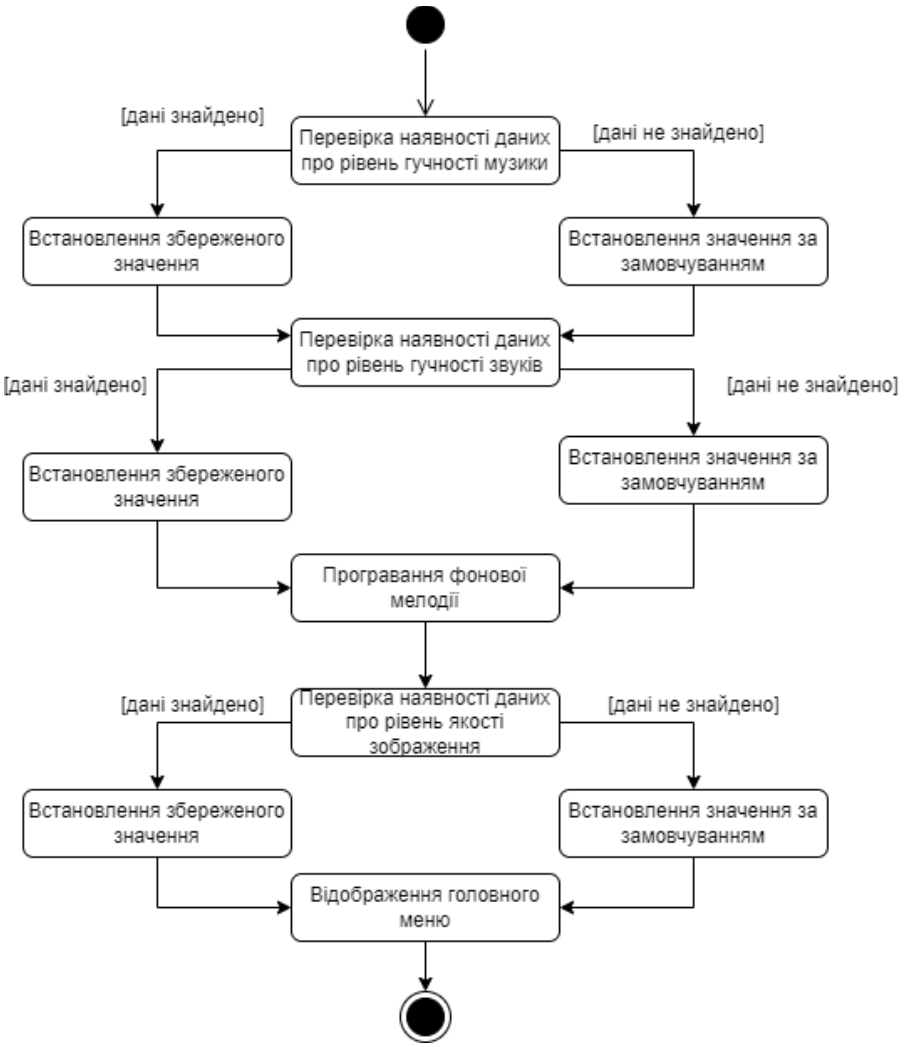


Рисунок 2.6 – Діаграма станів застосування користувацьких налаштувань

2.3 Проєктування людино-машинного інтерфейсу.

Схематичні начерки інтерфейсу основних вікон побудовано за допомогою онлайн інструменту Draw.io. Таким чином спроєктовано головне меню (рис. 2.7), меню налаштувань (рис. 2.8), меню вибору рівня (рис. 2.9), меню паузи (рис. 2.10).

Рисунок 2.7 – Головне меню

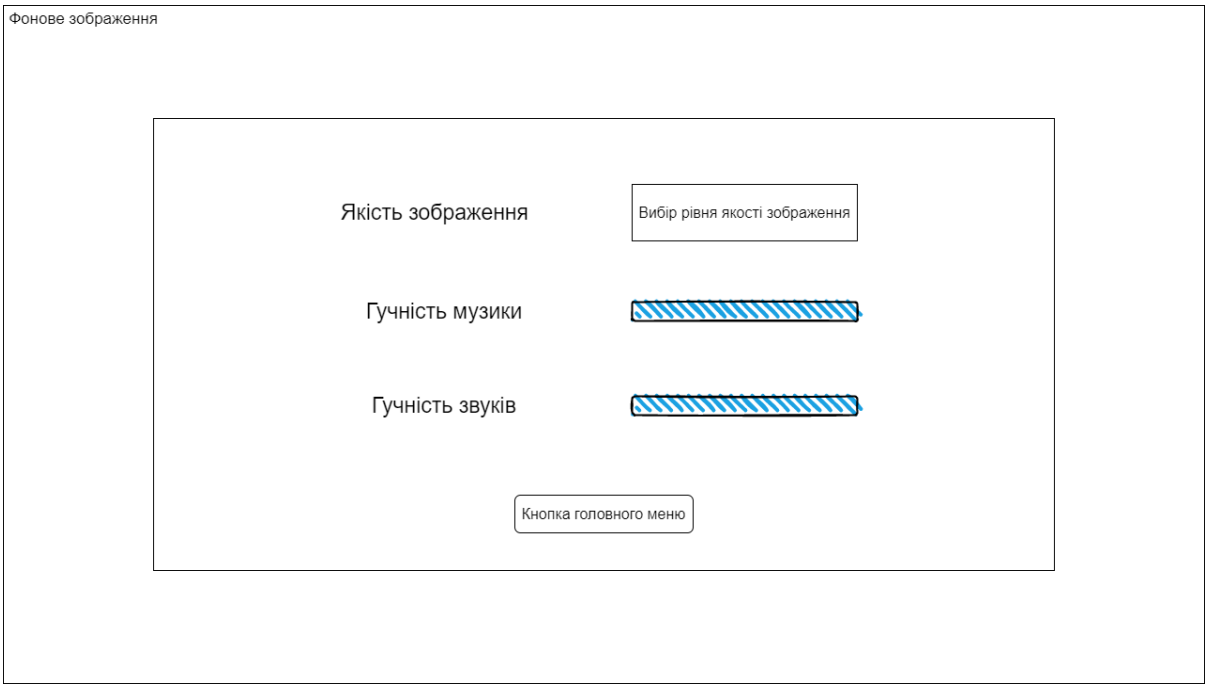


Рисунок 2.8 – Меню налаштувань

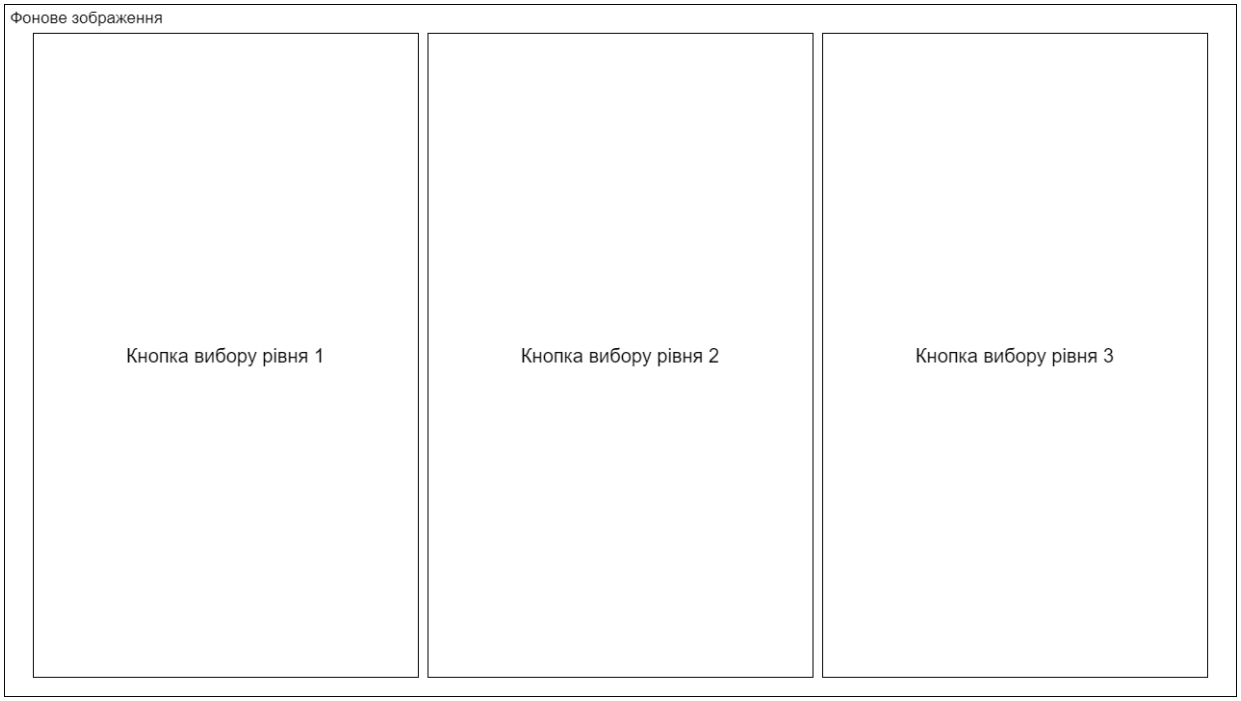


Рисунок 2.9 – Меню вибору рівня

Рисунок 2.10 – Меню паузи

2.4 Засоби розробки

Для створення ігрових додатків існує безліч допоміжних інструментів розробки. В залежності від задачі, необхідно мати відповідний інструмент. Під час розробки необхідними є:

- ігровий рушій;
- текстовий редактор;
- графічний редактор;
- цифрова звукова робоча станція.

2.4.1 Використаний ігровий рушій

Ігровий рушій – це базове програмне забезпечення гри, яке розраховане на повторне на використання та розширення, через що його розглядають як основу для розробки. Зазвичай в ньому вже закладено можливість реалізації:

- фізики;
- керування;
- рендеру;
- виявлення колізій;

Unity [20-21]– кросплатформене середовище розробки комп'ютерних ігор. Unity дозволяє створювати програми, що працюють під більш ніж 20 різними операційними системами, що включають персональні комп'ютери, ігрові консолі, мобільні пристрої, інтернет-програми та інші.

Редактор Unity має простий інтерфейс, який легко налаштовувати, що складається з різних вікон, завдяки чому можна проводити налагодження гри прямо в редакторі. Двигун підтримує мову C#. Розрахунки фізики здійснює фізичний двигун PhysX від NVIDIA.

Проєкт Unity ділиться на сцени – окремі файли, що містять свій набір об'єктів, сценаріїв, і налаштувань. Сцени можуть містити як, об'єкти, і порожні ігрові об'єкти — об'єкти, які не мають моделі. Об'єкти, у свою чергу, містять набори компонентів, з якими і взаємодіють скрипти. Також об'єкти мають

назву, може бути тег і шар, на якому він повинен відображатися. Так, у будь-якого об'єкта на сцені обов'язково присутній компонент Transform – він зберігає координати розташування, повороту і розмірів об'єкта по всіх трьох осях. У об'єктів з видимою геометрією також є компонент Mesh Renderer, що робить модель об'єкта видимою. До об'єктів можна застосовувати колізії.

У редакторі є система успадкування об'єктів: дочірні об'єкти будуть повторювати всі зміни позиції, повороту та масштабу батьківського об'єкта. Скрипти у редакторі прикріплюються до об'єктів як окремі компоненти.

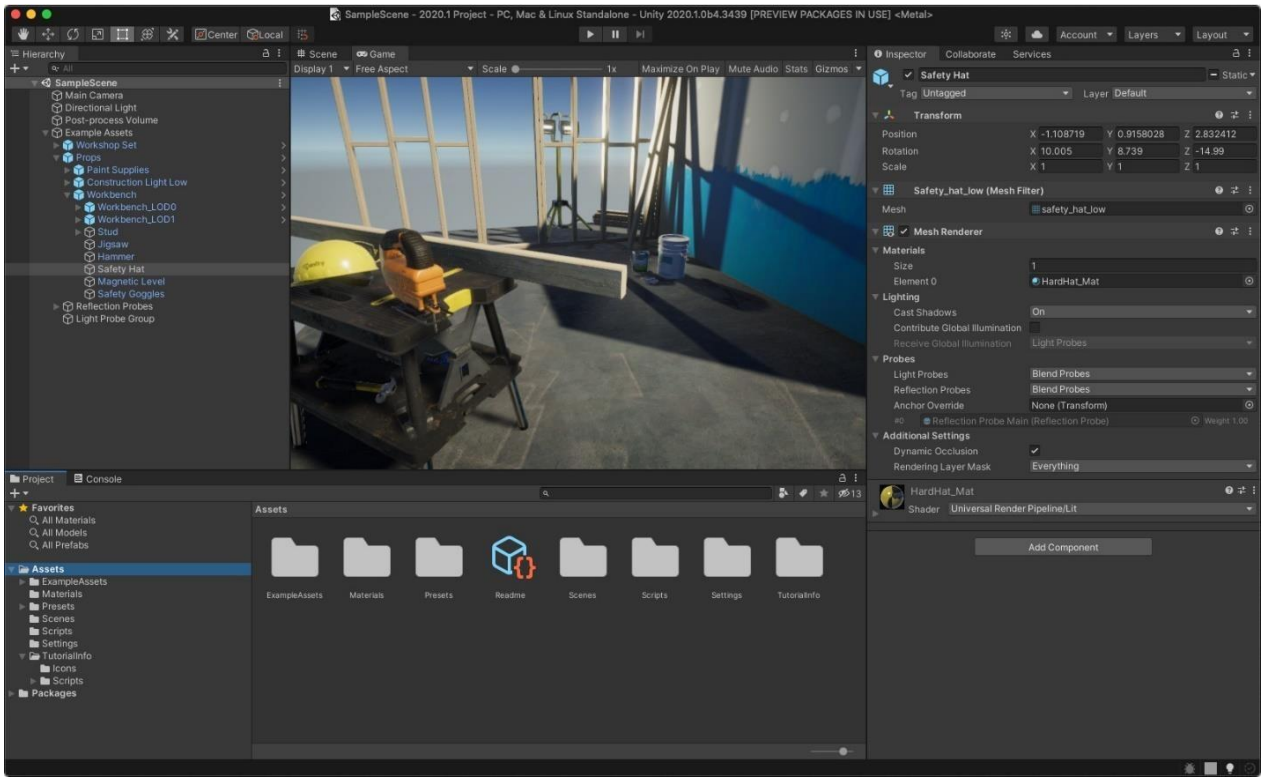


Рисунок 2.11 – Графічний інтерфейс ігрового рушія «Unity»

2.4.2 Обґрунтування вибору середовища розробки та мови програмування

Оскільки ігровий рушій Unity є більш підходящим для виконання задач даної роботи, необхідно обрати середу розробки для написання коду на мові C# [22-23]. Найкращим вибором буде Visual Studio, оскільки дана середа розробки зручно інтегрується з Unity, підсвічуючи синтаксис та

впроваджуючи підказки. На рисунку 2.12 зображено стандартно-згенерований код для сценарію Unity в середі розробки Visual Studio.

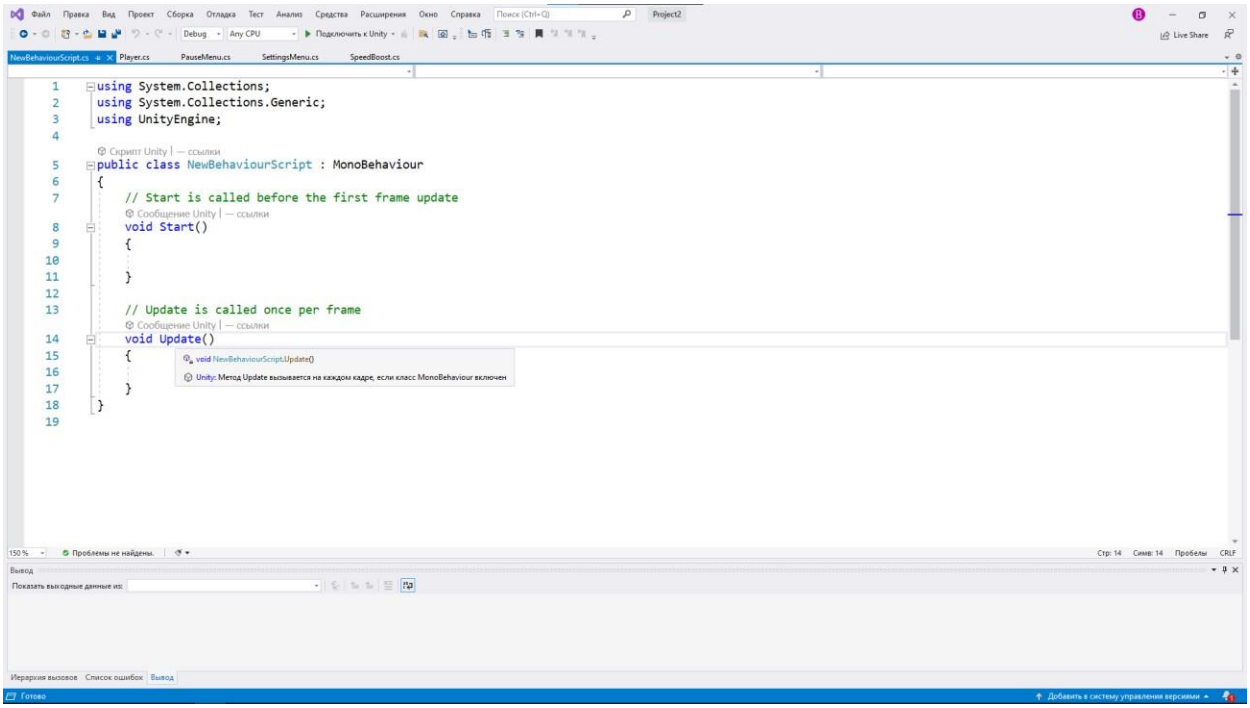


Рисунок 2.12 – Стандартно-згенерований код

C# [24-25] – об'єктно-орієнтована та типобезпечна мова програмування. C# відноситься до сімейства мов C. Вона надає мовні конструкції для підтримки об'єктно-орієнтованої концепції розробки. Завдяки цьому C# підходить для створення та застосування програмних компонентів.

2.4.3 Визначення додаткових інструментів розробки

Для роботи з аудіофайлами та растровими зображеннями можна використовувати будь-яке програмне забезпечення, яке може експортувати файли в форматах які розпізнає ігровий двигун Unity.

2.5 Формування концепт-документу проєкту.

На основі розглянутого матеріалу було сформовано наступний концепт-документ.

Жанр: платформер.

Мінімалістичний графічний дизайн.

Цільова аудиторія: досвідчені гравці.

Ігровий процес: підконтрольний гравцю об’єкт – куля. Мета – потрапити з точки А в точку Б. Спосіб пересування – плавний з інерцією. Різні типи платформ: пружні, прискорюючі, рухомі.

Три рівні з різними механіками, побудовані за принципом:

1. простір для освоєння механіки рівня з перешкодами. Камера слідкує за гравцем;
2. фінальна частина – бос. Бос в даному випадку – це особливий набір поведінкових алгоритмів які перешкоджають гравцю дістатися фінішу. Камера фіксується в позиції поодаль для широкого огляду кінцевої частини рівня.

Червоний рівень – платформи з прискоренням та пружні. Нерухомий бос в верхній середній частині екрана який змінює чи переставляє платформи. Бос намагається скинути гравця в смертельну зону. Графічне оформлення в червоних тонах з швидкою динамічною музикою на фоні.

Зелений рівень – рухомі платформи, пазл. На рівні присутні кнопки, рухомі об’єкти. Нерухомий бос в верхній середній частині екрана, для подолання якого необхідно активувати певні кнопки. Графічне оформлення в зелених тонах із спокійною музикою на фоні.

Синій рівень – велика кількість ворожих рухомих об’єктів. Помірна швидкість ігрового процес. Для проходження рівня необхідно активувати три кнопки, розміщені в певних місцях рівня. Графічне оформлення в синіх тонах з помірною динамічною музикою на фоні.

Строк завершення: 01.06.2022.

Технічні аспекти.

Ігровий рушій: Unity.

Текстовий редактор: Microsoft Visual Studio.

Графічний редактор: Adobe Photoshop.

Цифрова звукова робоча станція: Reaper.

Таблиця 2.1 – Список ризиків

Номер: 1	**Категорія**: Технологічний
Причина: Відсутність досвіду розробки подібних проєктів	**Симптоми**: Немає остаточного бачення проєкту
Наслідки: Незбалансованість рівня складності	**Вплив**: Погіршення ігрового досвіду користувача
Вірогідність: Середня	**Ступінь впливу**: Середня
Близькість: Нескоро	**Ранг**: B

Номер: 2	**Категорія**: Стратегічний
Причина: Непередбачена кількість багів	**Симптоми**: Відведено замало часу на тестування
Наслідки: Перенесення дати релізу	**Вплив**: Некоректна робота програмного додатку
Вірогідність: Висока	**Ступінь впливу**: Висока
Близькість: Скоро	**Ранг**: B

Номер: 3	**Категорія**: Військовий
Причина: Проведення військових дій на території країни	**Симптоми**: Порушення комунікації
Наслідки: припинення роботи над проєктом	**Вплив**: Закриття проєкту
Вірогідність: Висока	**Ступінь впливу**: Висока
Близькість: Дуже скоро	**Ранг**: A

Номер: 4	**Категорія**: Епідеміологічний
Причина: Виникнення епідемії хвороби COVID-19	**Симптоми**: Підвищена кількість хворих серед робітників
Наслідки: Низька продуктивність розробки	**Вплив**: Збільшення термінів розробки
Вірогідність: Низька	**Ступінь впливу**: Висока
Близькість: Дуже скоро	**Ранг**: A

2.6 Висновки за розділом

У даному розділі було сформовано перелік функцій програмного засобу, спроєктовано структуру системи та людино-машинний інтерфейс, обрано та засоби розробки та обґрунтовано доцільність їх використання.

Для подальшої розробки комп’ютерного ігрового додатку обрано ігровий рушій Unity, середу розробки Microsoft Visual Studio з використанням мови програмування C#.

На основі розглянутої інформації, сформовано концепт-документ в якому описано загальні положення щодо ігрового процесу та засобів розробки. В ньому ж приведено перелік потенційних ризиків для проєкту.

3 РОЗРОБКА ПРОГРАМНОГО ДОДАТКУ ОДНОКОРИСТУВАЦЬКОЇ ГРИ-ПЛАТФОРМЕРА

3.1 Підготовка матеріалів для додатку

Для створення нового проєкту необхідно відкрити інструмент для керування проєктами Unity Hub, натиснути кнопку «New» (рис. 3.1) та у відкритому вікні вказати шаблон проєкту, його назву та розташування (рис. 3.2).

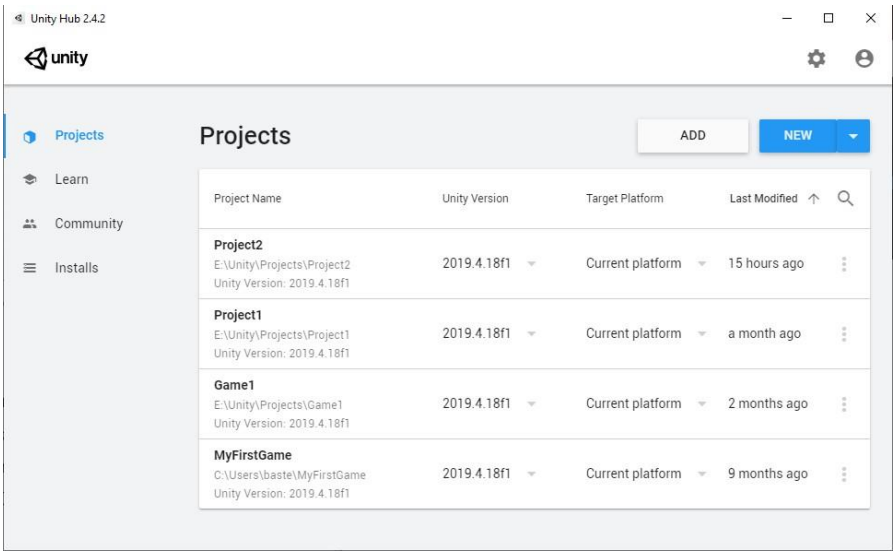


Рисунок 3.1 – Керування проєктами в Unity Hub

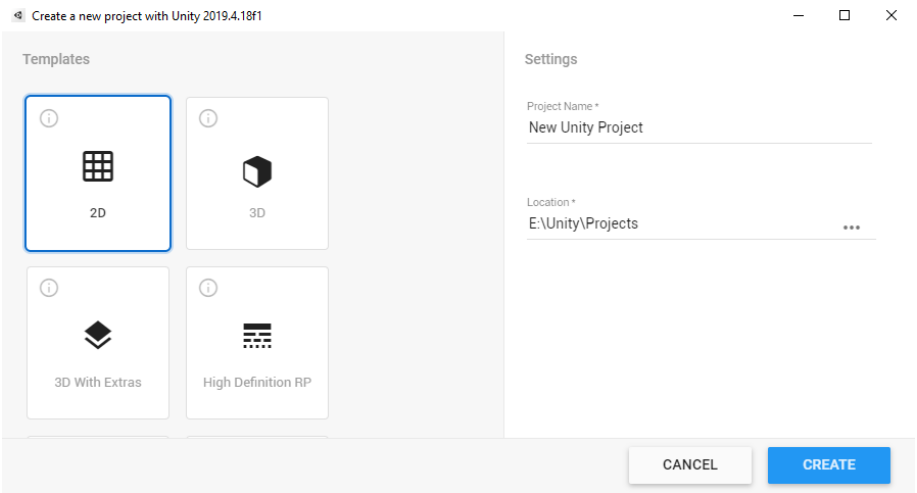


Рисунок 3.2 – Створення нового проєкту в Unity

Для розробки програмного запису та тестування розроблених класів на практиці необхідно мати наступні матеріали:

- спрайти для побудови рівня;
- спрайт для головного героя;
- фонові зображення для рівнів та меню;
- звукові ефекти.

Для побудови рівнів, за допомогою програми Adobe Photoshop, створено три одинакові набори фігур, які відрізняються за кольором, які зображено на рисунках В.1 – набір спрайтів для побудови першого рівня, В.2 – набір спрайтів для побудови другого рівня, В.3 – набір спрайтів для побудови третього рівня. Для відображення головного героя використано спрайт, зображений на рисунку В.4 – спрайт головного героя. Також в програмі Adobe Photoshop створено фонові зображення для головного меню та рівнів які зображено на рисунку В.5 – фонове зображення головного меню, В.6 – фонове зображення для першого рівня, В.7 – фонове зображення для другого рівня, В.8 – фонове зображення для третього рівня,. Всі ці елементи наведено в додатку В.

Головне меню, як і кожен рівень, має своє музикальне супроводження. Для звукового супроводу дій головного героя в програмі Reaper було оброблено такі звуки:

- звук підплигування;
- звук приземлення;
- звук прискорюючої поверхні;
- звук отримання пошкодження;

3.2 Побудова ігрового рівня

Ігровий рівень в редакторі Unity подається у вигляді однієї або декількох сцен, на кожній з яких присутня ієрархія ігрових об’єктів із власними наборами компонентів та їх властивостей.

3.2.1 Налаштування об’єкта камера

Спочатку створено нову сцену, на якій за замовчуванням присутній об’єкт «Main Camera» з компонентом Camera та Audio Listener (рис. 3.3).

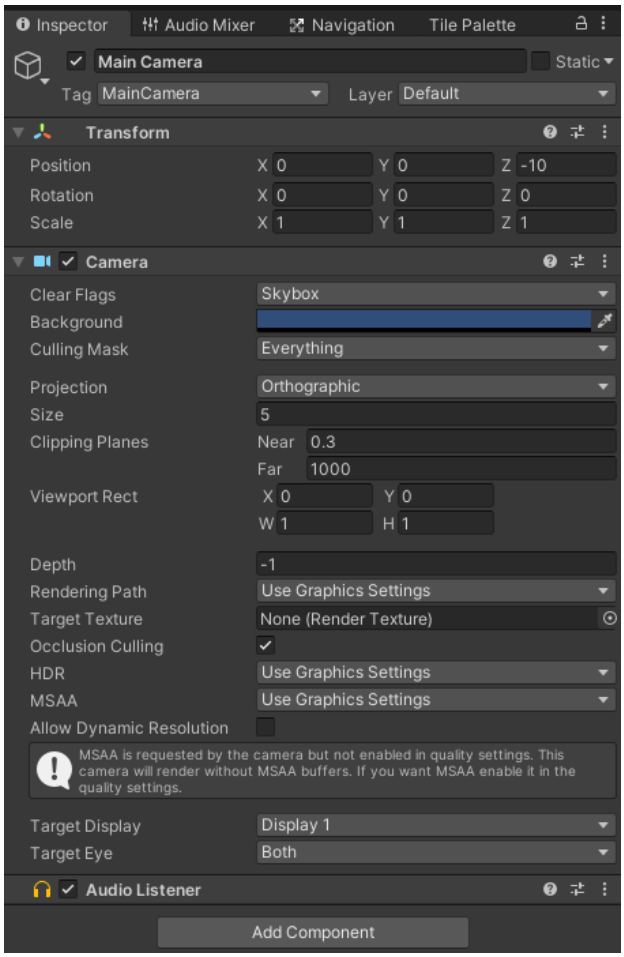


Рисунок 3.3 – Компоненти об’єкта «Main Camera»

Camera – це пристрій, який знімає і показують сцену очами користувача.

Audio Listener – отримує вхід від будь-якого джерела звуку в сцені та відтворює звуки через динаміки комп’ютера.

Для керування камерою на ігровому рівні, створено клас CameraControl.cs та додано його як компонент об’єкта Main Camera. Цей скрипт реалізує плавне пересування об’єкта Main Camera вслід за головним героєм. Цей клас має публічні поля:

– Transform player – компонента Transform ігрового об’єкта головного героя;

– Transform standpoint – компонента Transform ігрового об’єкта на якому фіксується позиція камери;

– Transform cameraCheck – компонента Transform ігрового об’єкта за допомогою якого перевіряється тип поведінки камери;

– float minZoom – мінімальне віддалення;

– float maxZoom – максимальне віддалення камери;

– float cameraSpeed – швидкість пересування камери.

Приватні поля:

– bool isFixed – поведінка камери;

– Vector3 position – вектор довжиною 3.

У методі подій Awake(), який викликається до будь-яких методів та після того як префаб завантажено на сцену, виконується перевірка, чи пусте поле player. Якщо так, то виконуються пошук об’єкта з типом Player та присвоєння його компоненти transform полю player.

У методі подій Update(), який викликається один раз за кадр, викликається метод FixCamera() якщо player знаходиться правіше від cameraCheck. Інакше викликається метод Follow().

Метод Follow() реалізує переміщення камери за головним героєм зі швидкістю cameraSpeed та її віддаленням на дистанцію minZoom.

Метод FixCamera() реалізовує фіксування позиції камери в точці standpoint з віддаленням на дистанцію maxZoom.

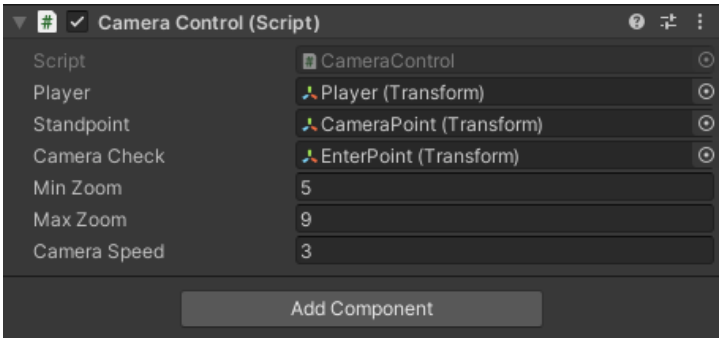


Рисунок 3.4 - CameraControl.cs як компонент об’єкта Main Camera

3.2.2 Налаштування об’єкта полотно

Для відображення UI елементів створено ігровий об’єкт Canvas - абстрактний простір, в якому проводиться налаштування та відображення UI. На ігровому рівні цей об’єкт використовується для відображення фонового зображення та меню паузи.

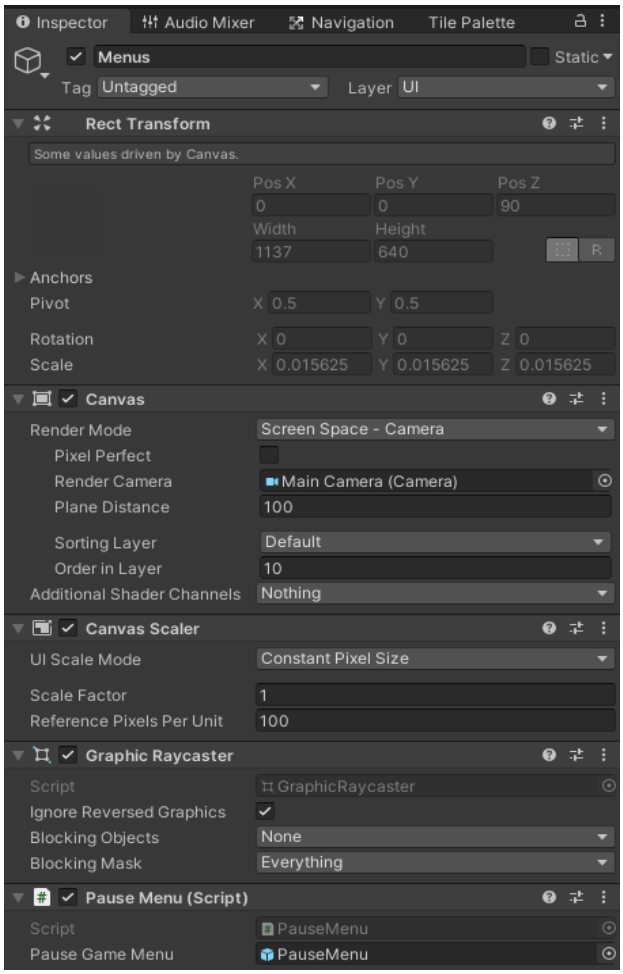


Рисунок 3.5 – Компоненти об’єкта «Menus»

Функціонал меню паузи реалізовано в класі PauseMenu.cs. Він реалізовує показ даного меню з можливостями продовжити гру або виходу в головне меню. Цей клас має одне публічне поле GameObject pauseGameMenu – ігровий об’єкт меню, що потрібно показати; та приватне поле bool isPaused – поточний стан меню.

У методі подій Update() виконується перевірка, чи натиснута кнопка «Escape». Якщо вона натиснута, перевіряється поточний стан isPaused. Якщо зараз відкрите меню паузи, то викликається метод Resume(), інакше Pause().

Метод Resume() реалізовує відновлення гри. В ньому зупиняється програвання поточної фонової музики, знімається з паузи фонова мелодія ігрового рівня, саме меню деактивується та відновлюється нормальний плин ігрового часу.

Метод Pause() реалізовує зупинку гри. В ньому запускається програвання поточної фонової музики, ставиться на паузи фонова мелодія ігрового рівня, саме меню активується та зупиняється плин ігрового часу.

Метод LoadMenu() реалізовує завантаження сцени головного меню.

На канвасі меню паузи, засобами Unity, створено дві кнопки: ResumeButton, яка викликає метод Resume(), та ToMenuButton, яка викликає метод LoadMenu().

3.2.3 Створення ігрового простору

Побудову двовимірного рівня можна реалізувати двома способами. Перший, це створити префаби всіх необхідних елементів рівня таких, як підлога, стіни, платформи та інші елементи, і розташовувати їх екземпляри на сцені.

Префаб – це особливий тип асетів Unity (компоненти які можуть представляти собою скрипти, звуки, графіку), який дозволяє зберігати ввесь ігровий об’єкт зі всіма компонентами та значеннями властивостей. Він є шаблоном для створення екземплярів збереженого об’єкта. Кожен екземпляр префаба можна налаштовувати окремо.

Другий спосіб, це використати компонент Tilemap – це система, яка зберігає та оброблює Tile Assets для створення 2D-рівнів. Він передає необхідну інформацію з плиток, розміщених на ньому, до інших пов’язаних компонентів, таких як Tilemap Renderer, компонент який відповідає за відображення плиток на Tilemap, і Tilemap Collider 2D, компонент який відповідає за створення колайдера для кожної плитки на сцені. Колайдер – це невидима фігура для обробки фізичних зіткнень об’єктів.

У роботі використано обидва способи: перший для створення меню паузи та об’єкта «куля»; другий для створення статичного оточення на сцені.

Для створення статичного оточення за допомогою Tilemap, Створено нові палітри для інструменту Tile Palette для кожного рівня і додано до них підготовані набори фігур, які було нарізано на окремі спрайти розміром 100x100 пікселів (рис. 3.6). За допомогою цього інструменту можна виділити один чи декілька спрайтів та використовувати їх як кисті для малювання.

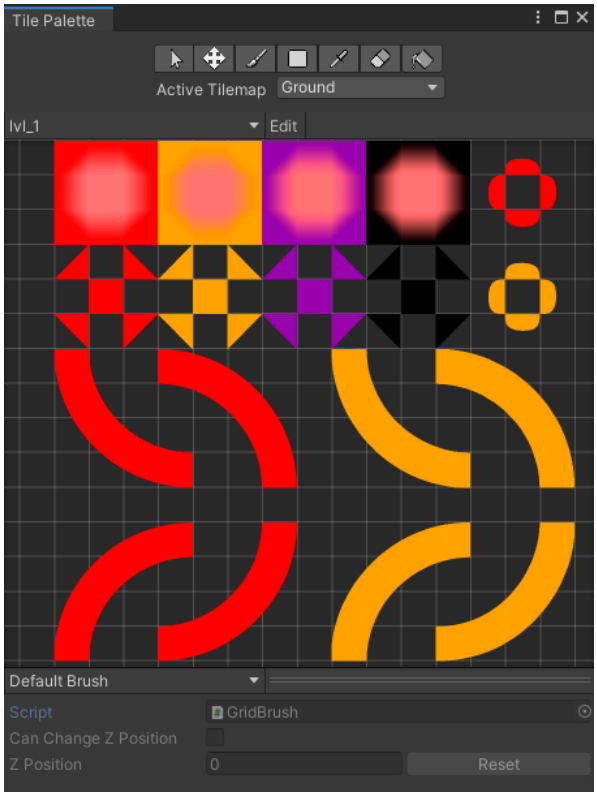


Рисунок 3.6 – Палітра для першого рівня

Оскільки на рівні передбачено поверхні декількох типів, то було створено чотири об’єкта Tilemap, яким присвоєно відповідні теги та шари. До кожного об’єкта Tilemap додано компоненти Tilemap Collider 2D, Composite Collider 2D, Rigitbody 2D.

Composite Collider 2D використовує вершини колайдерів і об’єднує їх разом в одну фігуру.

Rigitbody 2D передає об’єкт під контроль фізичного двигуна. Для всіх об’єктів Tilemap тип тіла встановлено Static, що фіксує їх у просторі.

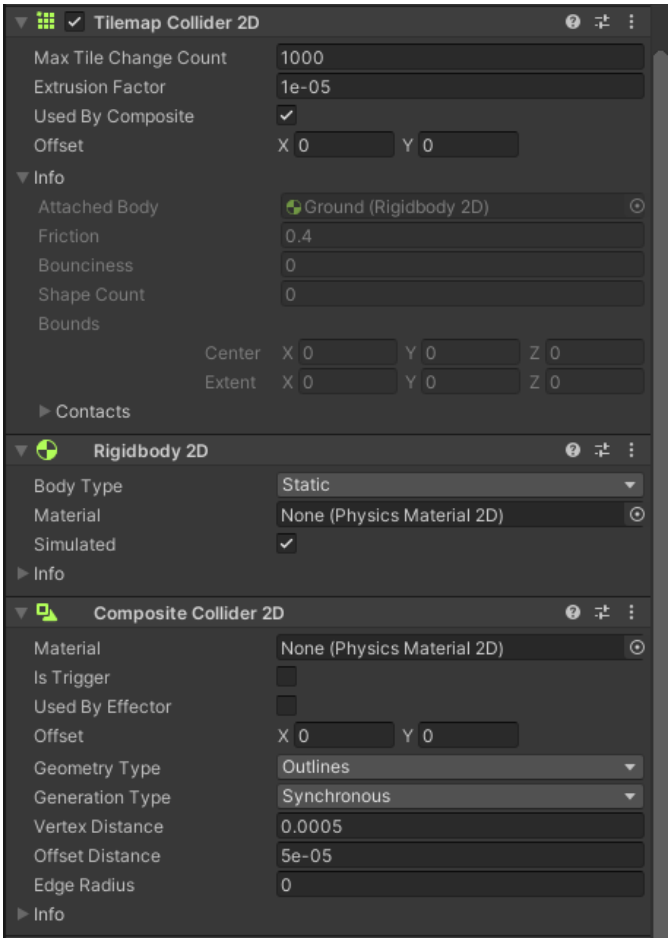


Рисунок 3.7 – Додані до Tilemap компоненти

Поверхня першого типу – Ground. Вона звичайною поверхнею для пересування, з якої й будується основна частина ігрового рівня.

Поверхня другого типу – DeathZone. Це поверхня, при контакті з якою головний герой отримую пошкодження.

Поверхня третього типу – SpeedBoost. Це поверхня, при контакті з якою гравець отримує прискорення. Прискорення реалізовано в класі SpeedBoost.cs.

Цей клас має публічні поля:

– float Boost – значення прискорення;

– float maxSpeed – значення максимальної швидкості;

Приватні поля:

– bool isActive – стан поверхні;

– Rigidbody2D rb – компонент Rigidbody2D об’єкта, який ввійшов у колізію.

У методі подій Update() перевіряється isActive, та якщо isActive = true, то викликається метод Activate().

Метод Activate() реалізує прискорення. Якщо поточна швидкість менша ніж maxSpeed, виконується прискорення.

У методі подій фізики OnCollisionEnter2D(Collision2D collision), який приймає інформацію про об’єкт колізії у вигляді параметра типу Collision2D, починає програватися звук прискорення, isActive присвоюється значення true, rb присвоюється значення компонента Rigidbody2D об’єкта колізії.

У методі подій фізики OnCollisionExit2D (Collision2D collision), який приймає інформацію про об’єкт колізії у вигляді параметра типу Collision2D, зупиняється програвання звуку прискорення, isActive присвоюється значення false.

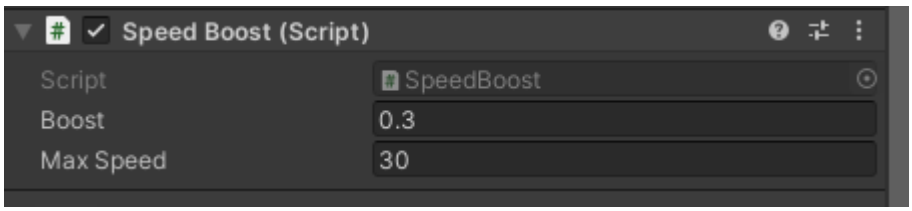


Рисунок 3.8 – SpeedBoost.cs як компонент об’єкта SpeedBoost

Поверхня третього типу – JumpBoost. Це пружна поверхня, з матеріалом Trampline присвоєному компоненті Composite Collider 2D.

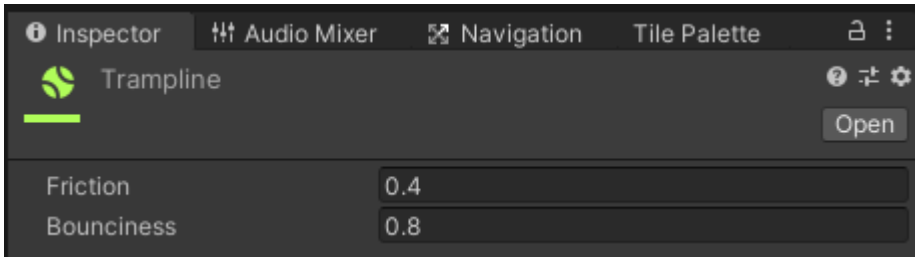


Рисунок 3.9 – Матеріал Trampline

Окремо взяті спрайти модифіковано, для створення рухомих платформ. Рух окремої платформи реалізовано в класі PlatformMove.cs.

Публічні поля:

– float verticalRange – максимальне/мінімальне відхилення від стартової позиції по вертикалі;

– float horizontalRange – максимальне/мінімальне відхилення від стартової позиції по горизонталі;

– float moveSpeed – швидкість руху;

– bool rotateClockwise – активація руху навколо центра платформи за годинниковою стрілкою;

– bool rotateCounterclockwise – активація руху навколо центра платформи проти годинникової стрілки;

– bool oneMove – активація руху протягом лише одного циклу

Приватні поля:

– bool movingRight – рух платформи праворуч;

– bool movingUp - рух платформи вгору;

– bool moveVertical – рух платформи вертикально;

– bool moveHorizontal – рух платформи горизонтально;

– Quaternion startRotation – стартовий кут повороту;

– Vector3 startPosition – стартова позиція;

Скрипт має публічний метод Activate() для активації виконання та публічний метод Deactivate() для зупинки виконання.

У методі подій Awake() ініціалізуються початкові значення startPosition, startPosition, moveVertical, moveHorizontal.

У методі подій FixedUpdate(), який на відміну від Update() викликається у відповідності з надійним незалежним від частоти кадрів таймером, перевіряються умови руху платформи та викликаються відповідні методи.

Метод MovePlatformVertical(speed, range) приймає параметри швидкості та діапазону руху, реалізовує рух по вертикалі в межах від -range до range.

Метод MovePlatformHorizontal(speed, range) приймає параметри швидкості та діапазону руху, реалізовує рух по горизонталі в межах від -range до range.

Метод RotatePlatformClockwise (speed) приймає параметр швидкості та реалізує рух навколо центра платформи за годинниковою стрілкою.

Метод RotatePlatformCounterclockwise (speed) приймає параметр швидкості та реалізує рух навколо центра платформи проти годинникової стрілки.

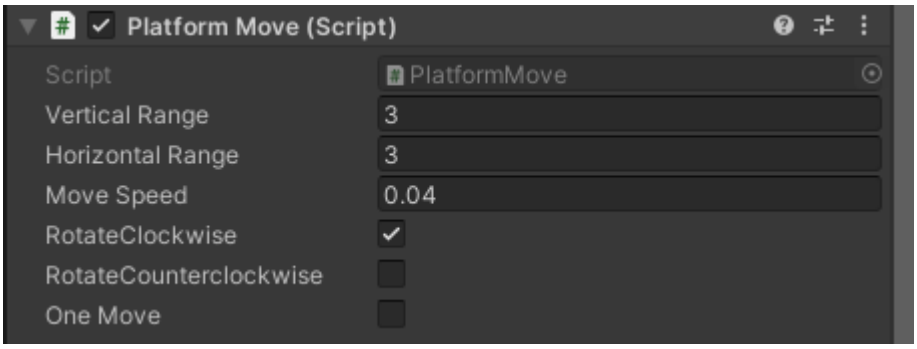


Рисунок 3.10 – PlatformMove.cs як компонент рухомої платформи

Клас PlatformToStart.cs реалізовує повернення об’єкта до стартової позиції. Використовується в комбінації з PlatformMove.cs. Має публічне поле float speed, яке описує швидкість руху, та приватні поля з початковими кутом нахилу та позицією Quaternion startRotation і Vector3 startPosition.

У методі подій Awake() ініціалізуються початкові значення startPosition, startPosition.

У методі подій Update() перевіряється активність компоненти PlatformMove, та в разі її не активності викликається метод MoveToStart (), який реалізує повернення об’єкта в початкову позицію.

Для завершення рівня необхідно дістатися фінішу, який відображено як два квадрати з анімацією обертання в протилежні сторони (рис. В.5). В Unity присутній інструмент для створення анімацій, за допомогою якого створено анімації для спрайтів фінішу (рис. 3.11-3.12).

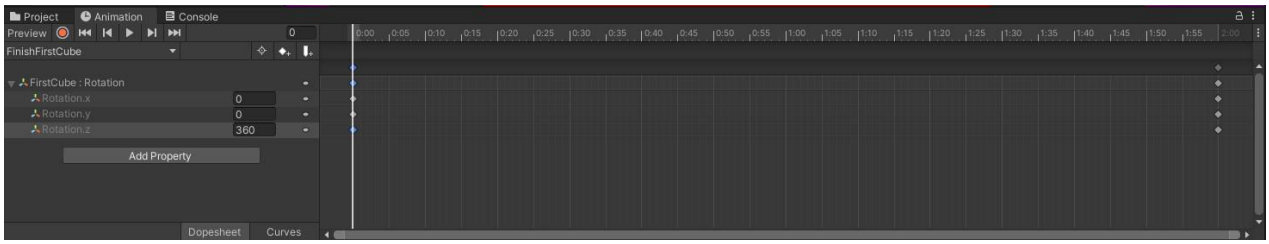


Рисунок 3.11 – Створення ключових кадрів анімації

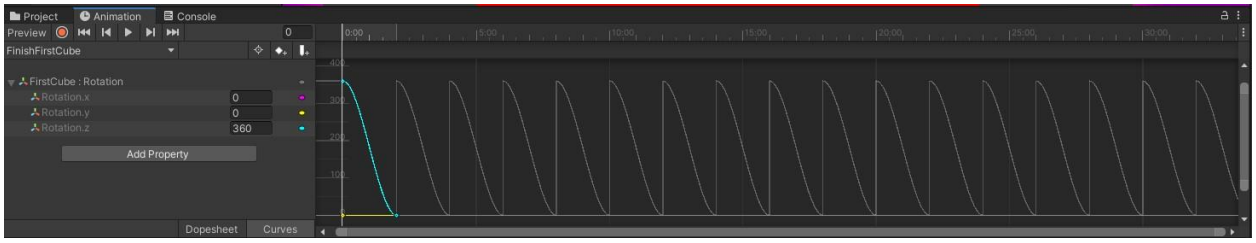


Рисунок 3.12 – Налаштування плавності переходу анімації

Компонент BoxCollider2D даного об’єкта відмічено як тригер, тож він не є фізичним об’єктом. При колізії головного героя та фінішу з’являється фінішне меню з надписом про завершення рівню та кнопкою переходу до головного меню, що реалізовано в класі FinishMenu.cs. В цьому класі присутнє публічне поле GameObject finishMenu яке містить об’єкт фінішного меню.

Метод LoadMenu() реалізовує завантаження головного меню та викликається кнопкою на об’єкті finishMenu.

У методі подій фізики OnTriggerEnter2D(Callider2D collision) реалізовано активацію фінішного меню при умові, що значення тегу параметра collision відповідає значенню «Player».

На ігрових рівнях присутні інтерактивні кнопки та плити, які активуються при входження тригеру. Кнопки реалізовані в за допомогою класу ButtonNew.cs. Він має приватне поле bool isPressed, для відстеження стану кнопки, публічне поле UnityEvent OnPressed для створення нової події.

Метод ToPress() присвоює полю isPressed значення «true».

У методі подій фізики OnTriggerEnter2D(Callider2D collision), якщо значення тегу об’єкта колізії рівне «Player», виконується програвання анімації натискання кнопки, присвоєння полю isPressed значення «true» та виклик методу Invoke() для OnPressed. Дії методу Invoke() описуються в інтерфейсі Unity як параметри компонента ButtonNew (рис. 3.13).

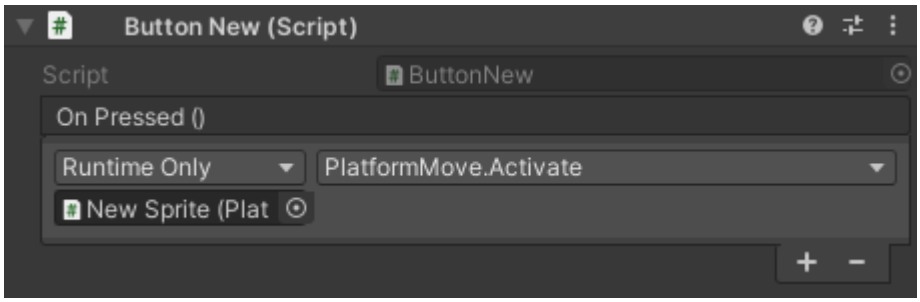


Рисунок 3.13 – ButtonNew.cs як компонент ігрового об’єкта кнопки

Ще одним інтерактивним елементом на рівні є плита. Вона реалізована за допомогою класу PlateNew.cs. Він має приватне поле bool isPressed, для відстеження стану кнопки, публічне поле UnityEvent OnPress та UnityEvent OnLeave для створення нових подій.

Метод ToPress() присвоює полю isPressed значення «true».

У методі подій фізики OnTriggerStay2D (Callider2D collision) виконується програвання анімації натискання плити, присвоєння полю isPressed значення «true» та виклик методу Invoke() для OnPress.

У методі подій фізики OnTriggerExit2D(Callider2D collision) виконується програвання анімації підняття плити, присвоєння полю isPressed значення «false» та виклик методу Invoke() для OnLeave.

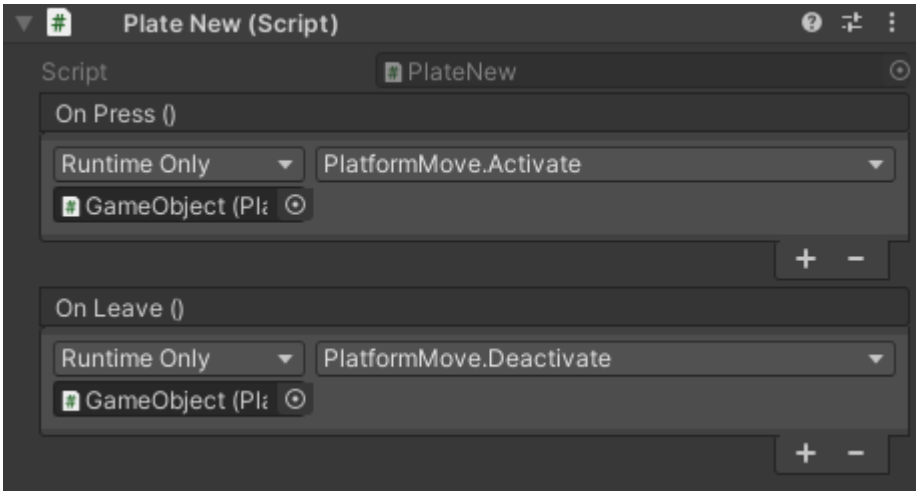


Рисунок 3.14 - PlateNew.cs як компонент ігрового об’єкта плити

У кінцевій частині ігрового рівня побудовано кімнату з фінальним випробуванням. Зазвичай головний герой при отриманні пошкоджень переноситься на ігровий об’єкт Respawn на самому початку рівня, який є точкою в просторі на сцені. При вході до кімнати, об’єкт Respawn змінює свою позицію, що реалізовано в класі MoveRespawn.cs. В цьому класі присутні публічні поля Transform player, що містить компоненту Transform ігрового об’єкта головного героя та Transform cameraCheck, що містить компоненту Transform ігрового об’єкта для перевірки типу поведінки камери.

У методі подій Update() реалізовано перетворення позиції об’єкта Respawn до позиції cameraCheck за умови, що player знаходиться правіше від cameraCheck.

На вході в фінальну кімнату знаходиться пересувна платформа, що блокує вихід, коли головний герой знаходиться в кімнаті, але відкривається, якщо головний герой знаходиться поза нею. Цю поведінку реалізовано в класі Door.cs. Він має публічні поля Transform player, що містить компоненту

Transform головного героя, float bottomPosition, що містить нижню крайню позицію платформи. Приватне поле float startY містить початкове значення позиції платформи по координаті y.

У методі подій Awake() полю startY присвоюється значення поточної позиції платформи.

У методі подій Update() виконується пересування платформи в залежності від позиції головного героя відносно цієї платформи.

У кінцевій кімнаті деяких рівнів знаходиться бос, який може складатися з набору наступних елементів:

– ока, пересування якого реалізовано в класі BossEye.cs;

– руки, поведінку якої реалізовано в класі BossArm.cs;

– квадратів, що вільно пересуваються, поведінку яких реалізовано за допомогою класу PlatformMove();

– об’єкта з постійною генерацією куль, яка реалізована за допомогою класів Bullet.cs, BolletPool.cs, BulletFire.cs.

Клас BossEye.cs має публічні поля:

– Transform target – компонента Transform об’єкта за яким пострібно стежити;

– float speed – швидкість пересування;

Приватні поля:

– bool toMove – стан переміщення;

– Vector3 position – змінна для збереження поточної позиції об’єкта;

– Vector3 standatrPos - змінна для збереження початкової позиції об’єкта;

У методі подій Update() виконується виклик методу Follow(), за умови, що позиція об’єкту не виходить за встановлені координати.

Метод Follow() реалізує переміщення об’єкта в напрямку target.

Для реалізації руху рук було зроблено анімацію удару, яка спрацьовує при входженні об’єкта з тегом «Player» в область тригеру на цій руці. Клас

BossArm.cs має публічне поле GameObject arm, яке містить ігровий об’єкт руку.

У методі подій фізики OnTriggerEnter2D(Callider2D collision), якщо значення тегу об’єкта колізії рівне «Player», виконується програвання анімації удару.

Для генерації куль, їх збереження, патерну поведінки, розроблено класи, структура яких базується на патерні програмування «Одинак» та «Пул об’єктів».

Клас Bullet.cs задає необхідні властивості для кулі. Він має публічні поля float moveSpeed, що містить швидкість кулі, та float stayTime, що містить час активного стану кулі. Приватне поле Vector2 moveDirection, яке містить вуктор довжиною 2 для вказування напрям руху для кулі.

У методі подій OnEnable(), який викликається при активації об’єкта, викликається метод Destroy() через проміжок часу stayTime.

У методі подій Update() виконується переміщення об’єкту зв напрямку moveDirection зі швидкістю moveSpeed.

У методі SetMoveDirection(Vector2 dir) виконується присвоєння значення dir значенню moveDirection.

У методі Destroy() виконується деактивація об’єкта.

У методі подій OnDisable(), який викликається при деактивації об’єкта, зупиняються всі виклики.

Клас BulletPool.cs реалізує пул об'єктів для куль. Він має публічні поля static BulletPool bulletPoolInstance, для отримання статичного посилання на клас, та GameObject pooledBullet, що містить ігровий об’єкт кулю. Приватні поля bool notEnoughtBulletsInPool, відображає наявність об’єктів у пулі, та List<GameObject> bullets, список об'єктів типу GameObject.

У методі подій Awake() значенню bulletPoolInstance присвоюється посилання на поточний екземпляр класу.

У методі подій Start() створюється новий список bullets.

У методі GetBullet(), який повертає GameObject, проводиться перевірка на наявність неактивного об’єкта із списку bullets. Якщо неактивний елемент не знайдено, то створюється копія ігрового об’єкта pooledBullet, яка додається до списку bullets та повертається як результат виконання функції. Інакше повертається неактивний елемент списку bullets.

Клас BulletFire.cs реалізує створення заданої кількості куль, рівномірно розподілених в заданому секторі. Він має публічні поля:

– static BulletFire fire – статичне посилання на клас;

– int bulletAmont – кількість куль за один виклик;

– float timeInterval – часовий інтервал між викликами;

– float startAngle – кут початку сектора;

– float endAngle – кут кінця сектора;

– GameObject bulletPool – пул, з якого потрібно брати кулі;

– Transform player – поточна позиція головного героя;

– Transform triggerPointBotLeft – лівий нижній кут зони початку обстрілу;

– Transform triggerPointTopRight – правий верхній кут зони початку обстрілу;

– bool checkPosition – необхідність проведення перевірки місця знаходження головного героя;

– bool stopFire – необхідність створення нових куль.

У методі подій Awake() значенню fire присвоюється посилання на поточний екземпляр класу.

У методі подій Start() виконується виклик методу Fire() з інтервалом timeInterval.

У методі StopFire() полю stopFire присвоюється значення «false».

У методі CheckPosition(), який повертає bool, перевіряється позиція головного героя відносно встановлених triggerPointBotLeft та triggerPointTopRight. Якщо головний герой знаходиться всередині умовно

визначеного прямокутника, метод повертає «true». Якщо значення поля stopFire рівне «true», то метод поверне «false» в будь-якому випадку.

Метод Fire() реалізує рівномірний розподіл заданої кількості куль у вказаному секторі та задання їм напрямку руху.

Клас FocusOnTarget.cs створено для його використання у комбінації з BulletFire.cs. FocusOnTarget.cs реалізує поворот ігрового об’єкта на кут по відношенню до цілі. Він має публічне поле Transform target, що містить компонент Transform цільового об’єкта.

У методі подій Update() виконується розрахунок кута повороту відносно поточної позиції target.

На сценах присутні ігрові об’єкти які переслідують головного героя. Їх поведінка реалізована в класі Follow.cs. В ньому є публічні поля:

– float speed – швидкість переміщення;

– Transform target – компонент Transform ігрового об’єкта цілі;

– Transform triggerPointBotLeft – лівий нижній кут зони початку руху;

– Transform triggerPointTopRight – правий верхній кут зони початкуУ руху.

методі подій FixedUpdate() викликається метод ToFollow(), в залежності від значення поверненого методом CheclPosition().

Метод ToFollow() реалізовує пересування об’єкта за target.

Співпрограма, програмний модуль який організований для забезпечення взаємодії з іншими модулями по принципу кооперативної багатозадачності, IEnumerator AnimationDeley() реалізує знищення об’єкту та програвання перед нею анімації. Викликається у методі подій фізики OnCollisionEnter2D(Callider2D collision).

3.2.4 Реалізація керування головним героєм

Клас Player.cs, що реалізовує керування головним героєм та його взаємодію з різними поверхнями, унаслідується від базового класу Entity.cs. Цей клас має публічні поля:

– Rigidbody2D rb – компонент Rigidbody2D;

– Vector2 moveVector – вектор довжиною 2, що задає напрям переміщення;

– bool isGrounded – стан контакту об’єкта з шаром Ground;

– bool isDead – прапорець стану.

У методі подій Awake() значенню rb присвоюється компонент Rigidbody2D.

Визначено віртуальний метод Death(), який відповідає за реалізацію знищення об’єкта.

Клас Player.cs має приватне поле Vector3 luft та публічні поля:

– public GameObject respawn – точка перенесення головного героя при отриманні пошкодження;

– float speed – швидкість переміщення;

– float jumpForce – вертикально прикладена сила;

У методі подій FixedUpdate() перевіряються вхідні сигнали з клавіатури та в залежності від натиснутих клавіш викликаються методи Jump(), Fall(), Walk().

У методі подій Update() перевіряється стан isDead, та якщо його значення рівне «true» викликається метод Death().

У методі подій Start() вмикається програвання фонової музики.

Метод Walk() реалізовує пересування по горизонталі зі швидкістю speed.

Метод Walk() реалізовує можливість виконувати стрибки із силою jumpForce, та програє звук стрибка.

Метод Fall() реалізовує можливість виконувати швидке падіння.

Реалізований метод Death() програє звук отримання пошкоджень, пересуває головного героя на точку respawn.

У методі подій фізики OnCollisionEnter2D(Collision2D collision) проводиться перевірка стану об’єкта та програвання звуків в залежності від поверхні зіткнення.

Всі звуки що відтворюються на ігровому рівні, за позицією прив’язані до головного героя. В нього присутній окремий компонент Sounds.cs, який є сховищем для аудіо файлів. Цей клас реалізовано за допомогою патерна «Одинак». Він має публічні поля типу AudioSource для збереження аудіо файлів та static Sounds sound для отримання статичного посилання на клас. В методі Awake() полю sound присвоєно посилання на клас.

3.3 Створення головного меню

На сцені головного меню присутні три об’єкта типу Panel: Menu, Settings, LevelSelect.

Menu містить три кнопки:

– Обрати рівень – перемикає активний Panel на LevelSelect;

– Налаштування – перемикає активний Panel на Settings;

– Вихід – вимикає програму.

Також має компонент MainMenu.cs, який реалізовує завантаження обраного рівня та вихід з програми.

Settings містить один випадаючий список з вибором рівня якості зображення, слайдер для регулювання гучності звуків, слайдер для регулювання гучності музики, кнопку повернення до головного меню.

Клас SetQuality.cs реалізує налаштування якості зображення та збереження цього параметру. Клас SetAudio.cs реалізує налаштування рівня гучності вказаного звукового параметра та збереження його значення. Для відновлення встановлених налаштувань при повторному запускі програми розроблено класи InitAudio.cs та InitQualty.cs.

LevelSelect містить три кнопки вибору рівня та кнопку повернення до головного меню, реалізація яких наведена в MainMenu.cs.

Для відтворення звуку було створено аудіоміксер з групами звуків Music та Sound для можливості регулювання рівня гучності кожного з них окремо.

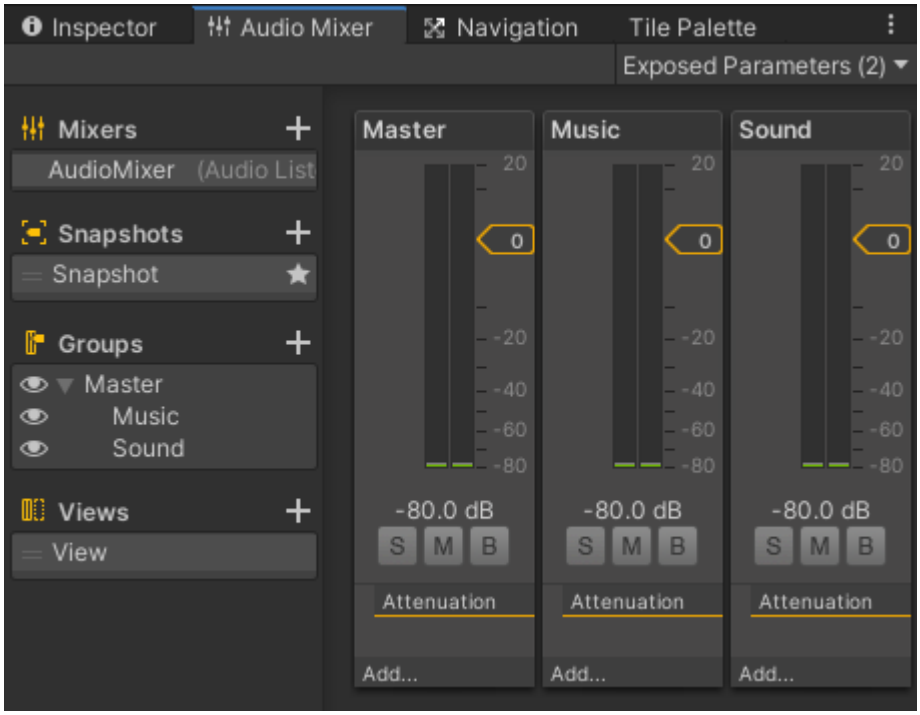


Рисунок 3.15 – Аудіоміксер

3.4 Висновки за розділом

У даному розділі було описано подробиці реалізації окремих модулів програми, основними з яких є камера, полотно, ігровий простір, головний герой, головне меню.

Наведено опис класів та детально охарактеризовано їх поля та методи. при розробці деяких класів використовувалися патерни «Одинак» та «Пул об’єктів».

4 ОПИС РОБОТИ ПРОГРАМНОЇ СИСТЕМИ

4.1 Меню програми

Запуск програми відбувається в повноекранному режимі. Спершу, завантажується сцена з активним головним меню в якому є кнопки для вибору рівня, налаштування, виходу з програми (рис. 4.1). На фоні грає мелодія, унікальна цієї сцени.

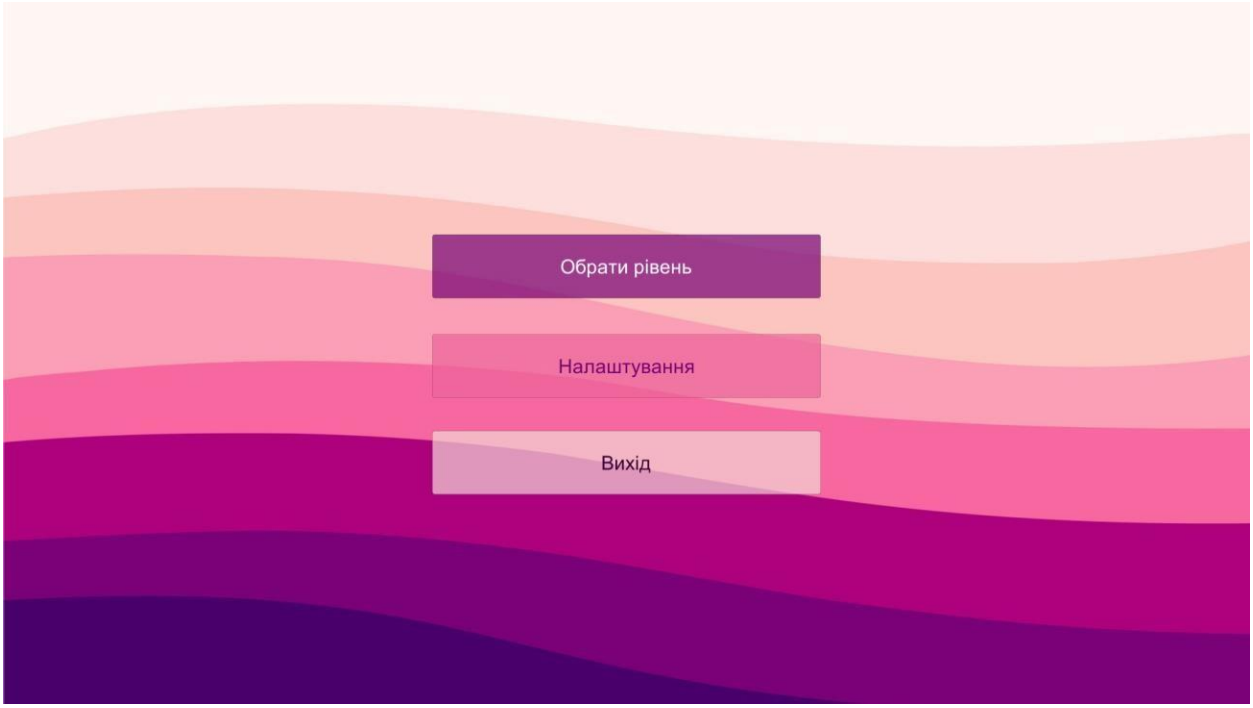


Рисунок 4.1 – Панель головного меню

При натисканні на кнопку «Вихід» програма завершує своє виконання. При натисканні кнопки «Налаштування» панель головного меню стає неактивною та активується панель налаштувань. На ній є випадаючий список для вибору якості зображення (рис. 4.2), повзунок для зміни рівня гучності музики та повзунок для зміни рівня гучності звуків. Налаштування гучності вступають в дію відразу після змінення їх значення (рис. 4.3).

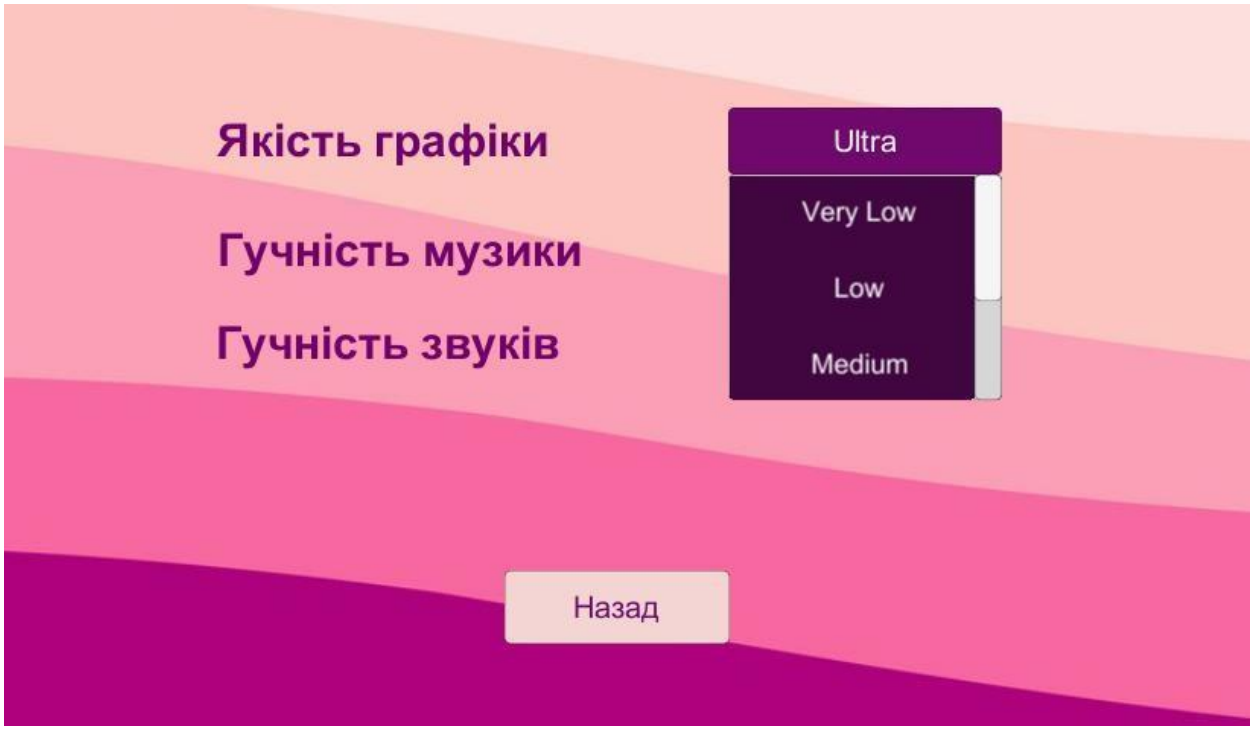


Рисунок 4.2 – Випадаючий список для налаштування якості зображення

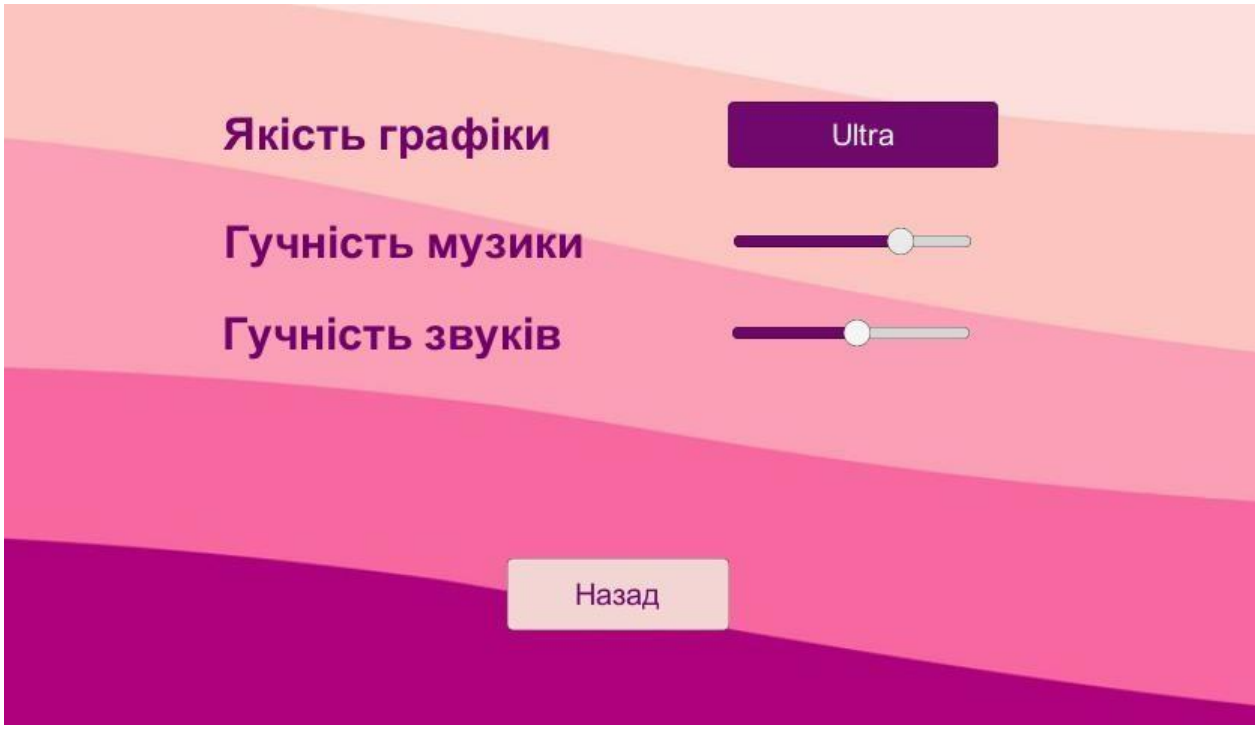


Рисунок 4.3 – Доступні опції для налаштування

При натисканні на кнопку «Обрати рівень», панель головного меню стає неактивною та активується панель з вибором ігрового рівня. На цій панелі знаходиться три великі кнопки із підписом та зображенням рівня, натиснувши на одну з низ, завантажиться сцена з відповідним рівнем. Панель вибору зображено на рисунку 4.4.

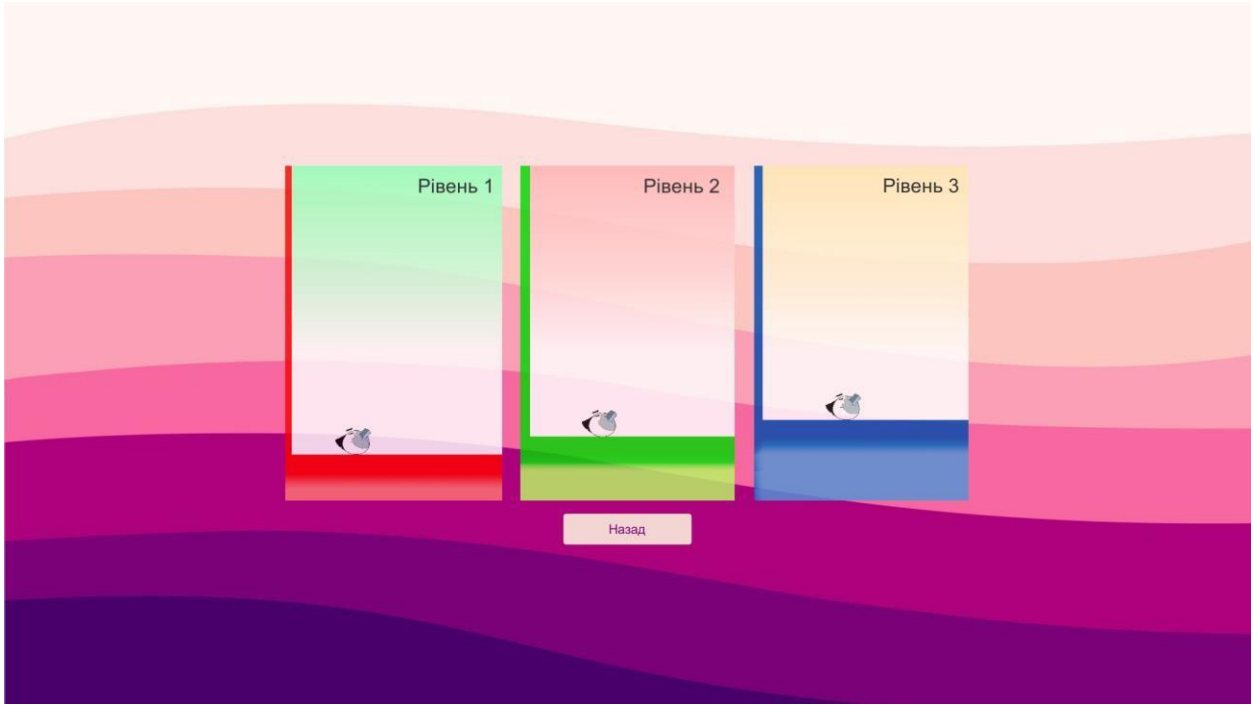


Рисунок 4.4 – Панель вибору рівня

На рівні є можливість призупинити гру, натиснувши на кнопку «Escape» після чого відкриється меню паузи з можливістю продовжити гру, повернутися до головного меню, налаштувати гучність звуків та музики. При відкритому меню паузи фонова музика на ігровому рівні зупиняється та вмикається унікальна музика меню паузи. При виході з меню паузи музика ігрового рівня продовжує грати, а музика меню звершується. Меню паузи зображено на рисунку 4.5.

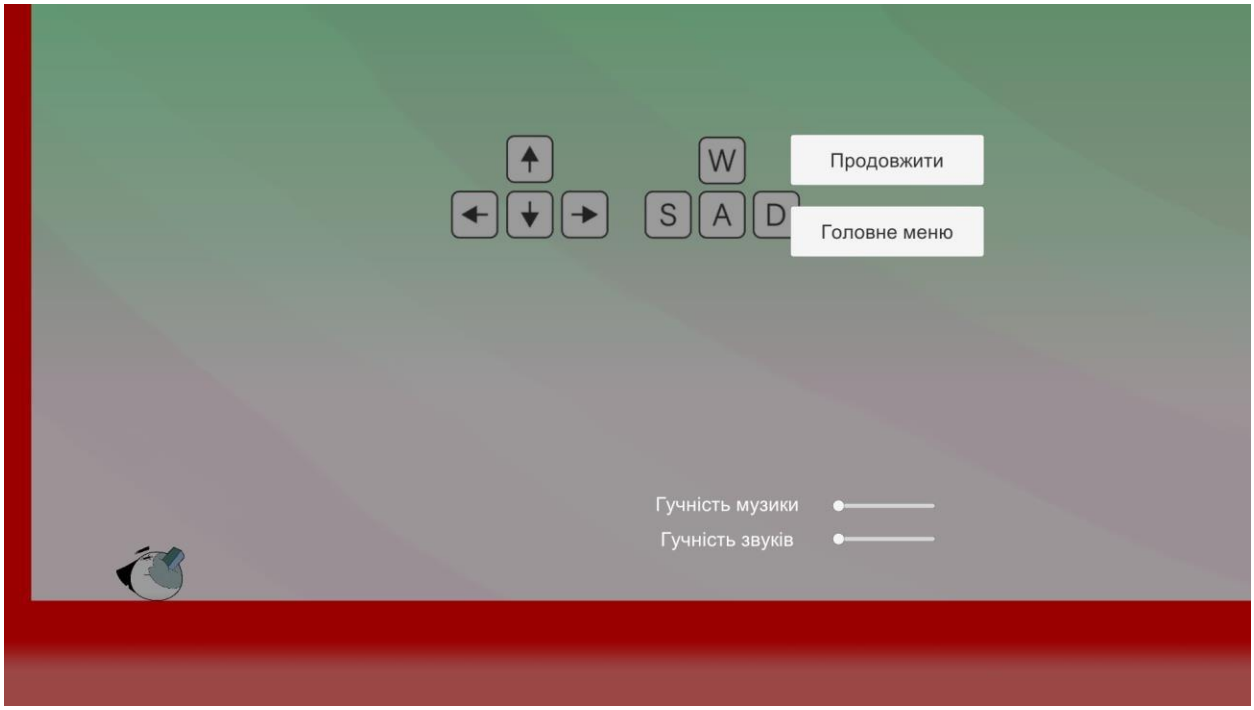


Рисунок 4.5 – Меню паузи

При досягненні фінішу, гра зупиняється та відображує фінішне меню, на якому є кнопка для повернення до головного меню. Фінішне меню зображено на рисунку 4.6.

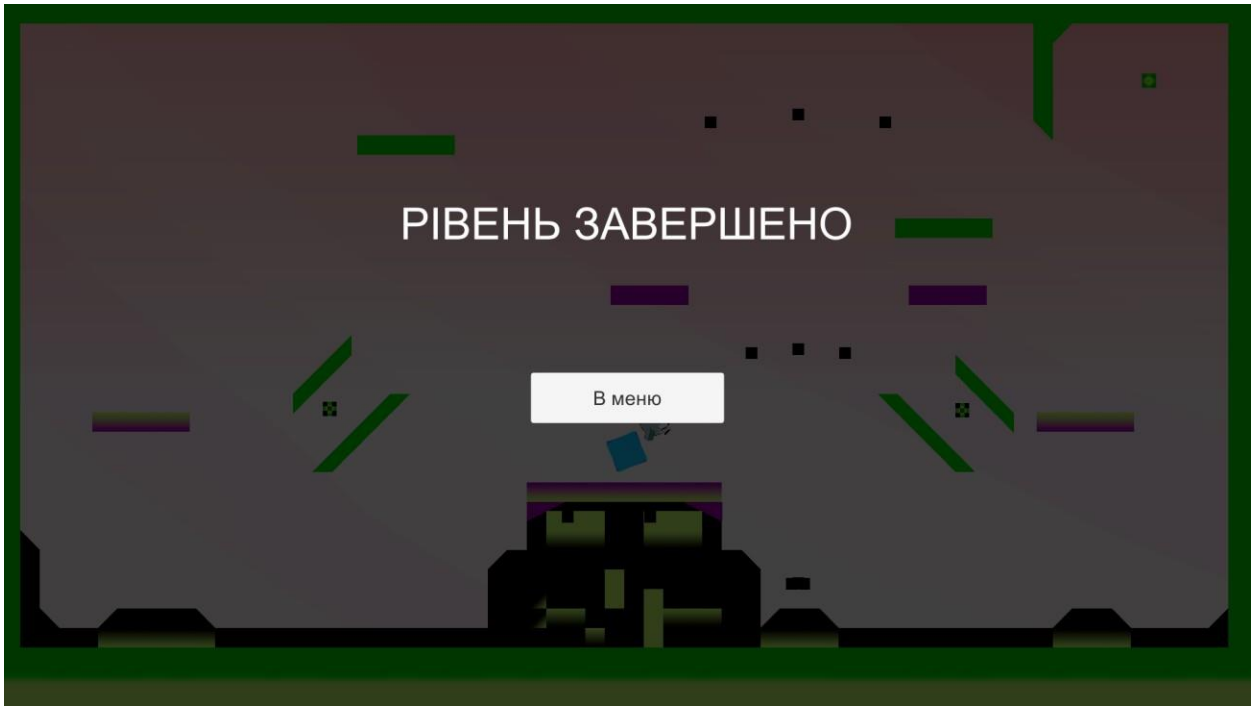


Рисунок 4.6 – Фінішне меню

4.2 Структура рівнів

Загальну структуру рівнів можна розділити на три частини:

– вступна частина – ознайомлює з об’єктами та ігровими механіками, які будуть використані на рівні;

– основна частини – впроваджую випробування на основі ігрових механік, що було показано у вступній частині;

– фінальна частина – об’єднує в собі використання всіх ігрових механік, які були показані на цьому та попередніх рівнях.

4.2.1 Перший рівень

Вступна частина цього рівня містить підказки щодо способу пересування, дає простір для ознайомлення з керуванням головного героя. Ознайомлює з шаром, що наносить пошкодження, з пружним шаром, з прискорюючим шаром. Вказує на можливість виконувати стрибки від стін. Вступна частина ігрового рівня зображена на рисунку 4.7.

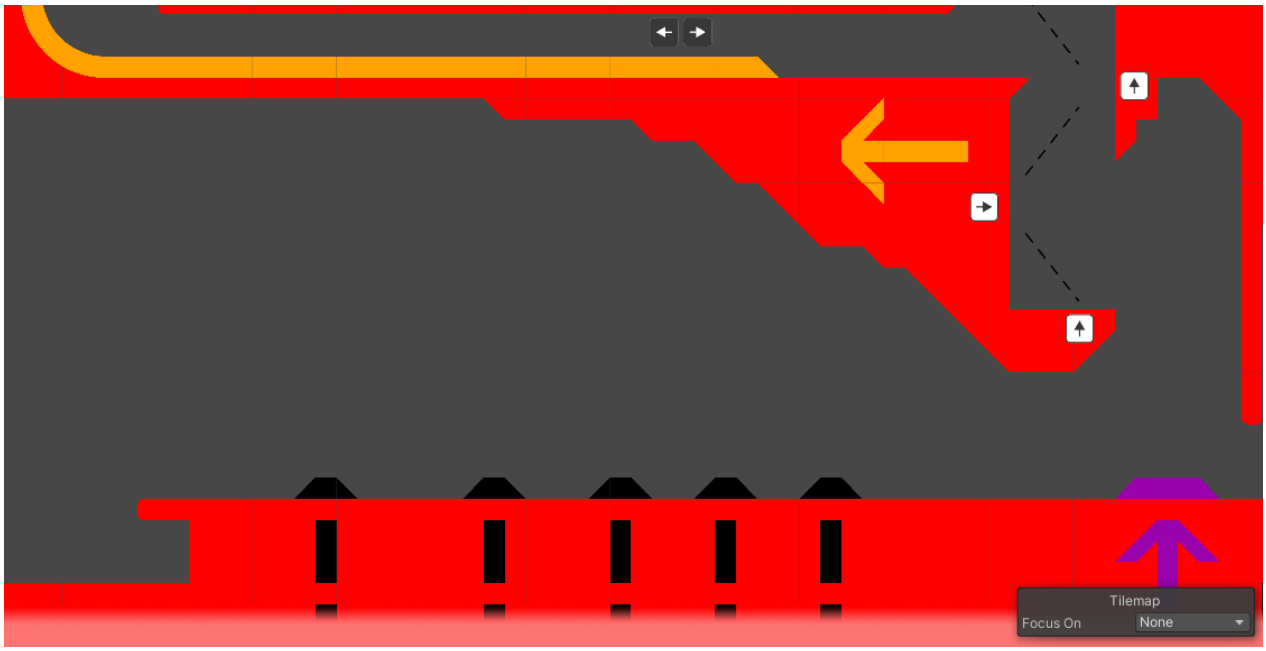


Рисунок 4.7 – Вступна частина першого рівня

Основна частина містить перешкоди на основі комбінування пружного та пошкоджуючого шару (рис. 4.8). Далі знаходяться пересувні платформи з різними патернами поведінки (рис. 4.9).

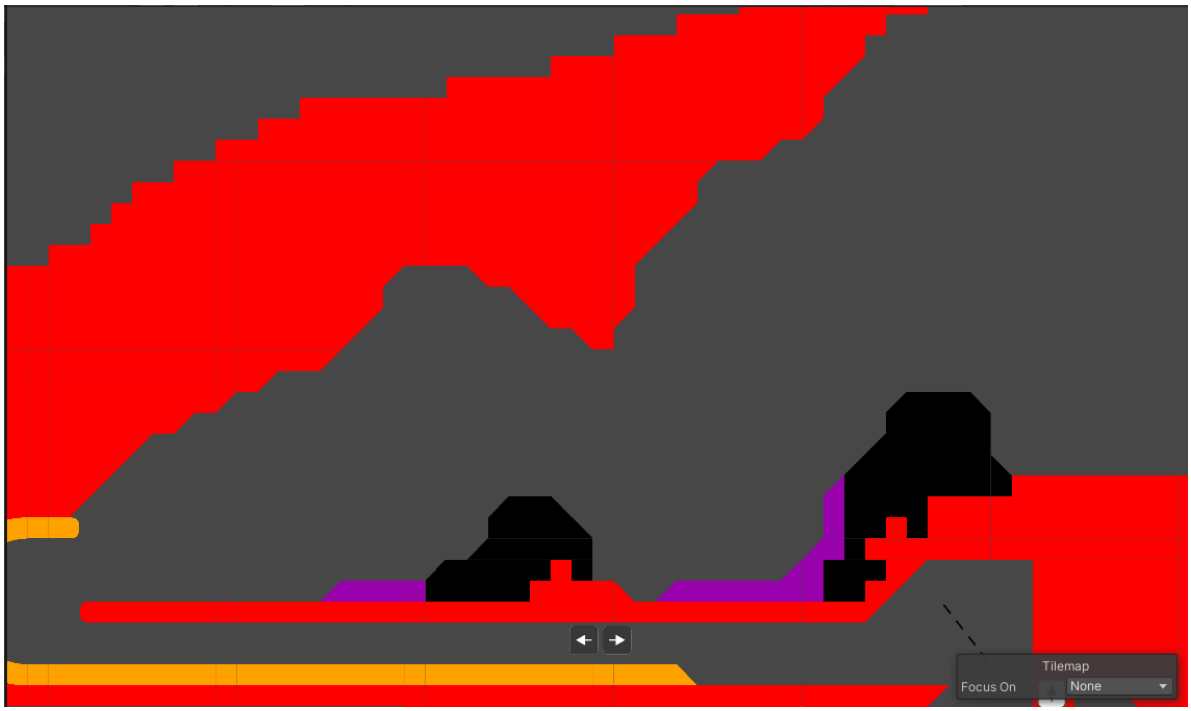


Рисунок 4.8 – Початок основної частини

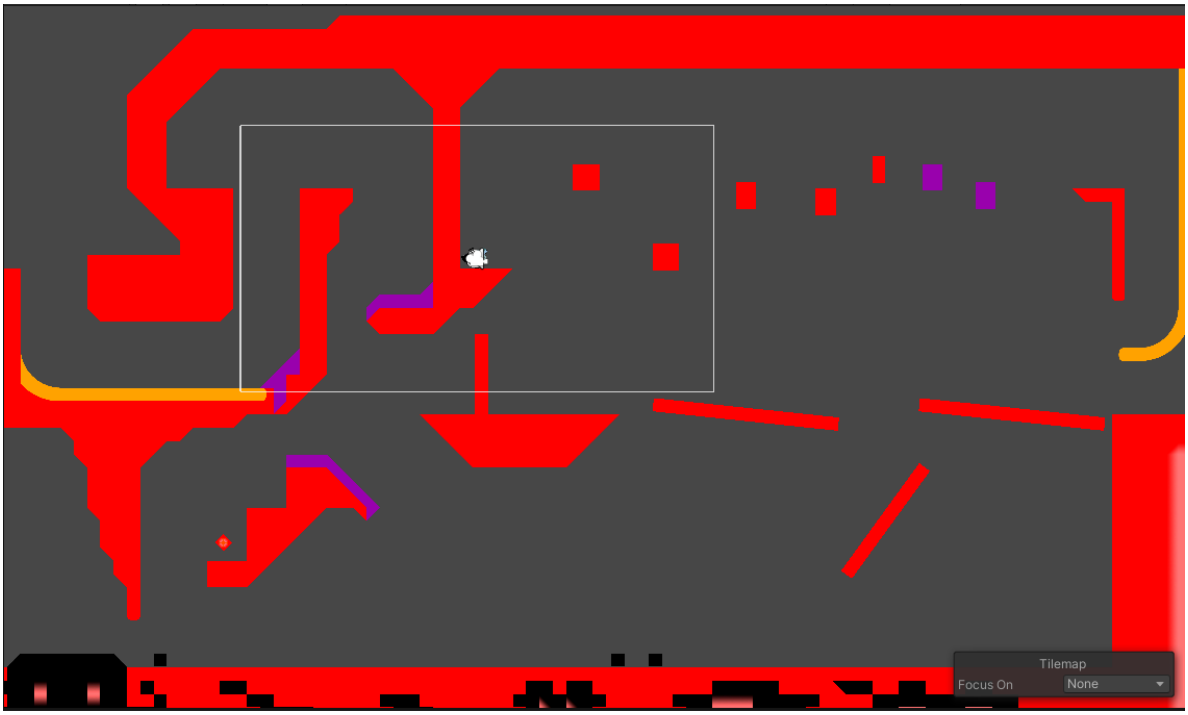


Рисунок 4.9 – Продовження основної частини першого рівня

Фінальна частина містить комбінацію всіх раніше використаних перешкод та нову перешкоду, що генерує кулі навколо себе. Генерація куль в ієрархії об’єктів та фінальну частину рівня показано на рисунку 4.10.

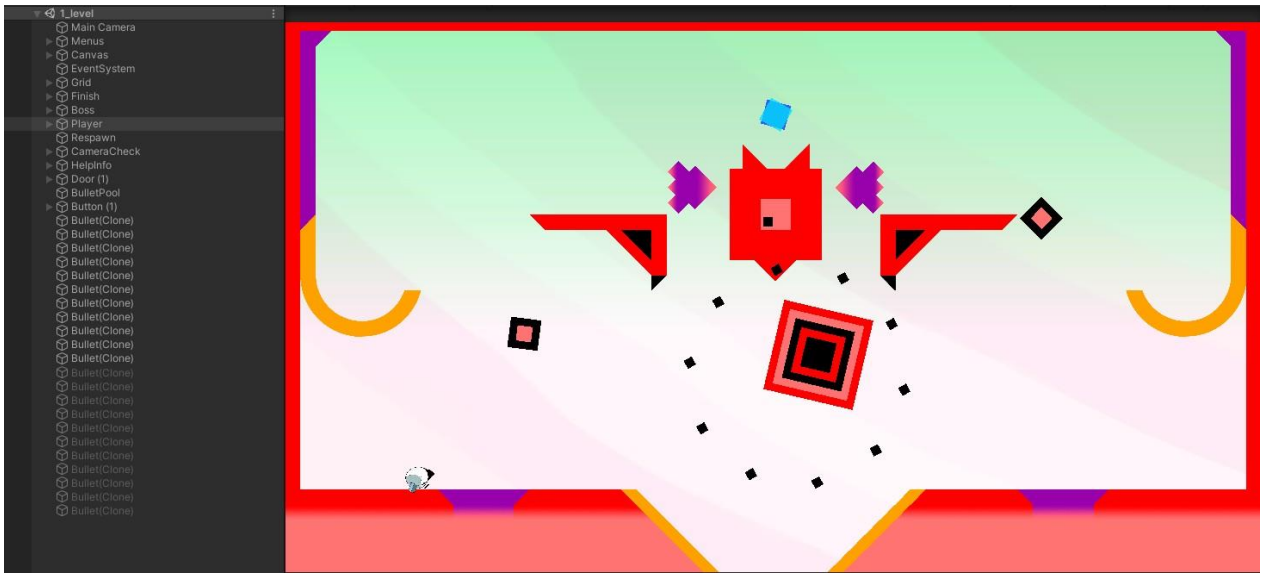


Рисунок 4.10 – Фінальна частина першого рівня

4.2.2 Другий рівень

Вступна частина цього рівня ознайомлює з інтерактивними елементами кнопкою та плиткою, варіантами взаємодії з ними. Вступна частина ігрового рівня зображена на рисунку 4.11.

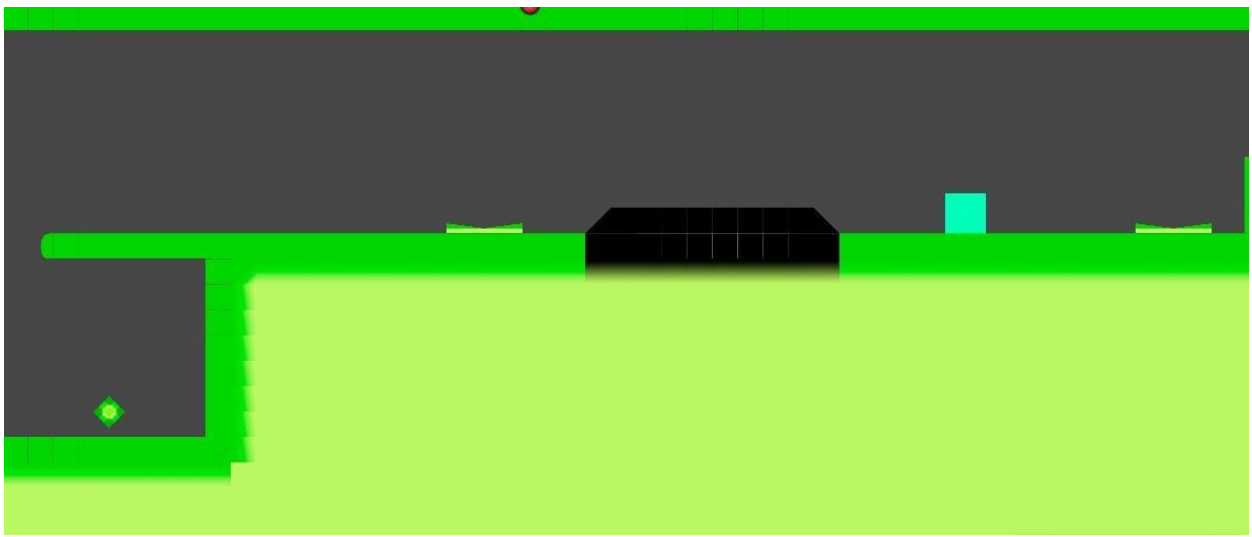


Рисунок 4.11 - Вступна частина другого рівня

Основна частина містить перешкоди на основі комбінування рухомих платформ, що активуються при натисканні плити, та точки генерації куль, яка вимикається при натисканні кнопки. Основна частина ігрового рівня зображена на рисунку 4.12.

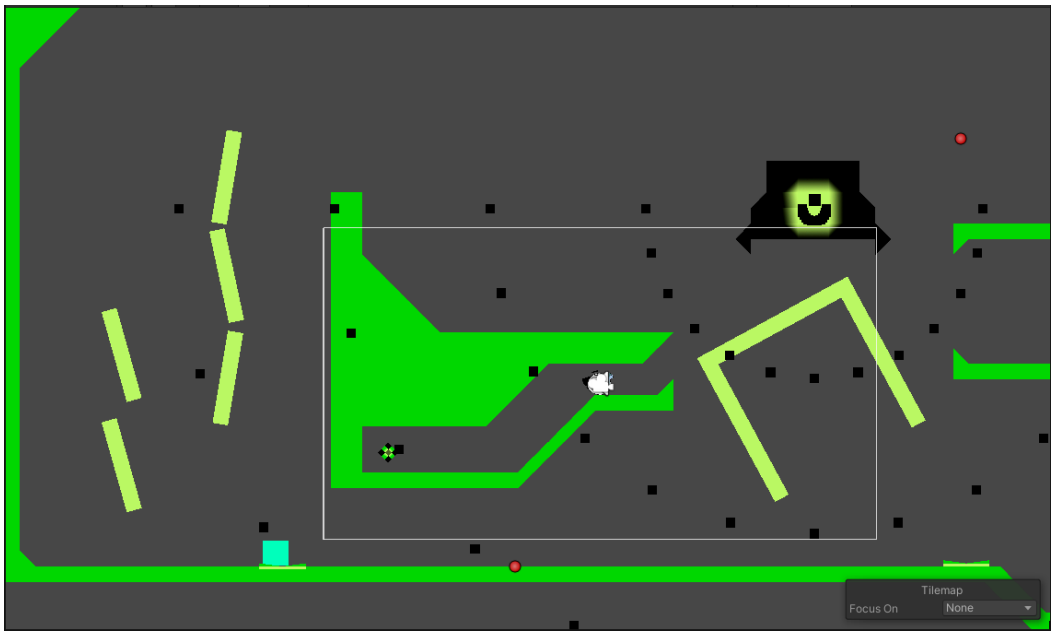


Рисунок 4.12 – Основна частина другого рівня

Фінальна частина містить комбінацію всіх раніше використаних перешкод. Фінальна частина ігрового рівня зображена на рисунку 4.13.

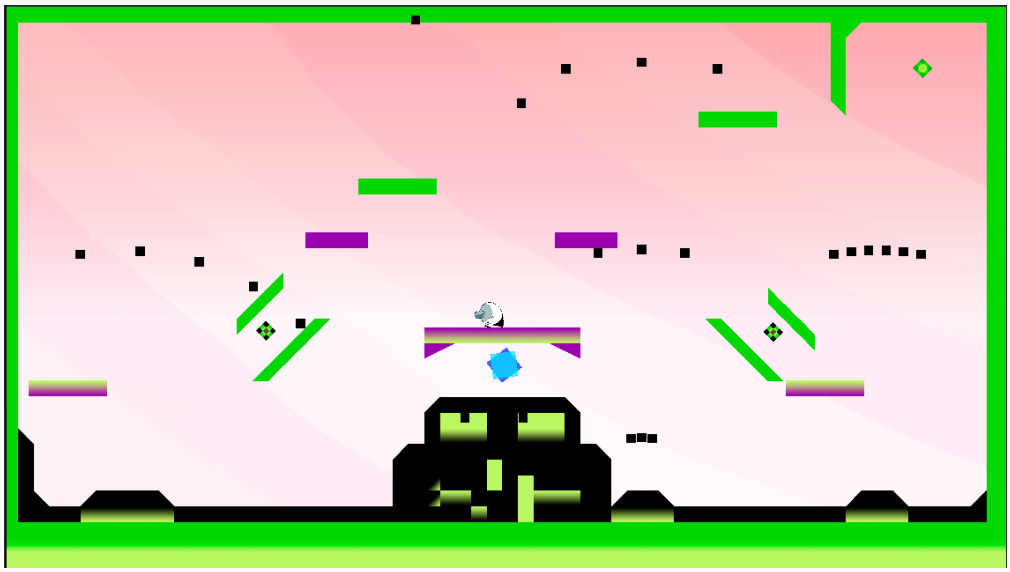


Рисунок 4.13 – Фінальна частина другого рівня

4.2.3 Третій рівень

Третій рівень декілька відрізняється від попередніх. На ньому немає перешкод в фінальній частині, але щоб до неї дістатися необхідно активувати три кнопки.

Вступна частина цього рівня ознайомлює з переслідуючими квадратами та точками генерації куль, які запускають їх в напрямку гравця. Вступна частина ігрового рівня зображена на рисунку 4.14.

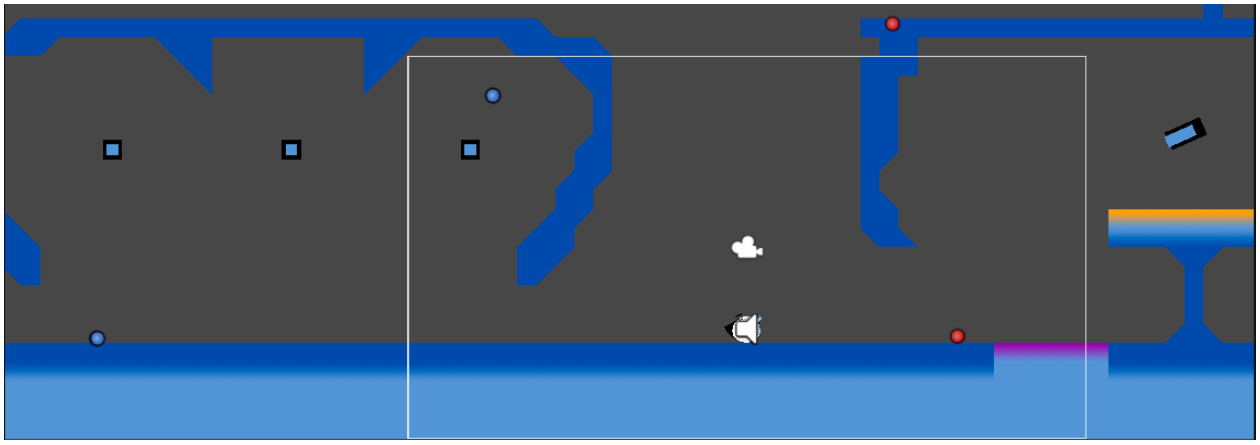


Рисунок 4.14 – Вступна частина третього рівня

Основна частина складається з трьох зон з кнопками, активація яких необхідна для проходження рівня. Перша зона містить невелику зміїсту ділянку на якій на головного героя націлено дві точки генерації куль. Перша зона вступної частини ігрового рівня зображена на рисунку 4.14.

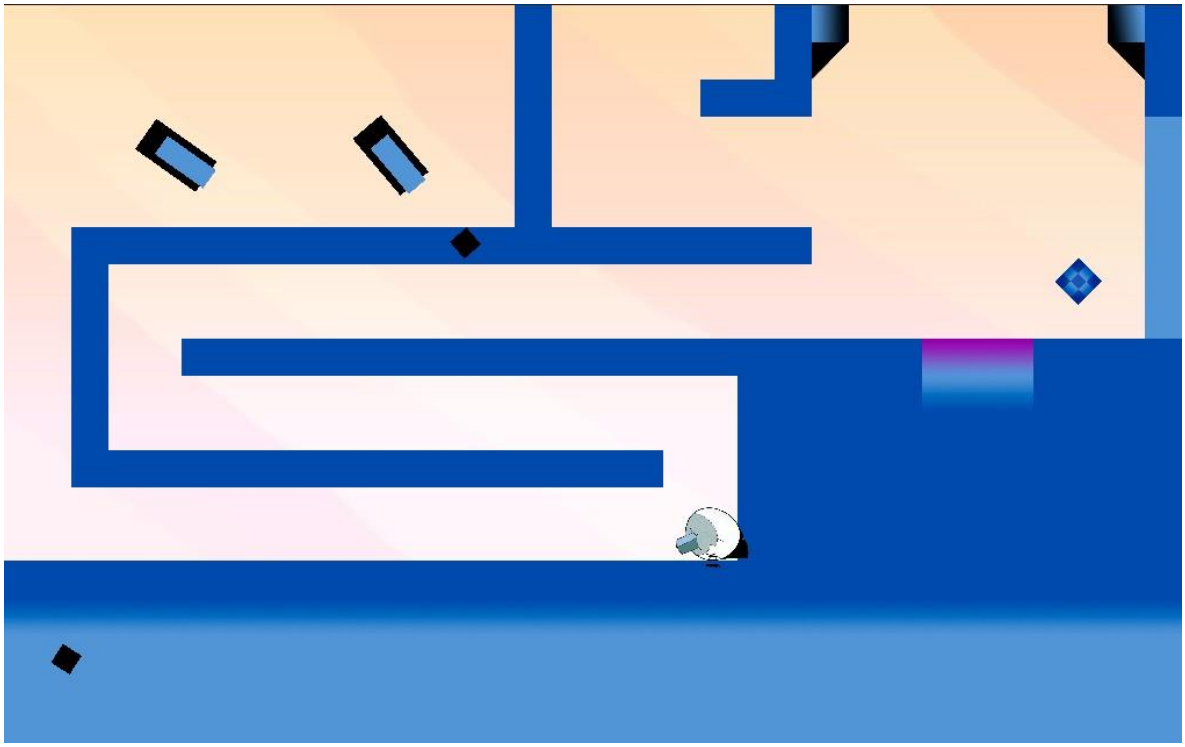


Рисунок 4.14 – Перша зона вступної частини третього рівня

Друга зона містить зміїсту ділянку на якій на головного героя націлено одну точку генерації куль. що генерує їх в невеликому секторі. Друга зона вступної частини ігрового рівня зображена на рисунку 4.15.

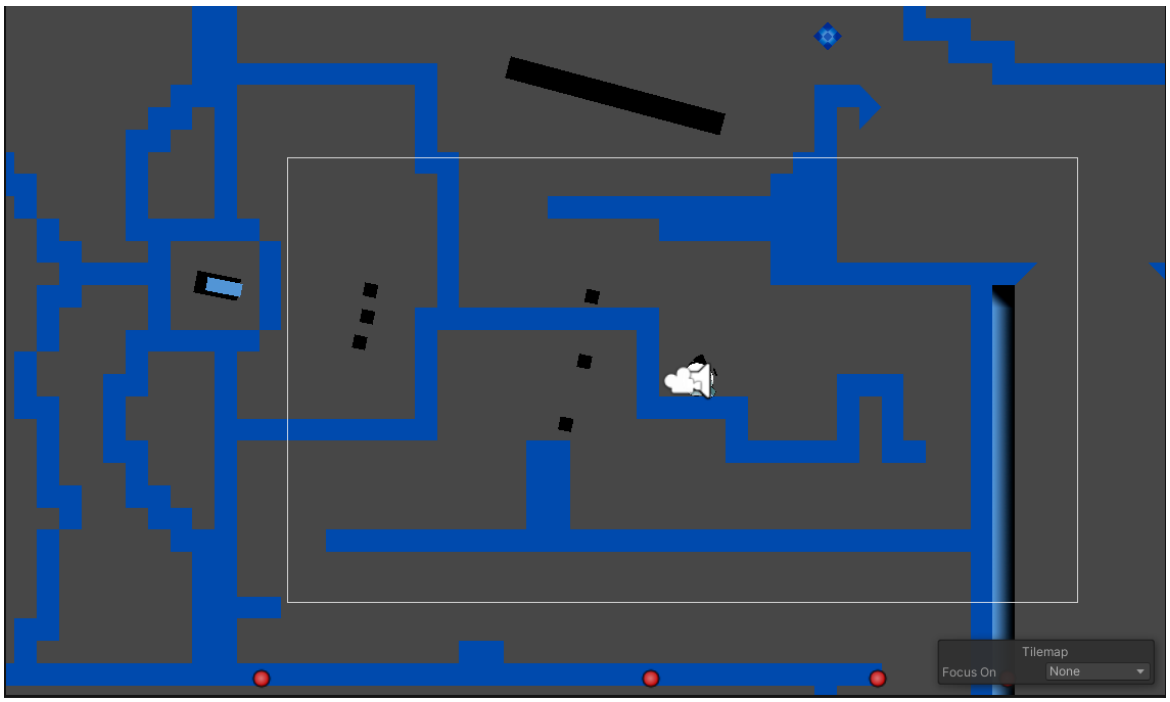


Рисунок 4.15 – Друга зона вступної частини третього рівня

Третя зона містить зміїсту ділянку на якій присутня велика кількість переслідуючих квадратів та плита, при активації якої починають підійматися платформ, що допомагають дістатися до кнопки. Якщо зійти з плити, то платформи починають спускатися. Третя зона вступної частини ігрового рівня зображена на рисунку 4.16.

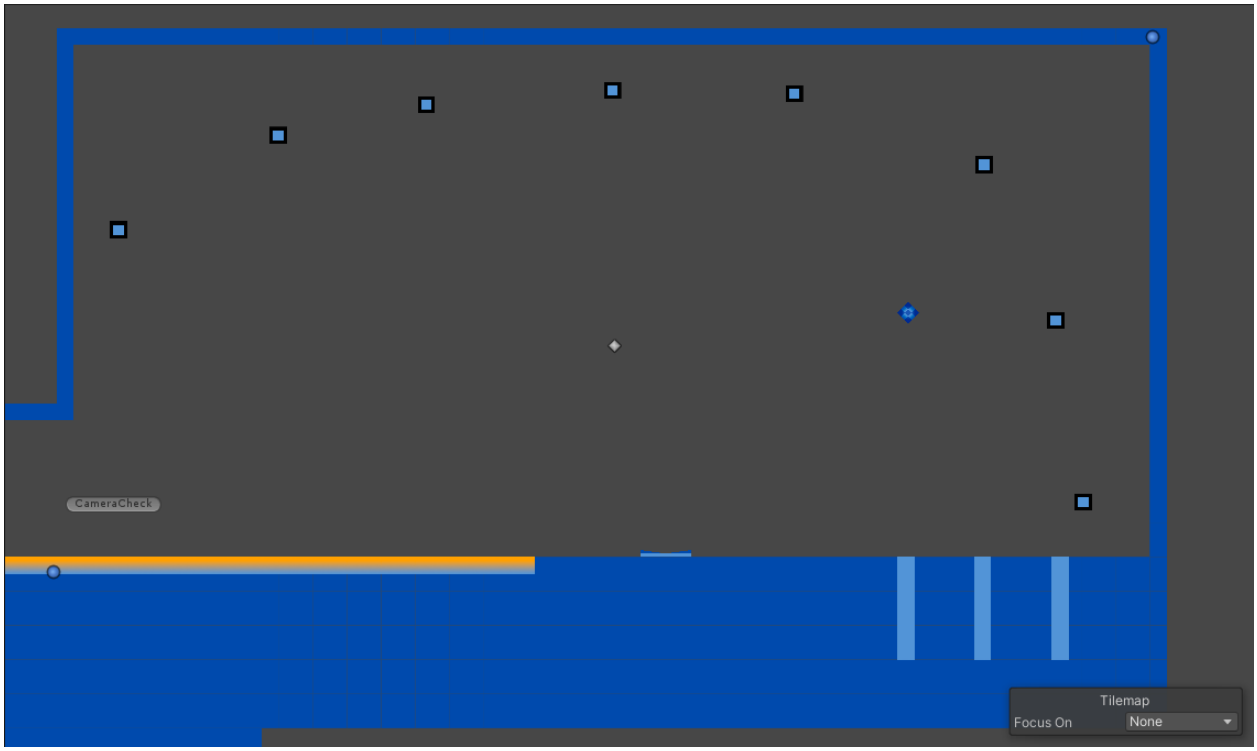


Рисунок 4.16 – Третя зона вступної частини третього рівня

Прохід до фінальної частини блокують три платформи, які переміщуються після активації відповідних кнопок. Фінальна частина ігрового рівня зображена на рисунку 4.17.

Рисунок 4.17 – Фінальна частина третього рівня

4.3 Висновки за розділом

У даному розділі було описано роботу програмної системи. Наведено зовнішній огляд елементів інтерфейсу.

Продемонстровано реалізовані людино-машинні інтерфейси та їх набори функцій. Описано та обґрунтовано побудовані структури ігрових рівнів, їх елементів. Розглянуто можливості користувача, як головного героя на ігровому рівні.

5 ТЕСТУВАННЯ ПРОГРАМНОЇ СИСТЕМИ

5.1 Тестування розробленої системи

Для виявлення помилок реальної поведінки програми відносно прогнозованої до релізу проєкту, регулярно проводилися тестування програмної системи на етапі розробки [26-27].

Тестування програмної системи проводилося функціональним методом. Функціональне тестування – це один із видів тестування, направленого на перевірку відповідності функціональних вимог програмного забезпечення до його реальних характеристик. Основною задачею функціонального тестування є підтвердження того, що розроблювана програмна система володіє всіма функціями, що було заплановано реалізувати в ній.

Процес тестування виконувався вручну. Ручне тестування проводиться тестувальником без використання програмних засобів, для перевірки програми шляхом імітування дій користувача.

Тестування програмної системи проводилося на п’яти рівнях:

– тестування компонентів – тестування мінімально можливого для тестування компоненту, такого як клас та його функції;

– інтеграційне тестування – тестування взаємодії між компонентами;

– системне тестування – тестування цілої системи на відповідність вимогам;

– Альфа-тестування – імітація реальної роботи програми самим розробником;

– Бета-тестування – тестування перед фінальної версії програмної системи з допомогою сторонньої групи людей;

5.1.1 Тестування компонентів

При тестуванні компонентів програмної системи було перевірено працепридатність кожного окремого скрипта. Їх тестування проводилося по мірі створення кожного з них.

Було знайдено та вирішено проблему прискорення. Якщо власна швидкість менша чи така ж як додаткове прискорення від спеціальної ділянки, гравець не міг пройти з право на ліво по ділянці з прискоренням, хоча міг пройти з ліво на право. Причина: при розрахунку напряму та швидкості руху використовується вектор, який при русі з право на ліво набуває від'ємних значень які нівелюються при додаванні прискорення. Для вирішення даної проблеми було перепрацьовано алгоритм прискорення таким чином, що при русі з право на ліво прискорення віднімалося від вектору прискорення.

5.1.2 Інтеграційне тестування

При інтеграційному тестуванні програмної системи було перевірено взаємодію головного героя з оточенням. Було виявлено проблему з підплигуванням.

Якщо гравець під час польоту доторкнеться до стінки, підплигне, спуститься на землю, не відриваючись від стінки, то втратить можливість плигати поки не відірветься від землі. Причина: Оскільки підлога та стіни знаходяться на одному шарі «Ground», гравець зміг підплигнути не відриваючись від цього шару, в наслідок чого змінна «isGrounded» не прийняла значення «true» бо повторної колізії шару «Ground» та гравця не відбулося й «isGrounded» залишила значення «false» після підплигування. Рішення: перед підплигуванням перемістити гравця на незначну висоту, щоб відбувся відрив від шару «Ground» та сталася повторна колізія з ним.

Також було виявлено проблему провалювання головного героя в текстури при зіткненні з ними на високій швидкості. Причина: властивість компоненти Rigidbody2D, що прикріплено до ігрового об'єкта головного героя, Collision Detection має значення Discrete. В цьому випадку Unity

розраховує фізику та колізію об’єктів в конкретні проміжки часу, через що момент обробки колізії може відбутися вже після фактичного зіткнення об’єктів. Рішення: змінити значення параметра Collision Detection на Continuous. В цьому випадку Unity розраховує першу точку зіткнення та пересуває об’єкт до неї.

5.1.3 Системне тестування

При системному тестуванні програмної системи було перевірено загальну роботу всіх компонентів програмної системи. Протестовано переходи між сценами, функціонал всіх кнопок. При перегляді побудованого рівня було помічено графічну проблему.

В редакторі та в грі помічено лінії на шарах елемента полотно. Проблему зображено на рисунку 5.1.

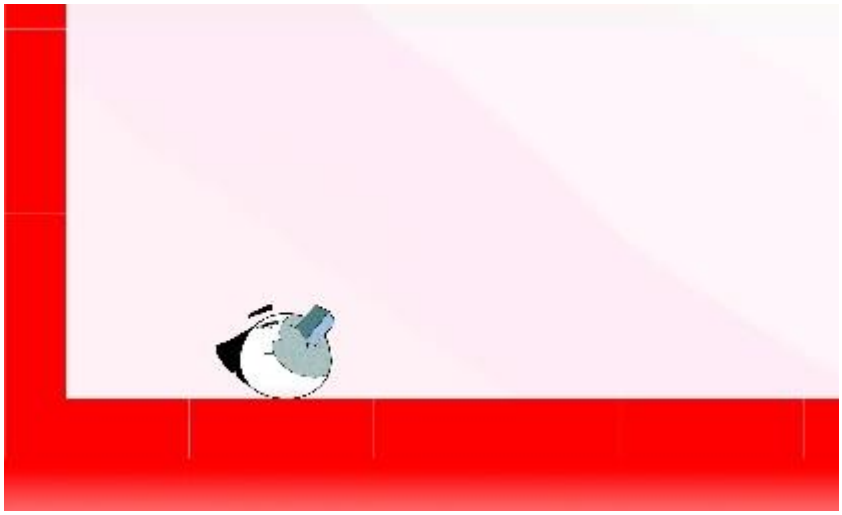


Рисунок 5.1 – Проблеми з відображенням полотна

Рішення: створити асет Sprite Atlas – асет, який консолідує декілька різних текстур в одну комбіновану текстуру. Замість відображення кожної текстури окремо, Unity звертається до Sprite Atlas та відображує всі текстури за один виклик. На рисунку 5.2 зображено створений асет, на рисунку 5.3 зображено результат з його використанням.

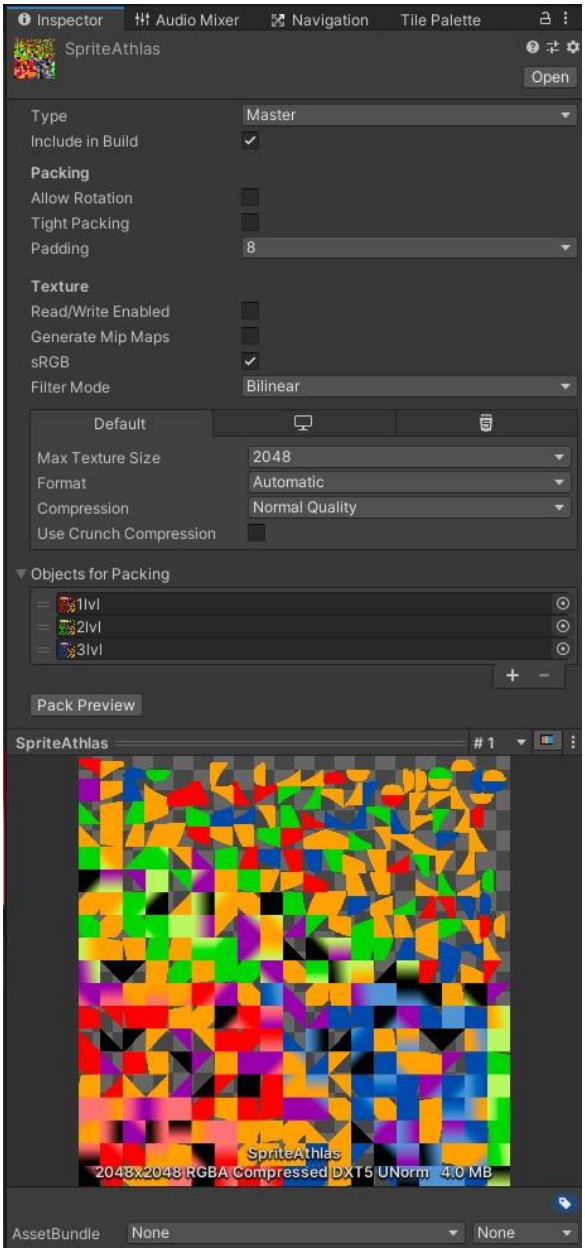


Рисунок 5.2 – Створений асет Sprite Atlas

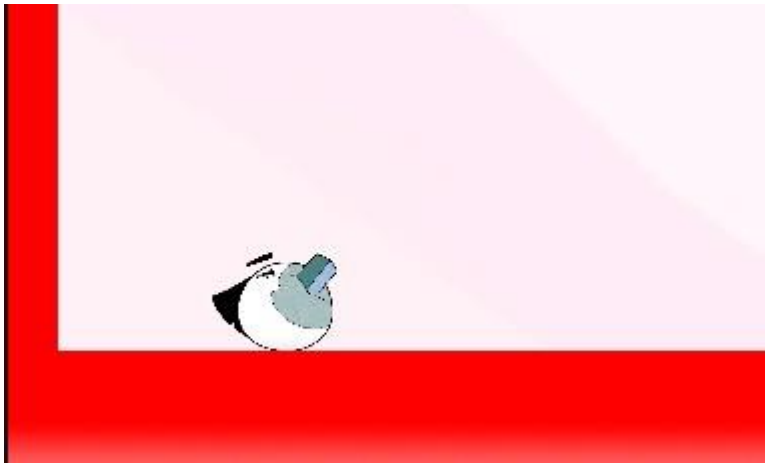


Рисунок 5.3 –Відображення полотна після створення Sprite Atlas

5.1.4 Альфа-тестування

При альфа-тестуванні програмної системи було проведено тестування ігрового процесу. Налаштовано швидкість рухомих платформ, підібрано щільність та швидкість польоту куль. На другому ігровому рівні знайдено та вирішено проблему побудови рівня. На ньому була можливість втратити необхідні для проходження рухомі ігрові об’єкти квадрати, підперши їх під вертикальну стінку без можливості відтягти назад. Рішення: додати елементи ландшафту, що допоможуть уникнути даної ситуації. Ландшафт другого рівня до зміни наведено на рисунку 5.4, після зміни на рисунку 5.5.

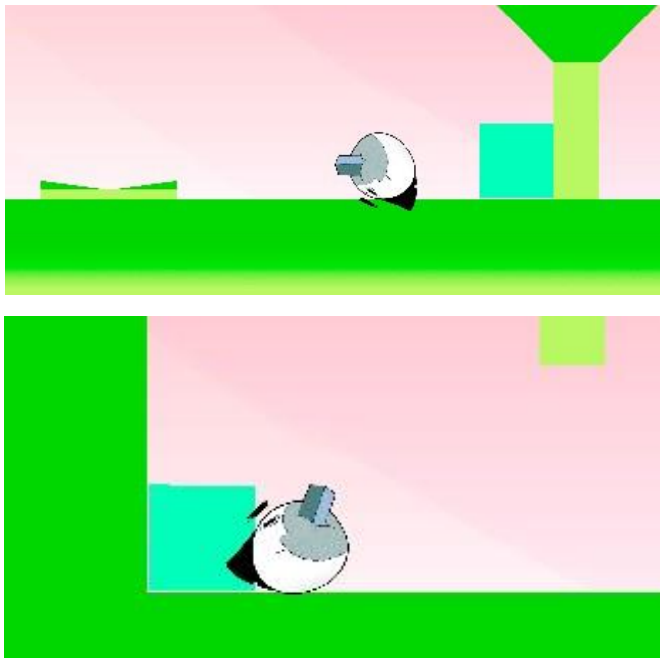


Рисунок 5.4 – Ландшафт другого ігрового рівня до зміни

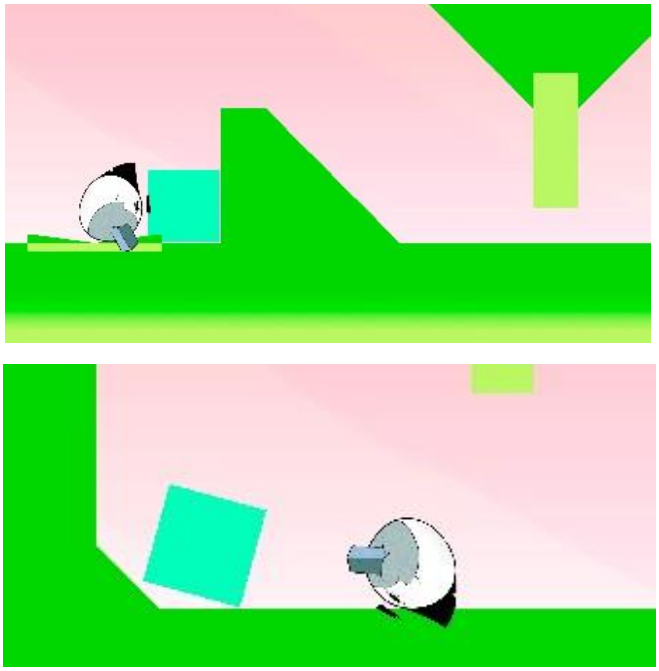


Рисунок 5.5 – Ландшафт другого ігрового рівня після зміни

5.1.5 Бета-тестування

При бета-тестуванні програмної системи було запропоновано покористуватися програмним засобом групі людей, які не брали участі в розробці. В результаті тестування ними програми, отримано наступні зауваження:

– на першому ігровому рівні, у фінальній частині, головний герой після падіння в яму не переноситься на точку переродження, а продовжує безкінечно падати;

– на другому ігровому рівні існує можливість встановлення точки відродження перед фінальною кімнатою, підійшовши впритул до стінки вказаної на рисунку 5.6;

– необхідно додати можливість налаштування звуку в меню паузи.

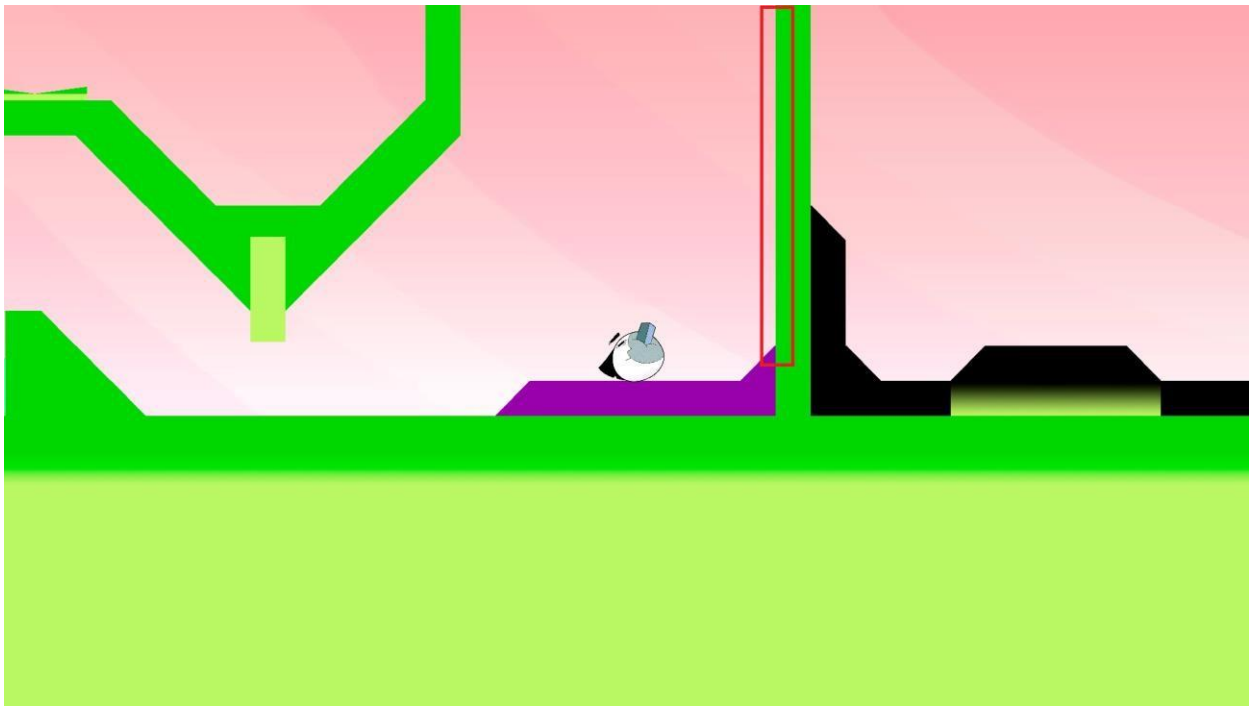


Рисунок 5.6 – Проблемна ділянка

Переглянувши перше зауваження, під ямою було додано елемент шару, що наносить пошкодження головному герою після чого він переноситься на точку відродження. Виправлення зображено на рисунку 5.7.

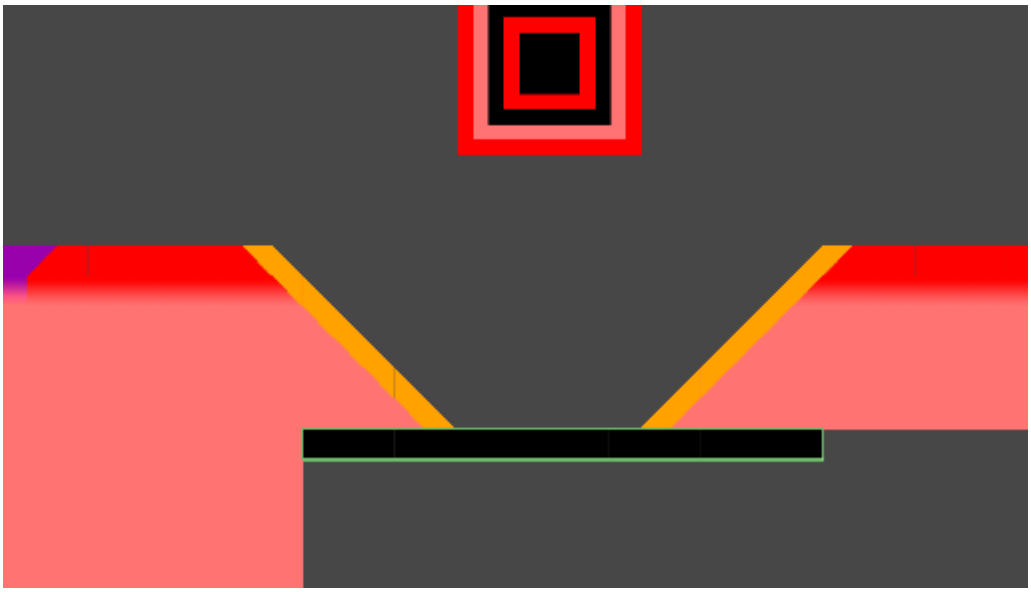


Рисунок 5.7 – Доданий елемент

Переглянувши друге зауваження, було пересунуто точку «CameraCheck» правіше від стінки, вказаної на рисунку 5.6.

Переглянувши третє зауваження, було додано можливість налаштування звуку в меню паузи, що зображено на рисунку 5.8.

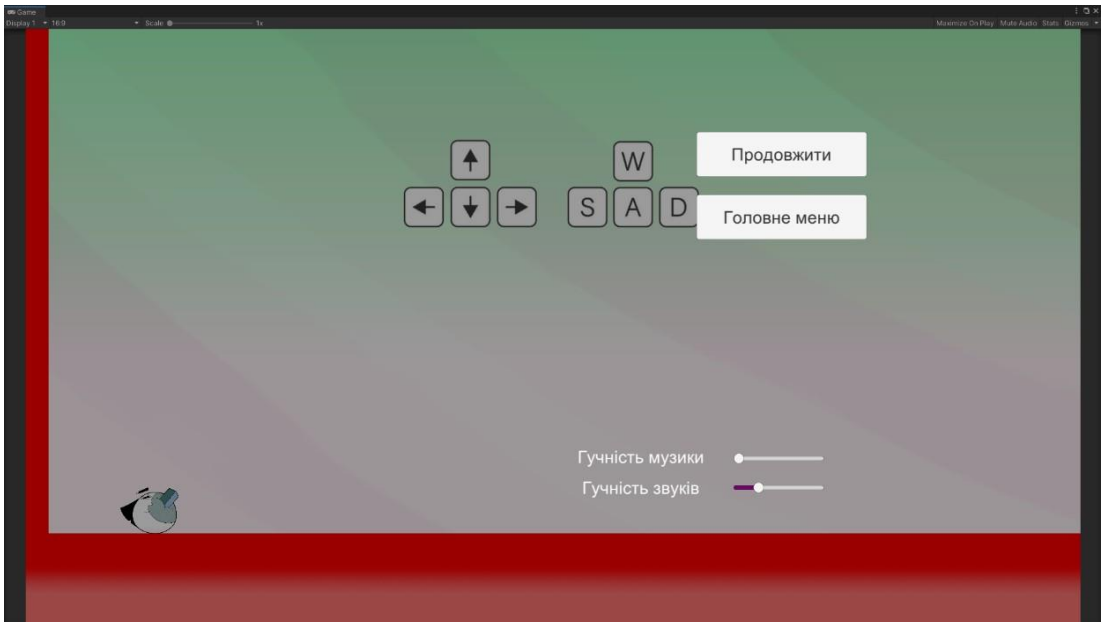


Рисунок 5.8 – Оновлене меню паузи

5.2 Висновки за розділом

У даному розділі було проведено тестування програмної системи. Описано метод та етапи тестування.

Під час тестування компонентів було виявлено та вирішено проблему роботи прискорюючого шару.

Під час інтеграційного тестування було виявлено та вирішено проблеми з підплигуванням та обробкою колізій головного героя.

Під час системного тестування було виявлено та вирішено проблему відображення текстури рівня.

Під час альфа-тестування було виявлено та відредаговано проблемні місця на ігрових рівнях.

Під час бета-тестування було зібрано відгуки інших користувачів та впроваджено зміни на їх основі.

ВИСНОВКИ

В результаті виконання кваліфікаційної роботи було створено програмний додаток однокористувацької гри-платформера для комп'ютерних пристроїв на базі Windows за допомогою засобів C# та Unity. Для досягнення поставленої мети використовувалися методи досліджень, які базуються на використанні основних положень теорії алгоритмів, проєктування програмного забезпечення, об’єктно-орієнтованого програмування.

На початку роботи було проведено системний аналіз предметної області, розглянуто представників сучасної ігрової індустрії, визначено основні моменти при проєктування та розробці комп’ютерного ігрового додатку

Розглянуто сучасний інструментарій для створення комп’ютерних ігор та підходи до проєктування комп’ютерних систем, таких як об’єктно-орієнтоване програмування та мову графічного моделювання систем UML. Серед патернів проєктування розглянуто «Одинак» та «Пул об’єктів». Сформовано основані завдання роботи та виконано їх в подальшому.

Для проєктування програмної системи було використано інструмент Draw.io, який підтримує мову UML, та з його допомогою спроєктовано:

- діаграму варіантів використання для користувача;
- діаграму діяльності користувача в головному меню;
- діаграму діяльності користувача на ігровому рівні;
- діаграму послідовності вибору ігрового рівня;
- діаграму станів застосування користувацьких налаштувань;

Розроблено прототипи людино-машинного інтерфейсу з наступними елементами:

- головне меню;
- меню налаштувань;
- меню вибору рівня;

– меню паузи.

Засобами розробки обрано ігровий рушій Unity, текстовий редактор Microsoft Visual Studio, графічний редактор Adobe Photoshop, цифрова звукова робоча станція Reaper, що зазначено в сформованому концепт-документі проєкта.

При розробці програмного засобу було підготовано графічні та звукові матеріали для розробки за допомогою зазначених інструментів. По описані концепції, побудовано ігрові рівні для яких розроблено сценарії пересування камери, відображення елементів на полотні, взаємодії ігрових об’єктів з різними поверхнями та іншими об’єктами на сцені, пересування платформ, генерації та патерну поведінки куль. Для керування головним героєм створено сценарій який реалізовує пересування в горизонтальній площині, можливість підплигування та швидкого приземлення, отримання пошкоджень від поверхні з певним тегом. Додано звукові ефекти та можливість їх налаштування.

Наведено опис роботи готової програмної системи з приведенням графічних ілюстрацій головного меню, меню налаштувань, меню вибору рівня та меню паузи. Описано та показано структуро кожного ігрового рівня.

Приведено ручне тестування програмної системи функціональним методом. Тестування програмної системи проводилося на п’яти рівнях:

– тестування компонентів;

– інтеграційне тестування;

– системне тестування;

– альфа-тестування;

– бета-тестування.

Всі виявлені помилки було виправлено.

В результаті розробки програмної системи було виконано всі поставлені завдання роботи.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. ДСТУ 3008-95. Документація. Звіти у сфері науки і техніки. Структура і правила оформлення. [Електронний ресурс] – Режим доступу: http://www.ukrbook.net/DSTU_pabl.htm.

2. C# documentation [Електронний ресурс] – режим доступу: https://docs.microsoft.com/uk-ua/dotnet/csharp.

3. Unity – Scripting API [Електронний ресурс] – Режим доступу: https://docs.unity3d.com/2019.4/Documentation/ScriptReference/.

4. Сторінка продукту «Geometry Dash»в засобі цифрової дистрибуції [Електронний ресурс] – Режим доступу: https://store.steampowered.com/app/322170/Geometry_Dash/?l=russian.

5. Сторінка продукту «Journey»в засобі цифрової дистрибуції [Електронний ресурс] – Режим доступу: https://store.steampowered.com/app/638230/Journey/.

6. Сторінка продукту «Thomas Was Alone»в засобі цифрової дистрибуції [Електронний ресурс] – Режим доступу: https://store.steampowered.com/app/220780/Thomas_Was_Alone/.

7. Сторінка продукту «Undertale» в засобі цифрової дистрибуції [Електронний ресурс] – Режим доступу: https://store.steampowered.com/app/391540/Undertale/.

8. Сторінка продукту «Hyper Light Drifter» в засобі цифрової дистрибуції [Електронний ресурс] – Режим доступу: https://store.steampowered.com/app/257850/Hyper_Light_Drifter/.

9. Сторінка продукту «Celeste» в засобі цифрової дистрибуції [Електронний ресурс] – Режим доступу: https://store.steampowered.com/app/504230/Celeste/.

10. Методологія розробки ігор з позиції гейм-дизайну [Електронний ресурс] – Режим доступу: https://www.slideshare.net/KonstantinSakhnov/sakhnov-sgevent.

11. Інструмент для створення діаграм [Електронний ресурс] – Режим доступу: https://startpack.ru/application/draw-io.

12. Совершенный код. Мастеркласс / Пер. с англ. — Макконнелл С. : Издательство «Русская редакция», 2010. — 896 стр. : ил.

13. Объектно Ориентированное Программирование. Хорошая книга для Хороших Людей. – М.: СОЛОН-Пресс, 2014. – 298 с.: ил.

14. Паттерны объектно-ориентированного проєктирования. – СПб.: Питер, 2021. – 448 с.: ил.

15. Паттерны проєктирования для C# и платформы .NET Core. – СПб.: Питер, 2021. – 352 с.: ил.

16. Паттерны программирования игр / Роберт Нистрем ; [перевод с английского М. А. Райтмана]. — Москва : Эксмо, 2021. — 432 с.

17. Введение в UML от создателей языка. 2-е изд.: Пер. с англ. Мухин Н. – М.: ДМК Пресс, 2015. – 496 с.: ил.

18. Самоучитель UML 2. – СПб.: БХВ-Петербург, 2007. – 576 с.: ил.

19. UML. Проєктирование систем реального времени, параллельных и распределенных приложений: Пер. с англ. – М.: ДМК Пресс, 2016. 700 с.: ил.

20. Unity в действии. Мультиплатформенная разработка на C#. 2-е межд. изд. – СПб.: Питер, 2019. – 352 с.: ил.

21. Unity и C#. Геймдев от идеи до реализации. 2-е изд. – СПб.: Питер, 2021. – 928 с.: ил.

22. Язык программирования C# 7 и платформы .NET и .NET Core, 8-е изд. : Пер. с англ. – СПб. : ООО «Диалектика», 2019 – 1328 с. : ил. – Парал. тит. англ.

23. C# 9 и .NET 5. Разработка и оптимизация. СПБ.: питер, 2022. – 832 с.: ил.

24. Head First. Изучаем C#. 4-е изд. / Пер. с англ. Е. Матвеева. – СПб.: Питер, 2022. – 768 с.: ил.

25. Изучаем C# через разработку игр на Unity. 5-е издание. – СПб.: Питер, 2022. – 400 с.: ил.

26. Серйозність та пріоритет бага - чи є різниця? - [Електронний ресурс] – Режим доступу: https://testmatick.com/ru/sereznost-i-prioritet-baga-est-li-raznitsa/.

27. Дневник охотника за ошибками. Путешествие через джунгли проблем безопасности программного обеспечения. Пер. с англ. Киселев А. Н. – М.: ДМК Пресс, 2013. – 240 с.: ил.

28. Спрайт головного героя – [Електронний ресурс] – Режим доступу: https://enajoelg.fandom.com/wiki/Moony?file=Moony+block+she.png.

ДОДАТОК А
ЗАУВАЖЕННЯ НОРМОКОНТРОЛЕРА

Таблиця А.1 – Зауваження нормоконтролера

Сторінка	Позначення	Зміст зауваження
Зміст		Інтервал не 1.5, заголовки розділів прописними
ПЗ		Нещільне заповнення сторінок ПЗ
С. ЗЗ		Невірний перелік
Додаток Г		Невірне оформлення

ДОДАТОК Б
ГРАФІЧНА ЧАСТИНА

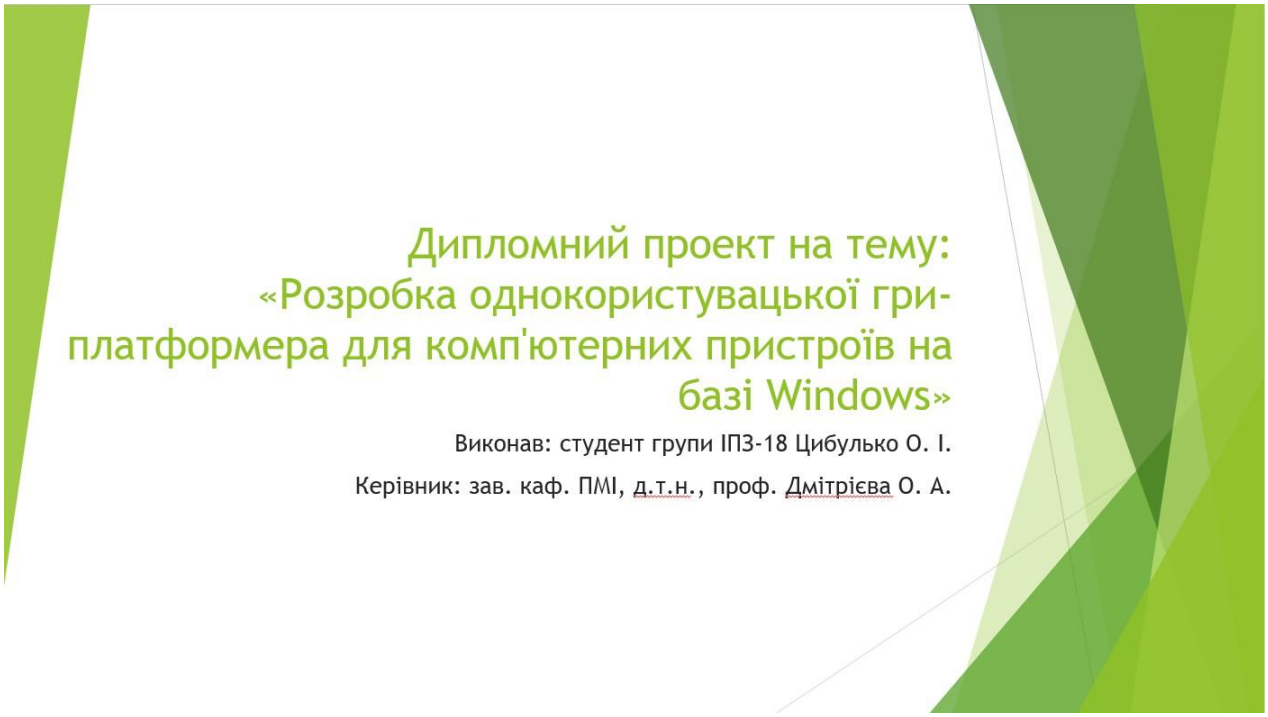


Рисунок Б.1 – Титул

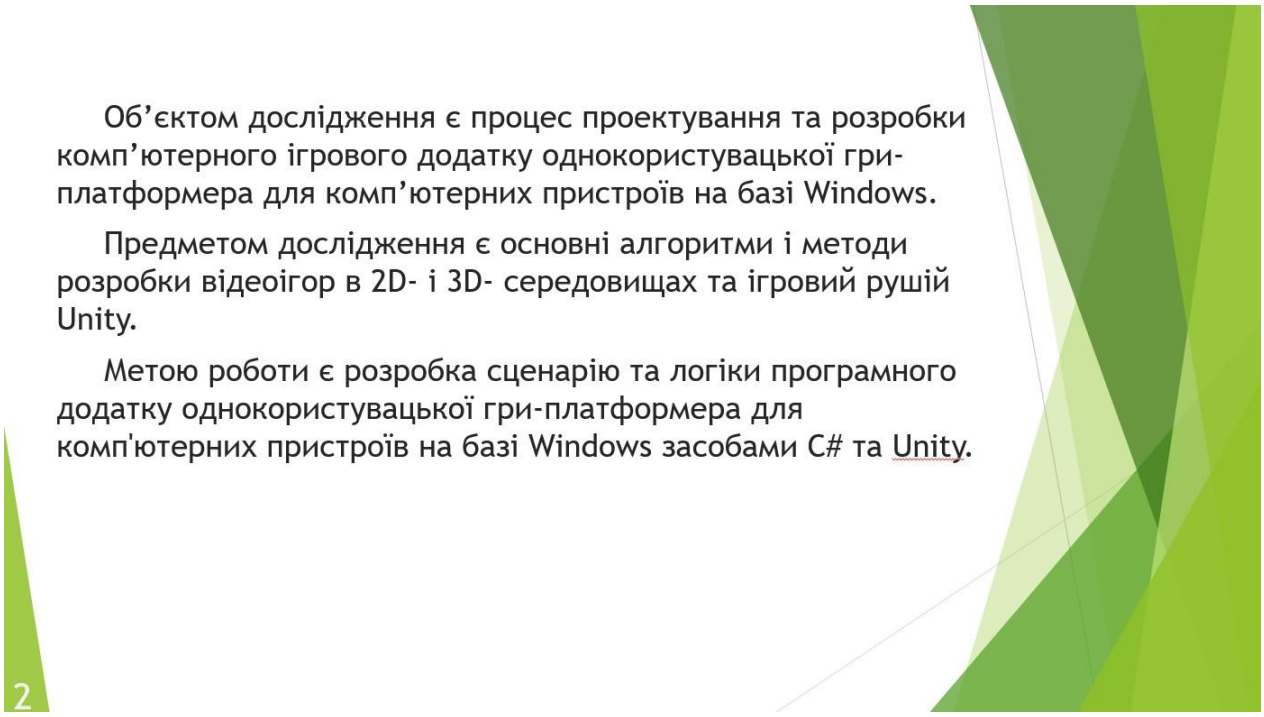


Рисунок Б.2 – Вступ

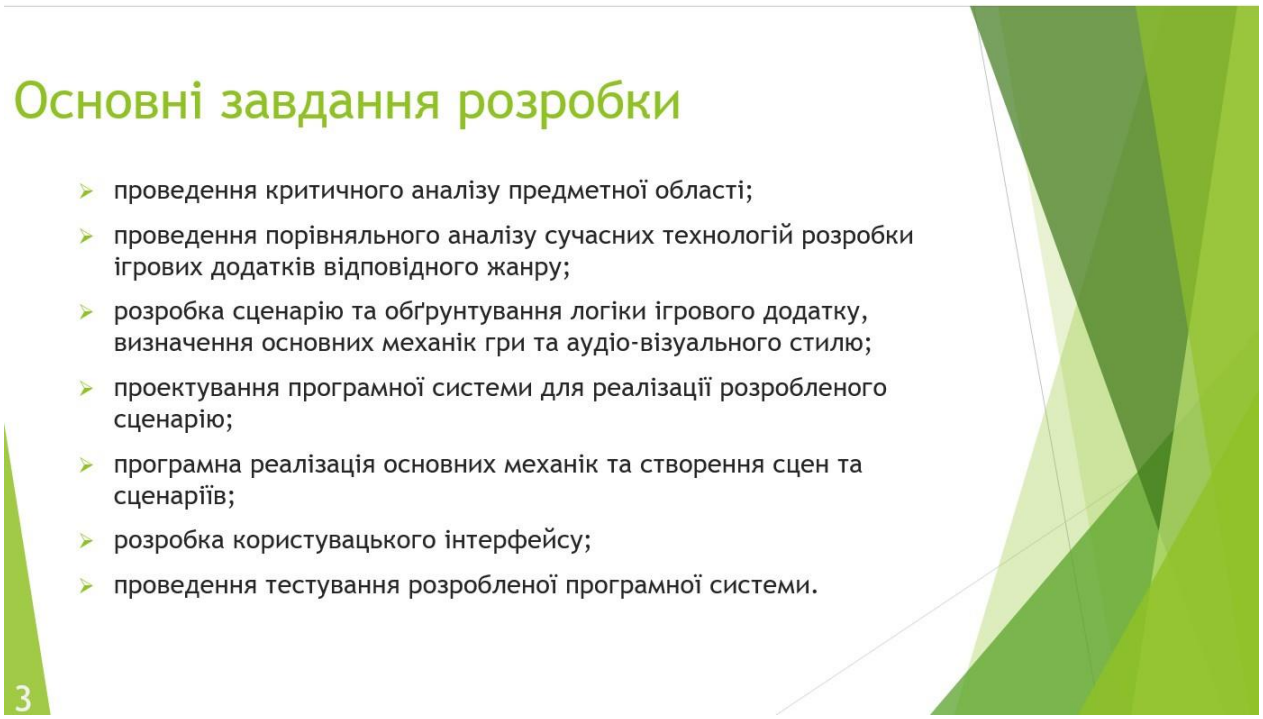


Рисунок Б.3 – Основні завдання

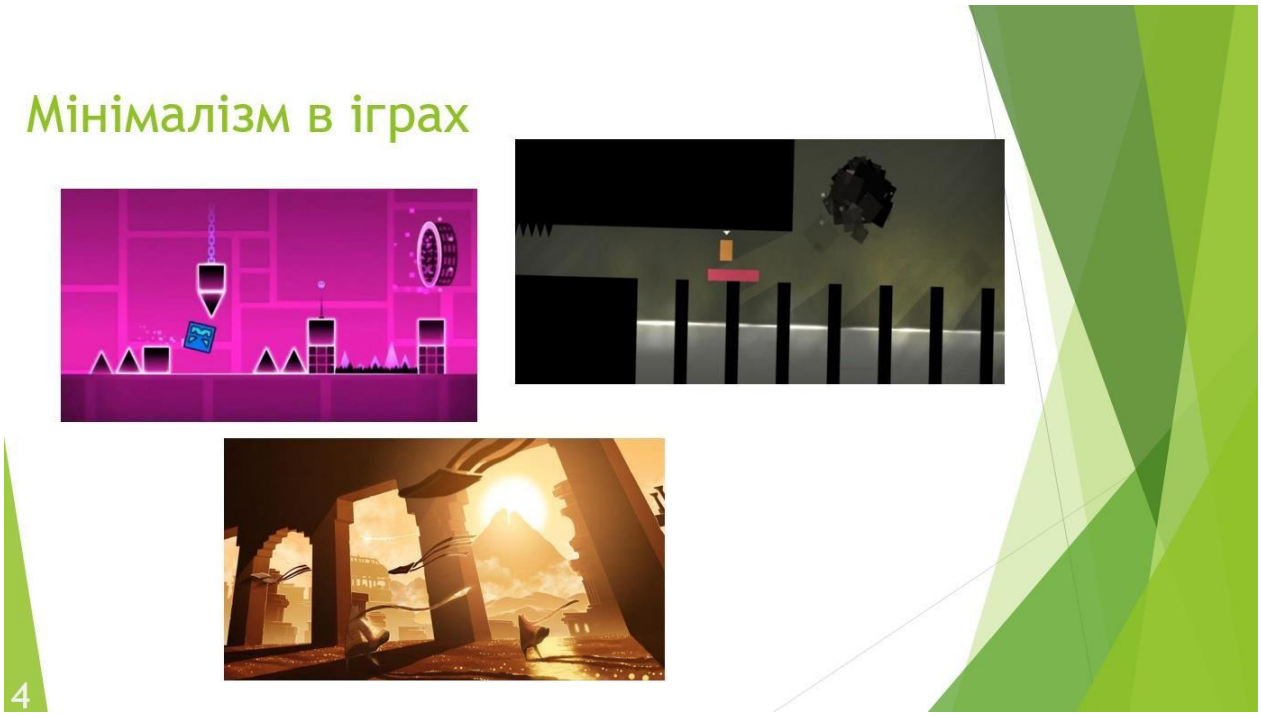


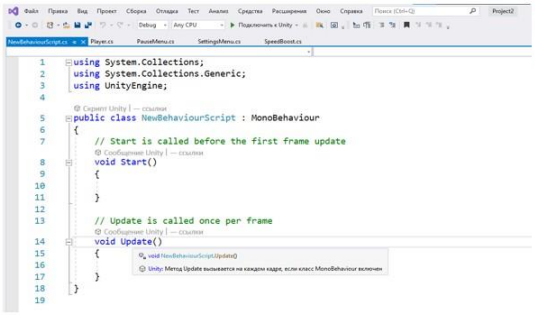
Рисунок Б.4 – Приклад мінімалізму

Рисунок Б.5 – Приклад піксель-арту

Рисунок Б.6 – Використані концепції та патерни

Засоби розробки

Unity - кросплатформене середовище розробки комп'ютерних ігор. Unity дозволяє створювати програми, що працюють під більш ніж 20 різними операційними системами, що включають персональні комп'ютери, ігрові консолі, мобільні пристрої, інтернет-програми та інші.



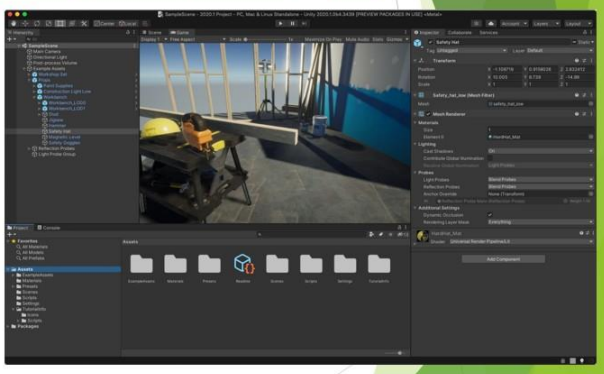
```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class NewBehaviourScript : MonoBehaviour
{
    // Start is called before the first frame update
    void Start()
    {

    }

    // Update is called once per frame
    void Update()
    {

    }
}
```

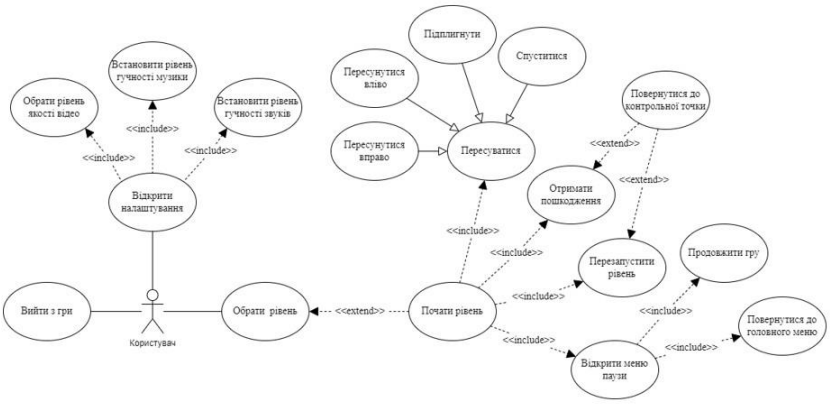


Найкращим вибором середи розробки буде Visual Studio, оскільки вота зручно інтегрується з Unity, підсвічуючи синтаксис та впроваджуючи підказки на мові програмування C#.

7

Рисунок Б.7 – Засоби розробки

Проектування програмного засобу



Діаграма варіантів використання для користувача

8

Рисунок Б.8 – Діаграма варіантів використання

Рисунок Б.9 – Діаграми діяльності користувача

Рисунок Б.10 – Діаграми послідовності та станів

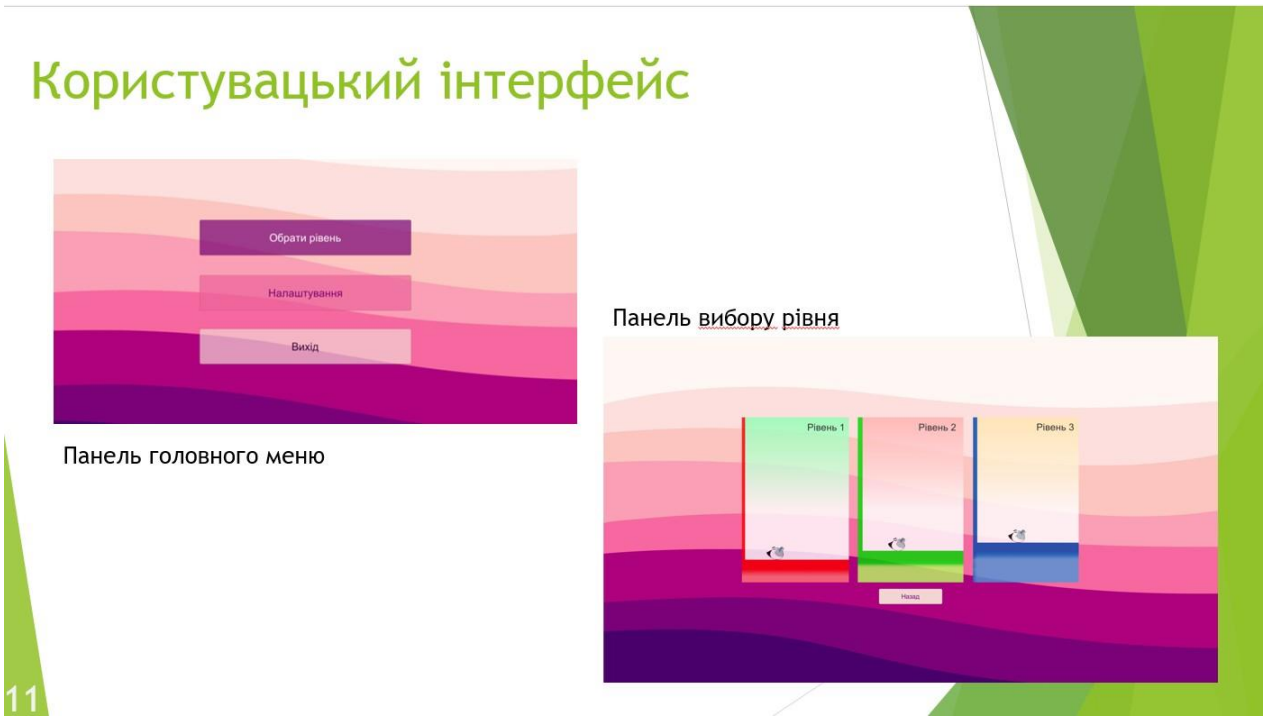


Рисунок Б.11 – головне меню та меню вибору рівня

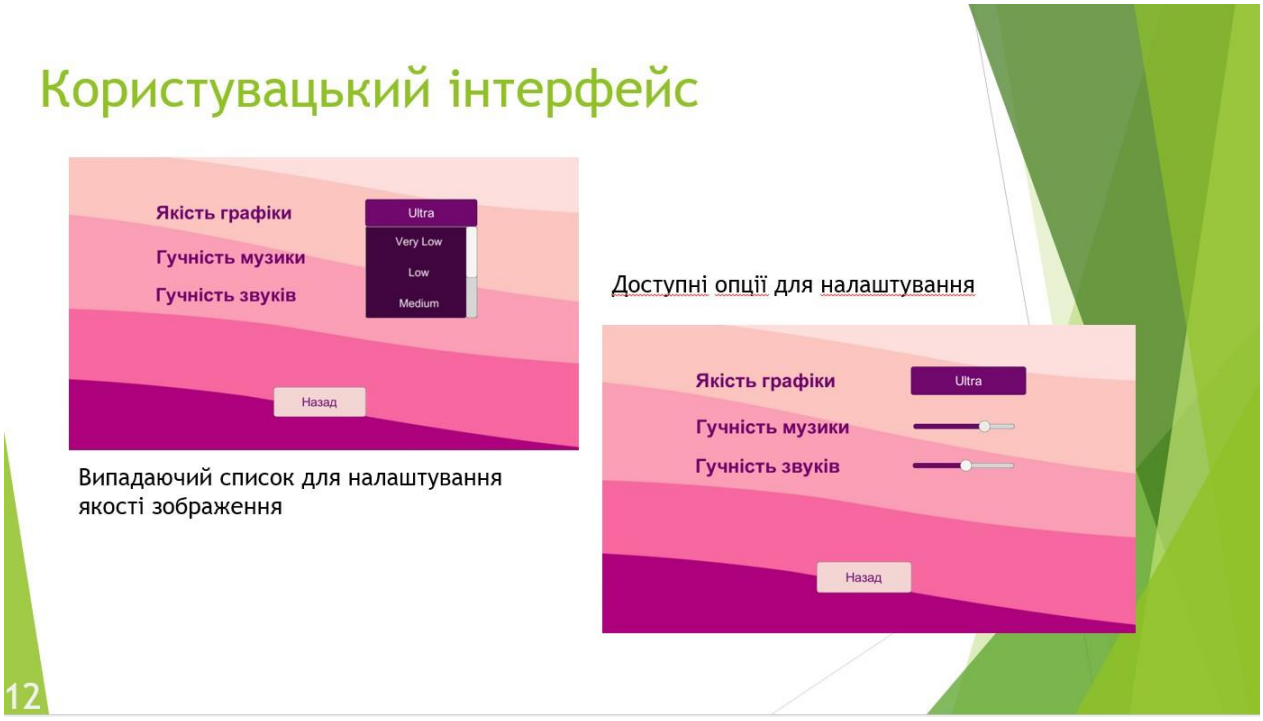


Рисунок Б.12 – Меню налаштувань

Рисунок Б.13 – Меню паузи та фінішу

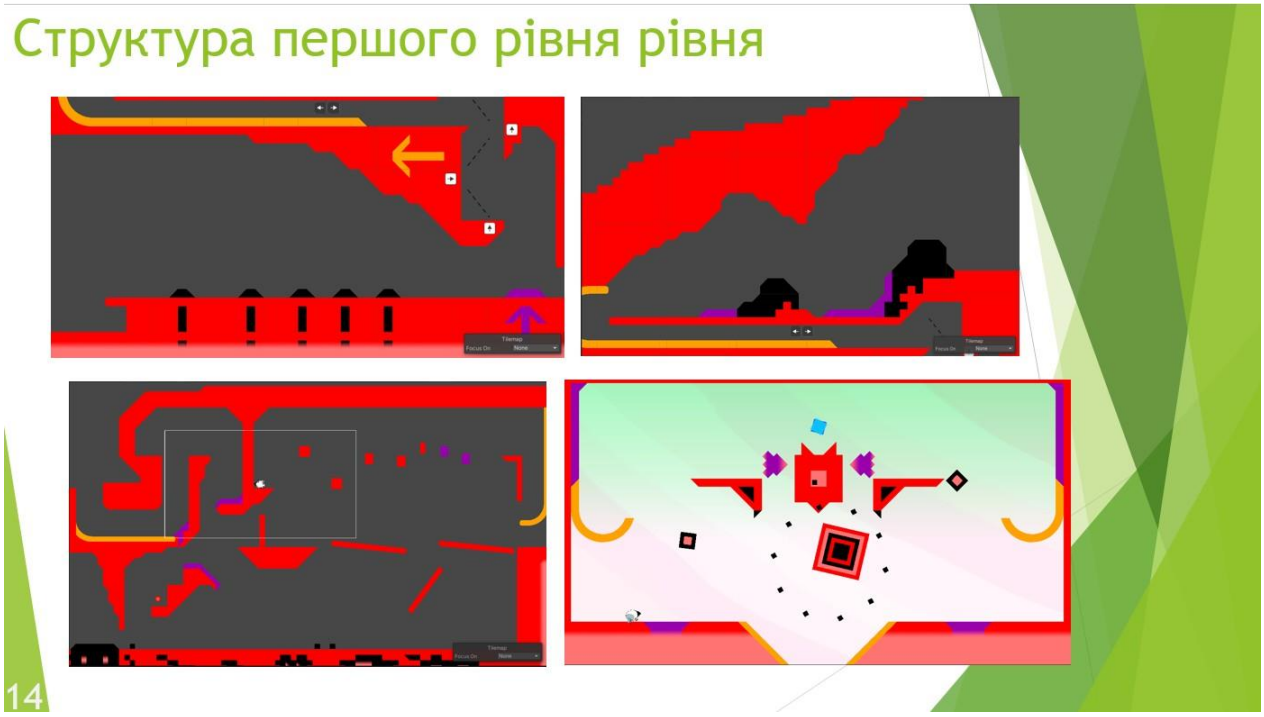


Рисунок Б.14 – Структура першого ігрового рівня

Рисунок Б.15 – Структура другого ігрового рівня

Рисунок Б.16 – Структура третього ігрового рівня

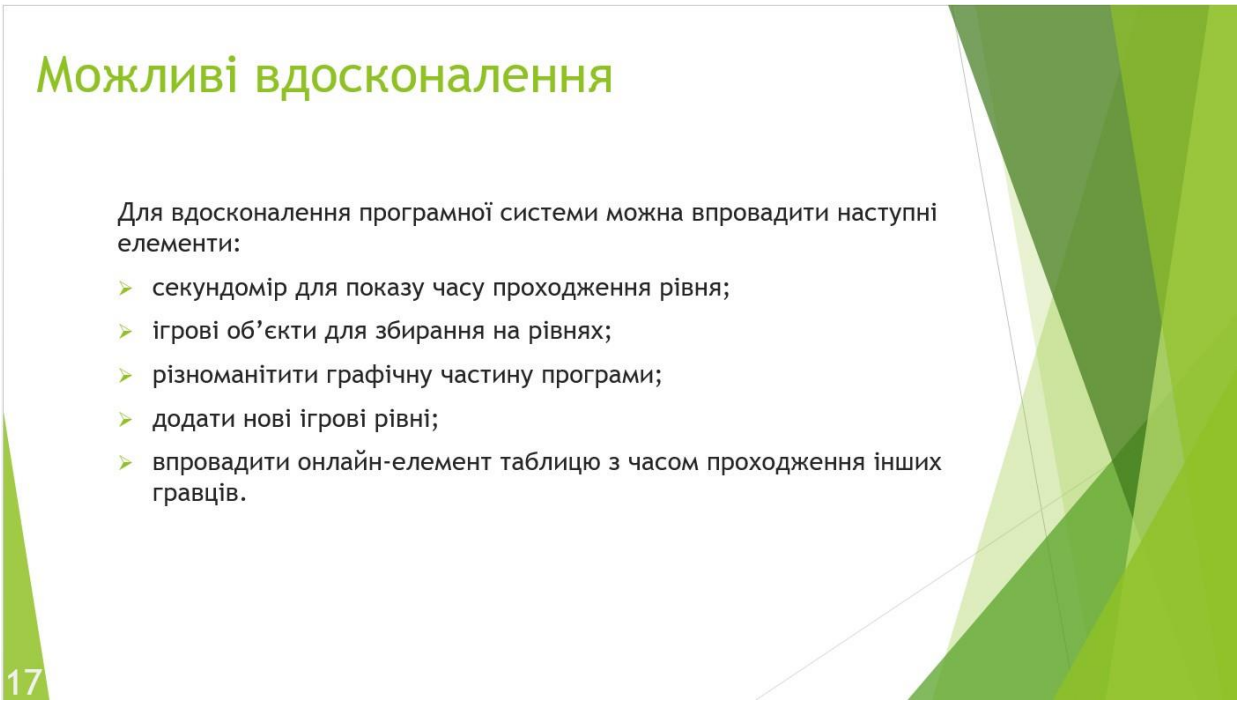


Рисунок Б.17 – Можливі вдосконалення

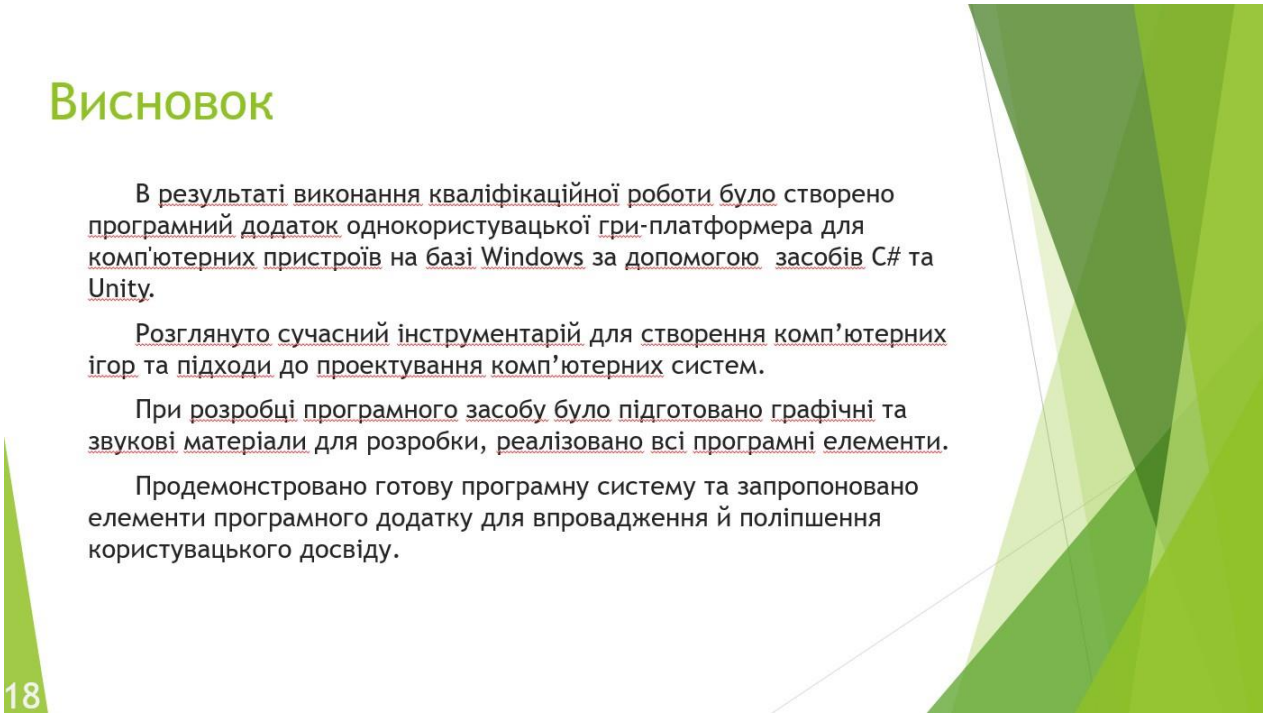


Рисунок Б.18 – Висновок

Додаток В

Графічні елементи

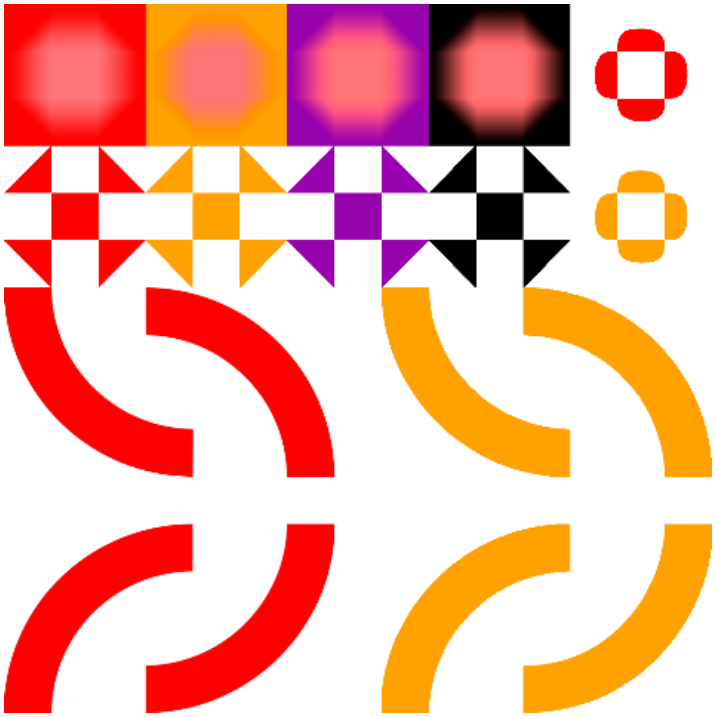


Рисунок В.1 – Набір спрайтів для побудови першого рівня

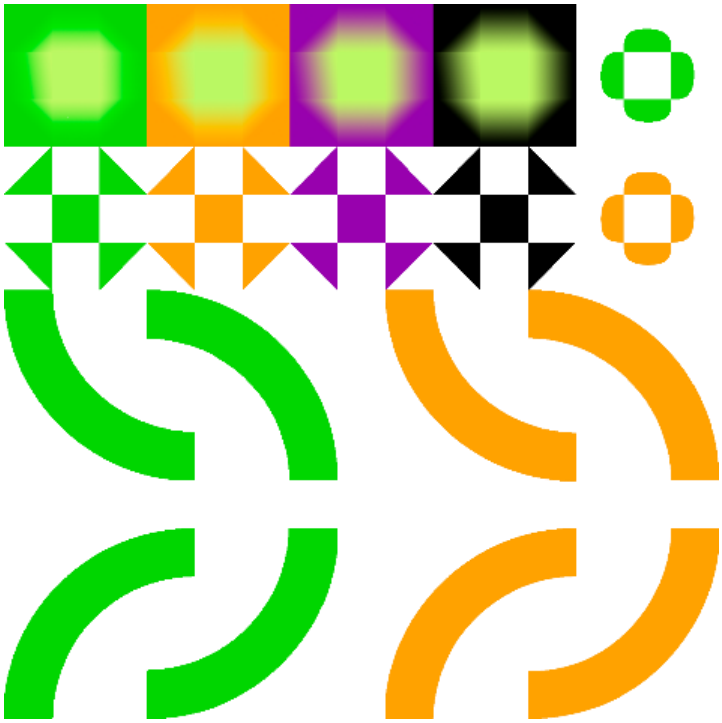


Рисунок В.2 – Набір спрайтів для побудови другого рівня

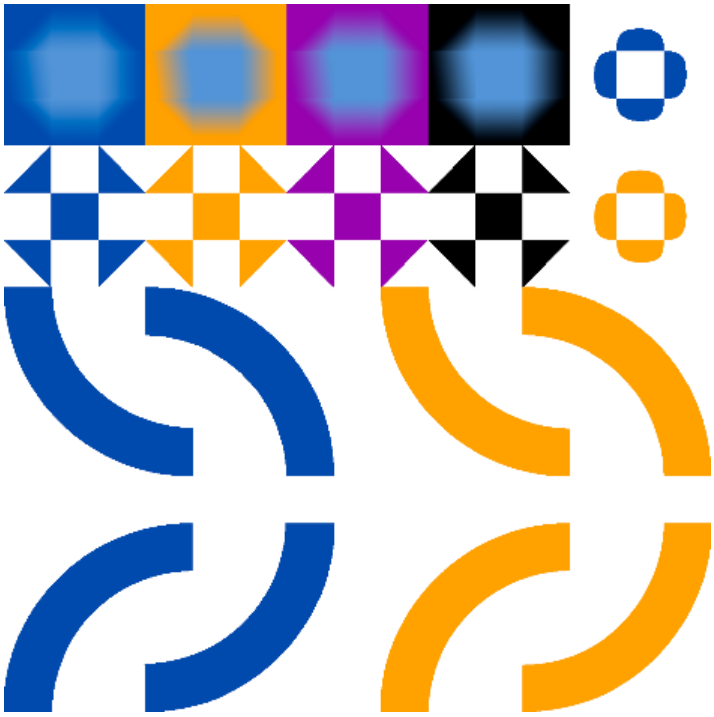


Рисунок В.3 – Набір спрайтів для побудови третього рівня

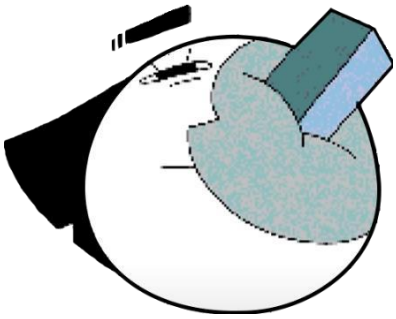


Рисунок В.4 – Спрайт головного героя [28]

Рисунок В.5 – Фонове зображення головного меню

Рисунок В.6 – Фонове зображення для першого рівня

Рисунок В.7 – Фонове зображення для другого рівня

Рисунок В.8 – Фонове зображення для першого рівня

Додаток Г.1

Лістинг класу «FinishMenu.cs»

```
using UnityEngine;
using UnityEngine.SceneManagement;

public class FinishMenu : MonoBehaviour
{
    public GameObject finishMenu;
    public void LoadMenu()
    {
        Time.timeScale = 1f;
        SceneManager.LoadScene("Main_menu");
    }

    void OnTriggerEnter2D(Collider2D collision)
    {
        if (collision.tag == "Player")
        {
            Time.timeScale = 0f;
            finishMenu.SetActive(true);
        }
    }
}
```

Додаток Г.2

Лістинг класу «MainMenu.cs»

```
using UnityEngine;
using UnityEngine.SceneManagement;

public class MainMenu : MonoBehaviour
{
   public void Playlevel_1()
   {
      SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex + 1);
   }
   public void Playlevel_2()
   {
      SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex + 2);
   }
   public void Playlevel_3()
   {
      SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex + 3);
   }

   public void ExitGame()
   {
      Debug.Log("game ended");
      Application.Quit();
   }
}
```

Додаток Г.3

Лістинг класу «PauseMenu.cs»

```
using UnityEngine;
using UnityEngine.SceneManagement;

public class PauseMenu : MonoBehaviour
{
    public GameObject pauseGameMenu;

    bool isPaused;

    void Update()
    {
        if (Input.GetKeyDown(KeyCode.Escape))
            if (isPaused) Resume();
            else Pause();
    }

    public void Resume()
    {
        Sounds.sound.pauseMenuMusic.Stop();
        Sounds.sound.backgroundMusic.UnPause();
        pauseGameMenu.SetActive(false);
        Time.timeScale = 1f;
        isPaused = false;
    }

    public void Pause()
    {
        Sounds.sound.backgroundMusic.Pause();
        Sounds.sound.pauseMenuMusic.Play();
        pauseGameMenu.SetActive(true);
        Time.timeScale = 0f;
        isPaused = true;
    }

    public void LoadMenu()
    {
        Time.timeScale = 1f;
        SceneManager.LoadScene("Main_menu");
    }
}
```

Додаток Г.4

Лістинг класу «InitAudio.cs»

```
using UnityEngine.Audio;
using UnityEngine;

public class InitAudio : MonoBehaviour
{
   public string volumeParameter = "MasterVolume";
   public AudioMixer audioMixer;
   void Start()
   {
      audioMixer.SetFloat(volumeParameter, PlayerPrefs.GetFloat(volumeParameter, -60f));
      PlayerPrefs.Save();
   }
}
```

Додаток Г.5

Лістинг класу «InitQuality.cs»

```
using UnityEngine;
using UnityEngine.UI;

public class InitQuality : MonoBehaviour
{
   public string qualityParameter = "QualityLevel";
   public Dropdown dropdown;
   void Start()
   {
      dropdown.value = PlayerPrefs.GetInt(qualityParameter, 5);
      QualitySettings.SetQualityLevel(PlayerPrefs.GetInt(qualityParameter, 5));
   }
}
```

Додаток Г.6

Лістинг класу «SetAudio.cs»

```
using UnityEngine.Audio;
using UnityEngine.UI;
using UnityEngine;

public class SetAudio : MonoBehaviour
{
    public string volumeParameter = "MasterVolume";

    public AudioMixer audioMixer;
    public Slider slider;

    private float volumeValue;
    public void OnSliderTrigger()
    {
        Sounds.sound.jumpSound.Play();
    }
    private void Awake()
    {
        slider.onValueChanged.AddListener(SliderValueChange);
    }

    private void Start()
    {
        volumeValue = PlayerPrefs.GetFloat(volumeParameter, Mathf.Log10(slider.value) * 20f);
        slider.value = Mathf.Pow(10f, volumeValue / 20f);
        PlayerPrefs.Save();
    }

    private void SliderValueChange(float value)
    {
        volumeValue = Mathf.Log10(value) * 20f;
        audioMixer.SetFloat(volumeParameter, volumeValue);
        PlayerPrefs.Save();
    }

    private void OnDisable()
    {
        PlayerPrefs.SetFloat(volumeParameter, volumeValue);
    }
}
```

Додаток Г.7

Лістинг класу «SetQuality.cs»

```
using UnityEngine.UI;
using UnityEngine;

public class SetQuality : MonoBehaviour
{
    public string qualityParameter = "QualityLevel";

    public Dropdown dropdown;

    private int qIndex;

    private void Awake()
    {
        dropdown.onValueChanged.AddListener(DropdownValueChange);
    }
    void Start()
    {
        dropdown.value = QualitySettings.GetQualityLevel();
        QualitySettings.SetQualityLevel(PlayerPrefs.GetInt(qualityParameter, 5));
    }
    private void DropdownValueChange(int value)
    {
        qIndex = value;
        QualitySettings.SetQualityLevel(value);
    }
    private void OnDisable()
    {
        PlayerPrefs.SetInt(qualityParameter, qIndex);
    }
}
```

Додаток Г.8

Лістинг класу «BossArm.cs»

```
using UnityEngine;

public class BossArm : MonoBehaviour
{
   public GameObject arm;
   private void OnTriggerEnter2D(Collider2D collision)
   {
      if (collision.CompareTag("Player")) arm.GetComponent<Animation>().Play("BossArm");
   }
}
```

Додаток Г.9

Лістинг класу «BossEye.cs»

```
using UnityEngine;

public class BossEye : MonoBehaviour
{
   public Transform target;

   public float speed;

   private bool toMove = true;
   private Vector3 position;
   private Vector3 standatrPos;
   private void Awake()
   {
      standatrPos = transform.position;
   }
   void FixedUpdate()
   {
      if (Mathf.Abs(transform.position.x) - Mathf.Abs(standatrPos.x) > 0.5f ||
Mathf.Abs(transform.position.y) - Mathf.Abs(standatrPos.y) > 0.5f)
         transform.position = standatrPos;
      else Follow();
   }
   void Follow()
   {
      toMove = position != target.position;
      position = target.position;
      if (toMove) transform.position = Vector3.MoveTowards(transform.position, position,
Time.deltaTime * speed);
   }
}
```

Додаток Г.10

Лістинг класу «Bullet.cs»

```
using UnityEngine;

public class Bullet : MonoBehaviour
{
    public float moveSpeed;
    public float stayTime;
    private Vector2 moveDirection;
    private void OnEnable()
    {
        Invoke("Destroy", stayTime);
    }

    void Update()
    {
        transform.Translate(moveDirection * moveSpeed * Time.deltaTime);
    }
    public void SetMoveDirection(Vector2 dir)
    {
        moveDirection = dir;
    }
    private void Destroy()
    {
        gameObject.SetActive(false);
    }
    private void OnDisable()
    {
        CancelInvoke();
    }
}
```

Додаток Г.11

Лістинг класу «BulletFire.cs»

```
using UnityEngine;

public class BulletFire : MonoBehaviour
{
    public static BulletFire fire { get; private set; }

    public int bulletAmont;
    public float timeInterval;
    public float startAngle;
    public float endAngle;

    public GameObject bulletPool;
    public Transform player;
    public Transform triggerPointBotLeft;
    public Transform triggerPointTopRight;
    public bool checkPosition;
    public bool stopFire;

    private void Awake()
    {
        fire = this;
    }

    void Start()
    {
        InvokeRepeating("Fire", 0f, timeInterval);
    }

    public void StopFire()
    {
        stopFire = true;
    }

    private bool CheckPosition ()
    {
        bool inRange = player.position.x > triggerPointBotLeft.position.x
                && player.position.x < triggerPointTopRight.position.x
                && player.position.y > triggerPointBotLeft.position.y
                && player.position.y < triggerPointTopRight.position.y;

        if (stopFire) inRange = false;
        return inRange;
    }

    private void Fire()
    {
        bool inPlace = true;

        if (checkPosition) inPlace = CheckPosition();
```

```
        if (inPlace)
        {
            float angleStep = (endAngle - startAngle) / bulletAmont;
            float angle = startAngle;

            for (int i = 0; i < bulletAmont + 1; i++)
            {
                float bulDirX = transform.position.x + Mathf.Sin((angle * Mathf.PI) / 180f);
                float bulDirY = transform.position.y + Mathf.Cos((angle * Mathf.PI) / 180f);

                Vector3 bulMoveVector = new Vector3(bulDirX, bulDirY, 0f);
                Vector2 bulDir = (bulMoveVector - transform.position).normalized;

                bulletPool = BulletPool.bulletPoolInstance.GetBullet();
                bulletPool.transform.position = transform.position;
                bulletPool.transform.rotation = transform.rotation;
                bulletPool.SetActive(true);
                bulletPool.GetComponent<Bullet>().SetMoveDirection(bulDir);

                angle += angleStep;
            }
        }
    }
}
```

Додаток Г.12

Лістинг класу «BulletPool.cs»

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

public class BulletPool : MonoBehaviour
{
   public static BulletPool bulletPoolInstance;
   public GameObject pooledBullet;
   private bool notEnoughtBulletsInPool = true;
   private List<GameObject> bullets;
   private void Awake()
   {
      bulletPoolInstance = this;
   }
   void Start()
   {
      bullets = new List<GameObject>();
   }
   public GameObject GetBullet()
   {
      if (bullets.Count > 0)
         for (int i = 0; i < bullets.Count; i++)
            if (!bullets[i].activeInHierarchy)
               return bullets[i];
      if(notEnoughtBulletsInPool) {
         GameObject bul = Instantiate(pooledBullet);
         bul.SetActive(false);
         bullets.Add(bul);
         return bul; }
      return null;
   }
}
```

Додаток Г.13

Лістинг класу «ButtonNew.cs»

```
using UnityEngine;
using UnityEngine.Events;

public class ButtonNew : MonoBehaviour
{
   private bool isPressed;

   public UnityEvent OnPressed;
   public void ToPress()
   {
      isPressed = true;
   }

   private void OnTriggerEnter2D(Collider2D collision)
   {
      if (collision.CompareTag("Player") && isPressed == false)
      {
         GetComponent<Animation>().Play("Button");
         isPressed = true;
         OnPressed.Invoke();
      }
   }
}
```

Додаток Г.14

Лістинг класу «CameraControl.cs»

```
using UnityEngine;

public class CameraControl : MonoBehaviour
{
   public Transform player;
   public Transform standpoint;
   public Transform cameraCheck;
   public float minZoom = 5;
   public float maxZoom = 9;
   public float cameraSpeed = 3;

   private bool isFixed;
   private Vector3 position;

   private void Awake()
   {
      if (!player) player = FindObjectOfType<Player>().transform;
   }

   private void Update()
   {
      if (cameraCheck.position.x < player.position.x) FixCamera();
      else Follow();
   }


   void Follow()
   {
      position = player.position;
      position.z = -10f;
      if (position.x < 0) position.x = 0;
      if (position.y < 0) position.y = 0;
      transform.position = Vector3.Lerp(transform.position, position, Time.deltaTime *
cameraSpeed);
      Camera.main.orthographicSize = Mathf.MoveTowards(Camera.main.orthographicSize,
minZoom, Time.deltaTime * cameraSpeed);
   }

   void FixCamera()
   {
      position = standpoint.position;
      position.z = -10f;
      transform.position = Vector3.Lerp(transform.position, position, Time.deltaTime *
cameraSpeed);
      Camera.main.orthographicSize = Mathf.MoveTowards(Camera.main.orthographicSize,
maxZoom, Time.deltaTime * cameraSpeed);
   }
}
```

Додаток Г.15

Лістинг класу «Door.cs»

```
sing UnityEngine;

public class Door : MonoBehaviour
{
   public Transform player;
   public float bottomPosition = 10f;

   private float startY;
   private void Awake()
   {
      startY = transform.position.y;
   }
   void Update()
   {
      if (transform.position.x < player.position.x && transform.position.y >= bottomPosition)
         transform.position = new Vector2(transform.position.x, transform.position.y -
Mathf.Abs(bottomPosition) * Time.deltaTime);
      if (transform.position.x > player.position.x && transform.position.y <= startY)
         transform.position = new Vector2(transform.position.x, transform.position.y +
Mathf.Abs(startY) * Time.deltaTime);
   }
}
```

Додаток Г.16

Лістинг класу «Entity.cs»

```
using UnityEngine;

public class Entity : MonoBehaviour
{
   public Rigidbody2D rb;
   public Vector2 moveVector;

   public bool isGrounded = false;
   public bool isDead = false;

   protected void Awake()
   {
      rb = GetComponent<Rigidbody2D>();
   }

   protected virtual void Death() { }
}
```

Додаток Г.17

Лістинг класу «FocusOnTarget.cs»

```
using UnityEngine;

public class FocusOnTarget : MonoBehaviour
{
   public Transform target;

   private void Update()
   {
      Vector3 difference = target.transform.position - transform.position;
      float rotateZ = Mathf.Atan2(difference.y, difference.x) * Mathf.Rad2Deg;
      transform.rotation = Quaternion.Euler(0f, 0f, rotateZ);
   }

}
```

Додаток Г.18

Лістинг класу «Follow.cs»

```
using System.Collections;
using UnityEngine;

public class Follow : MonoBehaviour
{
   public float speed;
   public Transform target;
   public Transform triggerPointBotLeft;
   public Transform triggerPointTopRight;

   void FixedUpdate()
   {
      if (CheckPosition()) ToFollow();
   }
   bool CheckPosition()
   {
      bool inRange = target.position.x > triggerPointBotLeft.position.x
               && target.position.x < triggerPointTopRight.position.x
               && target.position.y > triggerPointBotLeft.position.y
               && target.position.y < triggerPointTopRight.position.y;

      return inRange;
   }

   void ToFollow ()
   {
      Vector3 direction = (target.position - transform.position).normalized;
      GetComponent<Rigidbody2D>().AddForce(direction * speed);
   }
   private IEnumerator AnimationDeley()
   {
      GetComponent<Rigidbody2D>().velocity = new Vector2(0, 0);
      GetComponent<Animation>().Play("Button");
      yield return new WaitForSeconds(1);
      Destroy(gameObject);
   }
   private void OnCollisionEnter2D(Collision2D collision)
   {
      StartCoroutine(AnimationDeley());
   }
}
```

Додаток Г.19

Лістинг класу «MoveRespawn.cs»

```
using UnityEngine;

public class MoveRespawn : MonoBehaviour
{
   public Transform player;
   public Transform cameraCheck;

   void Update()
   {
      if (cameraCheck.position.x < player.position.x) transform.position = cameraCheck.position;
   }
}
```

Додаток Г.20

Лістинг класу «PlateNew.cs»

```
using UnityEngine;
using UnityEngine.Events;

public class PlateNew : MonoBehaviour
{
   private bool isPressed;

   public  UnityEvent  OnPress;
   public UnityEvent OnLeave;
   public void ToPress()
   {
      isPressed = true;
   }

   private void OnTriggerStay2D(Collider2D collision)
   {
      if (isPressed == false) GetComponent<Animation>().Play("Plate_down");
      isPressed = true;
      OnPress.Invoke();
   }

   private void OnTriggerExit2D(Collider2D collision)
   {
      if (isPressed == true) GetComponent<Animation>().Play("Plate_up");
      isPressed = false;
      OnLeave.Invoke();
   }
}
```

Додаток Г.21

Лістинг класу «PlatformMove.cs»

```
using UnityEngine;

public class PlatformMove : MonoBehaviour
{
    private bool movingRight;
    private bool movingUp;
    private bool moveVertical;
    private bool moveHorizontal;
    private Quaternion startRotation;
    private Vector3 startPosition;

    public float verticalRange = 0f;
    public float horizontalRange = 0f;
    public float moveSpeed = 0f;

    public bool rotateClockwise;
    public bool rotateCounterclockwise;
    public bool oneMove;

    public void Activate()
    {
        gameObject.GetComponent<PlatformMove>().enabled = true;
    }
    public void Deactivate()
    {
        gameObject.GetComponent<PlatformMove>().enabled = false;
    }

    private void Awake()
    {
        startPosition = transform.localPosition;
        startRotation = transform.rotation;
        if (verticalRange != 0) moveVertical = true;
        if (horizontalRange != 0) moveHorizontal = true;
    }
    private void FixedUpdate()
    {
        if (moveVertical) MovePlatformVertical(moveSpeed, verticalRange);
        if (moveHorizontal) MovePlatformHorizontal(moveSpeed, horizontalRange);
        if (rotateClockwise) RotatePlatformClockwise(moveSpeed);
        if (rotateCounterclockwise) RotatePlatformCounterclockwise(moveSpeed);
    }
    void MovePlatformHorizontal(float speed, float range)
    {
        if (transform.localPosition.x > range)
        {
            movingRight = false;
            if (oneMove) moveHorizontal = false;
        }
```

```
        if (transform.localPosition.x < -range)
        {
            movingRight = true;
            if (oneMove) moveHorizontal = false;
        }

        if (!movingRight) speed = -speed;
        transform.localPosition = new Vector2(transform.localPosition.x + speed,
transform.localPosition.y);
    }
    void MovePlatformVertical(float speed, float range)
    {
        if (transform.localPosition.y > range)
        {
            movingUp = false;
            if (oneMove) moveVertical = false;
        }
        if (transform.localPosition.y < -range)
        {
            movingUp = true;
            if (oneMove) moveVertical = false;
        }

        if (!movingUp) speed = -speed;
        transform.localPosition = new Vector2(transform.localPosition.x, transform.localPosition.y +
speed);
    }

    void RotatePlatformClockwise (float speed)
    {
        speed = -speed;
        transform.Rotate(new Vector3(0f, 0f, speed * 30));
        if (startRotation == transform.rotation && oneMove) rotateClockwise = false;
    }
    void RotatePlatformCounterclockwise(float speed)
    {
        transform.Rotate(new Vector3(0f, 0f, speed * 30));
        if (startRotation == transform.rotation && oneMove) rotateCounterclockwise = false;
    }
}
```

Додаток Г.22

Лістинг класу «PlatformToStart.cs»

```
using UnityEngine;

public class PlatformToStart : MonoBehaviour
{
   private Quaternion startRotation;
   private Vector3 startPosition;

   public float speed;

   private void Awake()
   {
      startPosition = transform.position;
      startRotation = transform.rotation;
   }
   void Update()
   {
      if (!gameObject.GetComponent<PlatformMove>().enabled) MoveToStart();
   }

   public void MoveToStart ()
   {
      if (transform.position != startPosition) transform.position =
Vector3.MoveTowards(transform.position, startPosition, speed);
      if (Quaternion.Angle(transform.rotation, startRotation) > 1) transform.Rotate(new
Vector3(0f, 0f, speed * Time.deltaTime * 60));
   }
}
```

Додаток Г.23

Лістинг класу «Player.cs»

```
using UnityEngine;

public class Player : Entity
{
    public GameObject respawn;

    public float speed = 6f;
    public float jumpForce = 400f;
    private Vector3 luft;

    private void Start()
    {
        Sounds.sound.backgroundMusic.Play();
    }
    void Update()
    {
        if (Input.GetKey(KeyCode.R)) isDead = true;
        if (isDead) Death();
    }

    void FixedUpdate()
    {
        if (Input.GetAxis("Vertical") > 0)
        {
            luft = transform.position;
            luft.y += 0.001f;
            transform.position = luft;
        }
        if (Input.GetAxis("Vertical") > 0 && isGrounded) Jump();
        if (Input.GetAxis("Vertical") < 0 && !isGrounded) Fall();
        if (Input.GetButton("Horizontal")) Walk();
    }

    void Walk()
    {
        moveVector.x = Input.GetAxis("Horizontal");
        rb.velocity = new Vector2(moveVector.x * speed, rb.velocity.y);
    }

    void Jump()
    {
        Sounds.sound.jumpSound.pitch = Random.Range(0.9f, 1.1f);
        Sounds.sound.jumpSound.Play();

        rb.AddForce(Vector2.up * jumpForce);
        isGrounded = false;
    }

    void Fall()
```

```
    {
        rb.AddForce(Vector2.down * jumpForce/10);
    }

    protected override void Death()
    {
        Sounds.sound.deathSound.Play();

        rb.velocity = new Vector2(0, 0);
        moveVector = rb.velocity;
        transform.position = respawn.transform.position;

        isDead = false;
    }

    private void OnCollisionEnter2D(Collision2D collision)
    {
        if (collision.gameObject.layer == LayerMask.NameToLayer("Ground")
            || collision.gameObject.layer == LayerMask.NameToLayer("SpeedBoost")
            || collision.gameObject.layer == LayerMask.NameToLayer("JumpBoost"))
        {
            isGrounded = true;
            if (collision.gameObject.layer == LayerMask.NameToLayer("Ground"))
                rb.velocity = new Vector2(0, rb.velocity.y);
        }
        if (collision.gameObject.layer == LayerMask.NameToLayer("DeathZone"))
            isDead = true;

        Sounds.sound.fallSound.pitch = Random.Range(0.9f, 1.1f);
        Sounds.sound.fallSound.Play();
    }
}
```

Додаток Г.24

Лістинг класу «Sounds.cs»

```
using UnityEngine;

public class Sounds : MonoBehaviour
{
    public static Sounds sound { get; private set; }

    public AudioSource jumpSound;
    public AudioSource speedBoost;
    public AudioSource fallSound;
    public AudioSource deathSound;
    public AudioSource pauseMenuMusic;
    public AudioSource backgroundMusic;

    private void Awake()
    {
        sound = this;
    }
}
```

Додаток Г.25

Лістинг класу «SpeedBoost.cs»

```
using UnityEngine;
public class SpeedBoost : MonoBehaviour
{
   Rigidbody2D rb;
   public float Boost = 0.3f;
   public float maxSpeed = 30f;
   bool isActive;
   void Update()
   {
      if(isActive) Activate();
   }
   private void OnCollisionEnter2D(Collision2D collision)
   {
      Sounds.sound.speedBoost.Play();
      isActive = true;
      rb = collision.gameObject.GetComponent<Rigidbody2D>();
   }
   private void OnCollisionExit2D(Collision2D collision)
   {
      Sounds.sound.speedBoost.Stop();
      isActive = false;
   }
   void Activate()
   {
      float BoostNeg = -Boost;
      if (rb.velocity.magnitude <= maxSpeed) {
         float bst = rb.velocity.x < 0 ? BoostNeg : Boost;
         rb.velocity = new Vector2(rb.velocity.x + bst, rb.velocity.y);}
   }
}
```

Додаток Г.26

Лістинг класу «Stoper.cs»

```
using UnityEngine;

public class Stoper : MonoBehaviour
{
   public  Transform player;
   public Transform undoRangeX;

   void Update()
   {
      if (player.position.x > undoRangeX.position.x) Undo();
   }

   public void Undo ()
   {
      Destroy(gameObject);
   }

   private void OnTriggerStay2D(Collider2D collision)
   {
      collision.GetComponent<Transform>().position = transform.position;
   }
}
```