Форма № 24

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Інститут інформатики та радіоелектроніки,
Факультет комп'ютерних наук і технологій
(повне найменування інституту, назва факультету)

Кафедра програмних засобів
(повне найменування кафедри)

Пояснювальна записка

до дипломного проєкту (роботи)
бакалавр
(ступінь вищої освіти)

на тему РОЗРОБКА ІГРОВОЇ ПРОГРАМИ «TOWER RUNNER» ДЛЯ МОБІЛЬНОГО ТЕЛЕФОНУ
DEVELOPMENT OF THE TOWER RUNNER GAME SOFTWARE FOR A MOBILE PHONE

Виконав: студент(ка) 4 курсу, групи КНТ-217
Спеціальності 122 Комп'ютерні науки
(код і найменування спеціальності)

Освітня програма (спеціалізація)
Комп'ютерні науки

Лизя Є. С.
(прізвище та ініціали)

Керівник Субботін С. О.
(прізвище та ініціали)

2021

Форма № 25

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»
(повне найменування закладу вищої освіти)

Інститут, факультет ІІРЕ, ФКНТ
Кафедра програмних засобів
Ступінь вищої освіти бакалавр
Спеціальність 122 Комп'ютерні науки
(код і найменування)
Освітня програма (спеціалізація) Комп'ютерні науки
(назва освітньої програми (спеціалізації)

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.
_____С.О. Субботін
"____" _____20___року

З А В Д А Н Н Я

НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

Лизі Євгенія Сергійовича
(прізвище, ім'я, по батькові)

1. Тема проєкту (роботи) Розробка ігрової програми «Tower Runner» для мобільного телефону. Development of the tower runner game software for a mobile phone.

керівник проєкту (роботи) Субботін Сергій Олександрович., д.т.н., професор ,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом закладу вищої освіти від "30" березня 2021 року № 103

2. Строк подання студентом проєкту (роботи) 17 травня 2021 року

3. Вихідні дані до проєкту (роботи) рекомендована література, технічне завдання

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз предметної області. 2. Проєктування програмного забезпечення. 3. Конструювання програмного забезпечення. 4. Інструкції з використання програмного продукту. 5. Текст програми.

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

6. Консультанти розділів проєкту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-5 Основна частина	Субботін С. О., професор		
Нормоконтроль	Дейнега Л.Ю., ст.викладач		

7. Дата видачі завдання “ 28 ” лютого 2021 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Постановка завдання роботи	1 тиждень	Завдання, ТЗ
2	Аналіз предметної області	2-3 тижні	Розділ 1
3	Розробка архітектури програми	4 тиждень	Розділ 2
4	Розробка програми. Тестування та експериментальне дослідження програми.	5-6 тижні	Розділ 3
5	Розробка інструкцій з використання програмного продукту	7 тиждень	Розділ 4
6	Оформлення пояснювальної записки та документів до неї. Нормоконтроль та рецензування	8 тиждень	ПЗ, Додатки
7	Захист роботи	9 тиждень	

Студент(ка) _____ Лизя Є. С.
(підпис) (прізвище та ініціали)

Керівник проєкту (роботи) _____ Субботін С.О.
(підпис) (прізвище та ініціали)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи бакалавра: 137 с., 35 рис., 5 табл, 2 дод., 40 джерел.

ІТ-ТЕХНОЛОГІЇ, ІГРОВИЙ ЗАСТОСУНОК, 3D ГРА, ANDROID ГРА

Об’єкт дослідження – процес розроблення ігрових програмих.

Предмет дослідження – засоби розробки ігрової програми для мобільного телефону.

Мета роботи – програмна реалізація ігрової програми в жанрі платформеру для збільшення кількості послуг, що надаються власним підприємством.

Матеріали, методи та технічні засоби: інтегроване середовище розробки Rider, мова програмування C#, ігровий рушій Unity3d.

Результати. Розроблено ігровий застосунок для мобільного телефону з операційною системою Android з такими характеристиками: система меню навігації (можливість змінювати ігровий рівень), оптимізований алгоритм бескінечної генерація ігрового рівня, меню налаштування: (включення та виключення вібрації, зменшення та збільшення гучності), підрахунок монет та балів за проходження рівня, запуск на мінімальній версії мобільної ОС – Android 6.0.

Висновки. Розроблена програмна реалізація ігрової програми в жанрі платформеру для операційної системи Android.

Галузь використання – використання застосунків на телефонах з операційною системою Android 6.0 і вище.

ABSTRACT

Explanatory note to the bachelor's thesis: 137 pages, 35 figures, 5 tables, 2 appendicxes, 40 sources.

IT TECHNOLOGIES, GAME APPLICATION, 3D GAME, ANDROID GAME

The object of research is the development of game programs.

The subject of research - methods and tools for developing a game program for a mobile phone.

The purpose of the work is the software implementation of the game program in the genre of platformer.

Materials, methods and technical means: Rider integrated development environment, C # programming language, Unity3d game engine.

Results. The game application «Tower Runner» has been developed for a mobile phone with the Android operating system. The game has these characterstics: navigation menu system (the ability to change the game level), optimized algorithm for infinite game level generation, adjustment menu (on and off vibration, decrease and increase volume), counting coins and points for passing the level, running on the minimum version of the mobile OS - Android 6.0.

Findings. The software implementation «Tower Runner» in the genre of platformer has been developed for the Android operating system.

The areas of application are playing games on phones with Android 6.0 and higher.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

API – програмний інтерфейс;

CPU – центральний процесор;

CSS3 – Cascading Style Sheets 3;

DOM – Document Object Model;

JSON –JavaScript Object Notation;

HTML – HyperText Markup Language;

SASS – Syntactically Awesome Stylesheets;

SDK – Software development kit;

VR – віртуальна реальність;

БД – база даних;

ОС – операційна система.

ВСТУП

Сучасний ринок ігрових додатків значно змінився після того, як були розроблені такі магазини як Google Play та App Store. Велика кількість різноманітних програм почали публкікуватися, серед яких перевжна кількість є ігровими застосунками. Виникає питання, які технології використовуються для створення подібних програм. Найпоширенішою, найзручшнішою та найпотужнішою технологією вважається ігровий рушій Unity3d, який дозволяє розробляти додатки будь-якого типу: автоматизаційні програми, 3D візуалізації, VR та AR застосунки, вебзастосунки. Все це можливо для будь-якої сучасної операційної системи, як для настільних комп'ютерів так і для мобільних пристроїв.

На сьогодні немає дефіциту якісних мобільних ігрових застосувань. Але незважаючи на це – розробка мобільного ігрового додатку є комплексним процесом, який потребує використання багатьох технологій.

Ціллю данної роботи є розробка ігрового додатку для операційної системи Android з використанням ігрового рушія Unity3d для збільшення кількості послуг, що надаються власним підриємством.

Вимоги до розроблюваного застосунку:

- система меню навігації: можливість змінювати ігровий рівень;
- оптимізований алгоритм бескінечної генерація ігрового рівня;
- меню налаштування: включення та виключення вібрації, зменшення та збільшення гучності;
- підрахунок монет та балів за проходження рівня;
- запуск на мінімальній версії мобільної ОС – Android 6.0.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

Платформер – жанр відеоігор, ігровий процес в якому складається зі стрибків персонажа по різноманітних платформах (звідси і назва) та через перешкоди, збирання предметів, звичайно необхідних для завершення рівня [1].

1.1 Становлення жанру

Першими платформерами були: Frogs (1978), де були стрибки, але не було платформ [2] і Space Panic (1980), де були платформи, але не було стрибків [3]. В Donkey Kong (1981) вперше з'явився скроллінг [4]. Першим справжнім платформером стала гра Pitfall [5], але справжня слава прийшла до жанру після виходу Super Mario Bros [6]. 1985 року. З того часу двовимірні платформери продовжують випускати до цих пір на різних ігрових платформах.

Перші плаформери були обмежені статичними ігровими світами, які займали один ігровий екран, а герой зображався в профіль. Персонаж лазив вгору і вниз по сходах або стрибав з платформи на платформу, часто б'ючись з супротивниками і збираючи пауер-апи. Незабаром процес проходження рівня перестав бути в основному вертикальним і змінився на горизонтальний з появою довгих багатоекранних ігрових світів з прокруткою. Вважається, що початок цьому поклала випущена фірмою Activision в 1982 році гра Pitfall для консолі Atari 2600. Manic Miner (1983) [7] і її продовження Jet Set Willy (1984) [8] були найбільш популярними платформерами для тодішніх домашніх комп'ютерів.

У 1985 році фірма Nintendo випустила для приставки Nintendo Entertainment System революційний платформер Super Mario Bros [9]. На відміну від попередника, Mario Bros. (1983), рівень займав декілька екранів. Гра мала великі і складні рівні, і стала прикладом для наступних творців ігор, навіть сьогодні багато людей вважають її однією з найкращих відеоігор. Гра мала фантастичну популярність і продавалася величезними тиражами. Для багатьох

людей вона стала першим в їхньому житті платформером, а головний герой, водопровідник Mapio, став символом фірми Nintendo.

1987 вийшла Castlevania, яка мала незвичайний для плаформерів фентезійний антураж і похмуру атмосферу [10]. 1988 на Apple II була зроблена гра Prince of Persia, яка вирізнялася глибиною ігрового процесу і графікою поряд з численними пастками та оригінальними головоломками [11].

У 1991 Sega випустила Sonic the Headhog, першу з другої, поряд з Mario, найуспішнішої серії платформерів. Гра ефективно використовувала можливості 16-бітної консолі, завдяки чому була одним з найшвидших платформерів свого часу [12].

1.2 Нескінчені ранери

Нескінченні ранери — ігри-платформери, геймплей яких полягає у постійному русі персонажа вперед по теоретично нескінченному світі гри. Управління гри обмежене можливістю стрибка персонажа, атаки, або виконання спеціальних дій. Мета цих ігор — пробігти якомога далі, перш ніж персонаж помре. Нескінченні бігалки зазнали особливого успіху на мобільних платформах. Вони добре підходять для невеликого набору елементів управління, часто обмежується одним натисканням екрану для стрибків.

Temple Run (2011) [13] та його наступник Temple Run 2 [14] стали особливо популярними іграми цього жанру. Остання стала найшвидше поширюваною мобільною грою в світі у січні 2013 року, з 50 млн. завантаженнями в межах тринадцяти днів. Інші успішні «нескінченні ранери» включають Subway Surfers [15], Sonic Dash [16], Rayman Jungle Run [17], Stampede Run [18], Snowden Run 3D [19] і Jetpack Joyride [20].

1.3 Тривимірні платформери

Поява тривимірних платформерів зі зростанням потужності ігрових платформ принесла зміну кінцевих цілей в таких іграх. У більшості двовимірних платформерів гравцеві потрібно було досягти на рівні тільки однієї мети, однак у багатьох тривимірних платформерах, кожен рівень необхідно прочісувати, збираючи фрагменти головоломок, зірки чи предмети для прокачки персонажа. Це дало можливість більш ефективного використання великих тривимірних областей і винагороджувало гравця за ретельне дослідження рівня, але деякі гравці вважають збирання незліченних дрібничок більш нудним заняттям, ніж випробування на пам'ять і реакцію. Donkey Kong 64 (1999) була розкритикована за те, що гравцеві доводилося часто перемикатися між п'ятьма різними ігровими героями, щоб отримати банани різних кольорів та інші предмети [21]. Однак не всі тривимірні платформери були такими, найяскравішим прикладом є Crash Bandicoot (1996) [22]. Ця гра залишалося вірною традиції двовимірних платформерів і в ній використовувалися досить плоскі рівні, наприкінці яких розташовувалася ігрова мета. Spyro the Dragon (1998) мала нелінійне проходження і справжнє вільне переміщення в трьох вимірах в рамках рівня [23].

1.4 Аналіз існуючих аналогів гри «Tower Runner»

Аналіз буде проводитися по таким критеріям: графіка, управління, користувацький інтерфейс, музикальний супровід.

1.4.1 Helix Crush

Helix Crush – аркадна гра платформер (рис 1.1), де користувач повинен розбивати фруктові дольки в ритмі з мелодією та оминати всі перешкоди. Графіка складається з блоків, фону та візуальних ефектів. Має різний спектр кольорів. Досить яскрава, але різко сприймається зором. Управління відбувається за допомогою проведення пальцем, зручно. Музикальний супровід багатий на різні композиції. Користувацький інтерфейс зручний [24].

Рисунок 1.1 – Helix Crush

1.4.2 Dancing Road: Color Ball Run!

Dancing Road: Color Ball Run! – гра (рис. 1.2) в якій потрібно збирати шари того ж кольору, що і головний герой. Управління виконується пересуванням шару по платформі вліво та вправо. Графіка приємна, достатньо атмосферна, безліч високоякісний візуальних ефектів, а також динамічна анімація. Присутня велика кількісь композицій. Користувацький інтерфейс зручний [25].

Рисунок 1.2 – Dancing Road: Color Ball Run!

1.4.3 Jump for Queen 2020: Rescue on the Helix

Jump for Queen 2020: Rescue on the Helix – гра (рис. 1.3), в якій потрібно спуститися до низу і зібрати всі ігрові елементи. Графіка трохи застаріла: якість текстур низька, анімація пересування головного героя примітивна. Музикальний супровід на низькому рівні, весь час повторюється одна і та сама композиція. Користувацький інтерфейс не зручний через не зручне росположення графічних елементів. Управління зручне, відбувається за допомогою проведення пальцем [26].

Рисунок 1.3 – Jump for Queen 2020: Rescue on the Helix

1.4.4 Helix Stack Ball

Helix Stack Ball – гра (рис. 1.4), в якій потрібно спуститися до низу, розбиваючи різнокольорові блоки та оминаючи чорні блоки. Управління відбувається за допомогою проведення пальцем, досить плавно. На повторенні лише одна композиція. Користувацький інтерфейс дуже зручний, все інтуїтивно зрозуміло, зручне росположення графічних елементів. Графіка не має багатьох візуальних ефектів, але на око сприймається достатньоє приємно. Має шикорий спектр кольорів [27].

Рисунок 1.4 – Jump for Queen 2020: Rescue on the Helix

1.5 Порівняння аналогів

В ході проведених досліджень, були вивчені та проаналізовані аналоги ігрових додатків Tower Runner. На таблиці 1.1 приведено порівняння згідно зазначених критеріїв по п’ятибальній шкалі (від 1 до 5).

Таблиця 1.1 - Порівняння аналогів

Додаток	Графіка	Управління	Користувацький інтерфейс	Музикальний супровід
Helix Stack Ball	4	5	5	3
Jump for Queen 2020: Rescue on the Helix	2	5	3	3
Dancing Road: Color Ball Run!	5	5	5	5
Helix Crush	4	5	5	5

1.6 Постановка задач дипломної роботи

В ході проведених досліджень були вивчені та проаналізовані ігрові додатки в жанрі платформеру.

Мета роботи – програмна реалізація ігрової програми в жанрі платформеру на операційну систему Android.

Задачі роботи:

- аналіз вимог до системи;
- розробка архітектури системи;
- розробка дизайну клієнтської частини;
- вибір інструментарію розробки;
- програмна реалізація ігрової програми;
- розробка інструкцій з використання програми.

1.7 Висновки до розділу 1

В даному розділі вирішено задачу аналізу предметної області. На його основі поставлено завдання до дипломної роботи.

Вирішено завдання розкрити суть жанру платформеру, а також історію його становлення. З’ясували, що є типи платформерів як: нескінчені ранери і тривимірні платформері, до яких і належить розроблювана програма. Проаналізовані існуючі аналоги розроблюваного додатку, серед яких такі проекти: Helix Crush, Dancing Road: Color Ball Run!, Jump for Queen 2021: Rescue on the Helix, Helix Stack Ball.

Вирішено завдання порівняти існуючі аналоги розроблюваного додатку. Побудовано таблицю порівняння зазначених аналогів по таким критеріям: графіка, управління, користувацький інтерфейс, музикальний супровід. Це головні критерії, які впливають на тривалість гри кінцевого користувача. Ця таблиця порівняння дозволяє визначити, чи буде гра успішною, чи ні. Отже, для успішного виконання мети даного проекту, розроблюваний додаток повинен відповідати найвищим балам по зазначеним критеріям.

Вирішено завдання поставити задачі до дипломної роботи на основі проведених досліджень.

В наступному розділі необхідно проаналізувати вимоги до системи, описати розробку архітектури системи, обрати фреймворк для розробки, обрати середовище програмування.

2 ПРОЄКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Даний розділ описує проєтування програмного забезпечення.

Розкриваються такі питання: аналіз та визначенню вимог до системи, опис розробки архітектури системи, аналіз та вибір ігрового рушія, аналіз та вибір інтегрованої середи програмування.

2.1 Аналіз вимог до системи

Розроблювана ігрова програма в жанрі платформеру повинна бути оптимізованою для мобільних пристроїв, зручною у використанні та цікава в процесі використання.

Програмний продукт повинен містити наступний функціонал:

- пересування по ігровій локації зверху в вниз;
- логіка виграшу та програшу;
- генерація бескінечних рівнів;
- навігація по рівням складності;
- налаштування гри;

2.2 Розробка архітектури системи

Для реалізації проекту обраний шаблонний підхід MVC, який дозволяє розділити систему на 3 розділи:

- model;
- view;
- controller.

Модель надає дані і методи роботи з ними: запити до бази даних, перевірка на коректність. Модель не залежить від представлення (не знає як дані візуалізувати) і контролера (не має точок взаємодії з користувачем), просто

надаючи доступ до даних і управління ними. Модель будується таким чином, щоб відповідати на запити, змінюючи свій стан, при цьому може бути вбудовано повідомлення «спостерігачів». Модель, за рахунок незалежності від візуального представлення, може мати кілька різних представлень для однієї «моделі».

Представлення відповідає за отримання необхідних даних з моделі і відправляє їх користувачеві. Представленняне не оброблює введені данні користувачем.

Контролер забезпечує «зв'язок» між користувачем і системою. Контролює і направляє дані від користувача до системи і навпаки. Використовує модель і представлення для реалізації необхідного дії [28].

Схема, яка демонструє принцип роботи MVC, зображена на рисунку 2.1.

Рисунок 2.1 – Схема MVC

Також, для підтримки принципу єдиної відповідальності і можливості розбити систему на модулі, використаний архітектурний підхід Dependency Injection.

Dependency Injection – процес надання зовнішньої залежності програмного компоненту. Є специфічною формою «інверсії управління» (англ. Inversion of control, IoC), коли вона застосовується до управління залежностями

[29]. У повній відповідності з принципом єдиною відповідальності, об'єкт віддає піклування про побудову необхідних йому залежностей зовнішньому, спеціально призначеному для цього спільного механізму. В данному випадку, класу – GameRoot, який ініціалізує всі компоненти необхідними залежностями і даними.

Використовуючи впровадження залежності, об'єкт просто надає властивість, яка в змозі зберігати посилання на потрібний тип сервісу, і коли об'єкт створюється, посилання на реалізацію потрібного типу сервісу автоматично вставляється в це властивість (поле).

На рисунку 2.2 зазначена різниця між традиційним підходом і Dependency Injection.

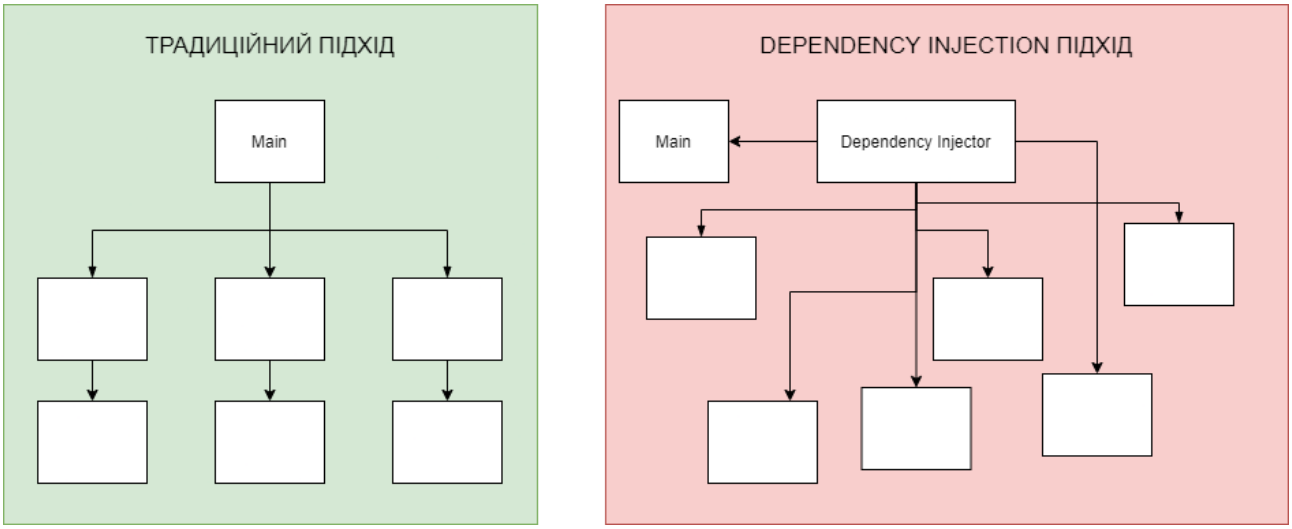


Рисунок 2.2 – Різниця між традиційним підходом і Dependency Injection.

Діаграма класів (рисунок 2.3) була розроблена за домогою мови UML [30]. Відношення між класами відображають принцип роботи Dependency Injection. Всі залежності між класами встановлюються через клас контейнер GameRoot. В GameRoot викликаються методи Init кожного класу, параметрами якого є необхідні класи залежності для ініціалізуємого класу. Таким чином всі залежності стають явними, що надає можливість відслідкувати зв’язки між компонентами і поділити на модулі.

Кожен клас є або моделю, або контроллером, або представленням. Таким чином реалізується архітектурний підхід MVC.

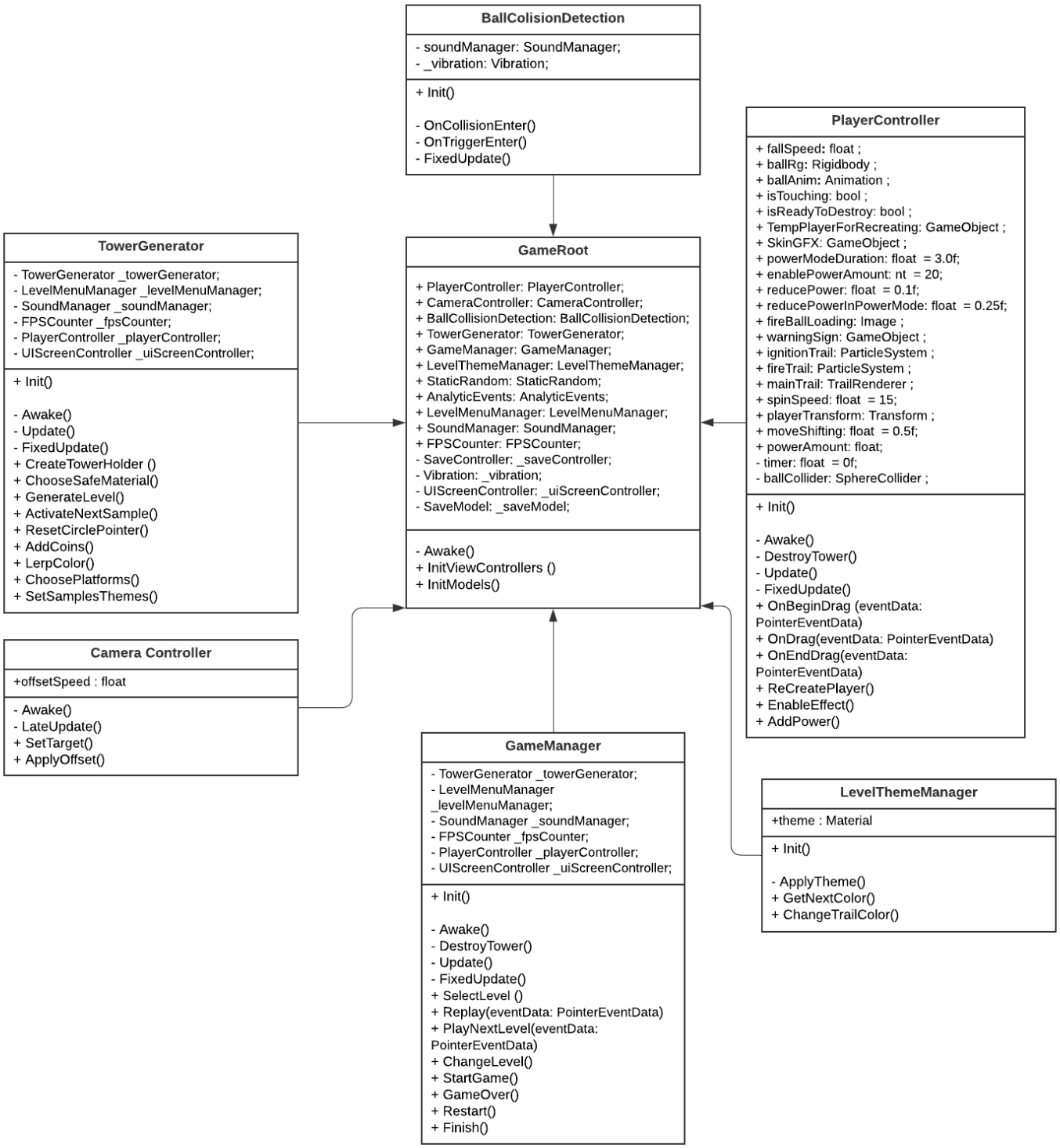


Рисунок 2.3 – Діграма класів програми

2.3 Вибір фреймворку для розробки

У даному розділі проводиться аналіз існуючих ігрових рушіїв з метою вибору найкращого варіанту для розробки мобільного додатку. Серед найпопулярніших ігрових рушіїв для розробки мобільних ігор виявилися такі три: Unity3d, Unreal Engine 4, GameMaker Studio.

Ігровий рушій (англ. Game engine) – це основний набір програмних компонентів і візуальних інструментів, який дозволяє створювати і запускати інтерактивні додатки з графічним забезпеченням, яке обробляються в реальному часі [31].

2.3.1 Unity3d

Unity - міжплатформна середа розробки комп'ютерних ігор, розроблена американською компанією Unity Technologies. Unity дозволяє створювати додатки, що працюють на більш ніж 25 різних платформах, що включають персональні комп'ютери, ігрові консолі, мобільні пристрої, інтернет-додатки та інші [32].

Редактор Unity має простий Drag & Drop інтерфейс, який легко налаштовувати, що складається з різних вікон, завдяки чому можна проводити налагодження гри прямо в редакторі. Рушій використовує для написання скриптів мову C#. Раніше підтримувалися також Boo (діалект Python, підтримку прибрали в 5-й версії) і модифікація JavaScript, відома як UnityScript (підтримка припинена в версії 2017.1). Розрахунки фізики виробляє фізичний движок PhysX від NVIDIA. Графічний API - DirectX (на даний момент DX 11, підтримується DX 12).

Проект в Unity ділиться на сцени (рівні) – окремі файли, що містять свої ігрові світи зі своїм набором об'єктів, сценаріїв, і налаштувань. Сцени можуть містити в собі як, власне, об'єкти (моделі), так і порожні ігрові об'єкти – об'єкти,

які не мають моделі «пустишки». Об'єкти, в свою чергу містять набори компонентів, з якими і взаємодіють скрипти. Також у об'єктів є назва (в Unity допускається наявність двох і більше об'єктів з однаковими назвами), може бути тег (мітка) і шар, на якому він повинен відображатися. Так, у будь-якого об'єкта на сцені обов'язково присутній компонент Transform – він зберігає в собі координати місця розташування, повороту і розмірів об'єкта по всіх трьох осях. У об'єктів з видимої геометрією також за замовчуванням присутній компонент Mesh Renderer, що робить модель об'єкта видимою. До об'єктів можна застосовувати колізії (в Unity т. Н. Колайдери - collider), яких існує декілька типів [33].

Також Unity підтримує фізику твердих тіл і тканини, а також фізику типу Ragdoll (тряпічная лялька). У редакторі є система успадкування об'єктів; дочірні об'єкти будуть повторювати всі зміни позиції, повороту і масштабу батьківського об'єкта. Скрипти в редакторі прикріплюються до об'єктів у вигляді окремих компонентів.

Unity 3D підтримує систему Level Of Detail (скор. LOD), суть якої полягає в тому, що на далекій відстані від гравця високодеталізовані моделі замінюються на менш деталізовані, і навпаки, а також систему Occlusion culling, суть якої в тому, що у об'єктів, що не потрапляють в поле зору камери не візуалізується геометрія і колізія, що знижує навантаження на центральний процесор і дозволяє оптимізувати проект. При компіляції проекту створюється виконуваний (.exe) файл гри (для Windows), а в окремій папці – дані гри (включаючи всі ігрові рівні і спільні бібліотеки) [31].

Переваги:

- безкоштовна версія;
- наявність потужного і інтуїтивно зрозумілого редактора;
- візуалізація елементів редактора в реальному часі;
- кросплатформенна підтримка;
- модульна система компонентів.

Недоліком є обмеження візуального редактора при роботі з багатокомпонентними схемами;

На рисунку 2.4. сображено графічний інтерфейс Unity3d.

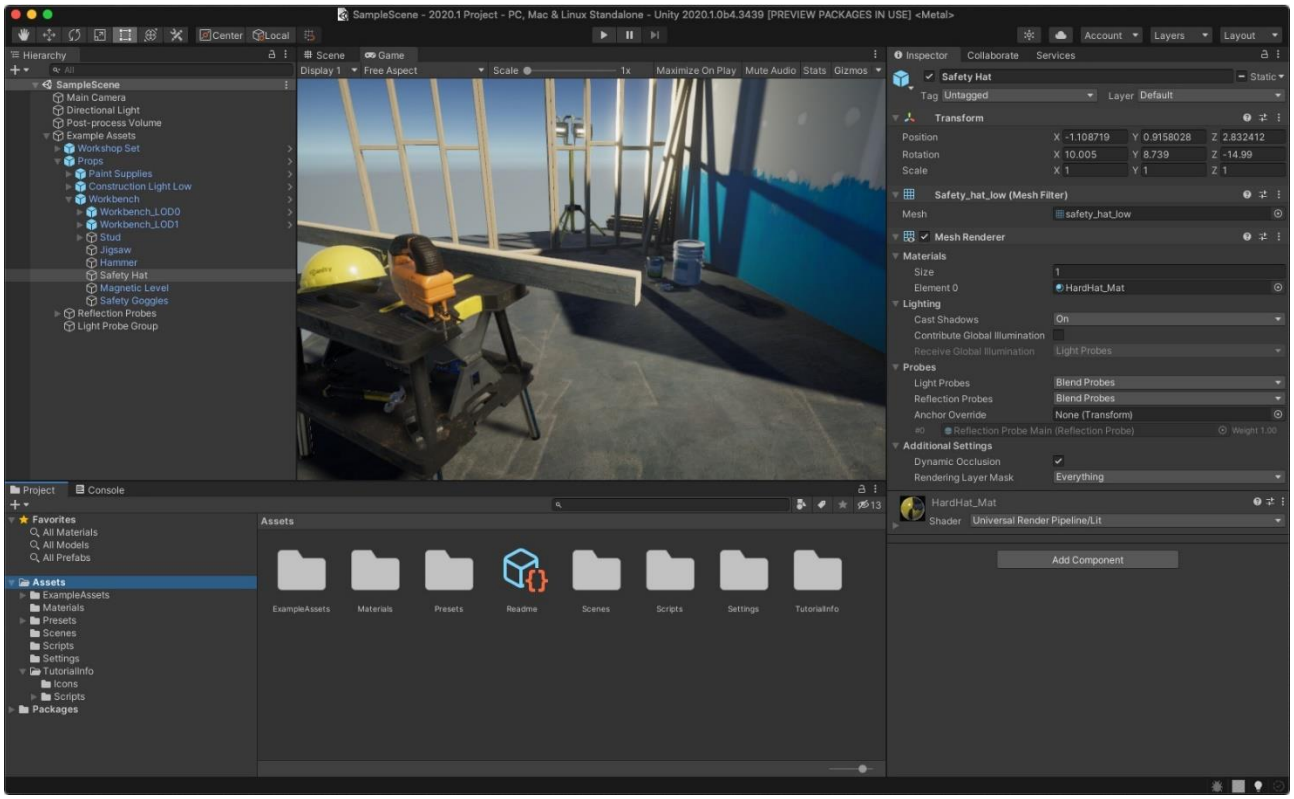


Рисунок 2.4 – Графічний інтерфейс Unity3d.

2.3.2 Unreal Engine 4

Unreal Engine – це сучасний ігровий рушій на базі мови C ++, розробкою і підтримкою якого займається компанія Epic Games. Можливості UE дозволяють створювати і редагувати елементи 3D анімації, спецефекти в кінофільмах і іграх, а також розробляти різні навчальні програми. Код програми на цьому движку працює на більшості сучасних платформ і операційних систем (Android, iOS, Linux, Mac OS, Microsoft Windows, PlayStation 4, PSP, Xbox One,

PS Vita). Таким чином, однією з особливостей рушія Unreal Engine є його універсальність [34].

До складу технології Unreal входить: графічний рушій, рушій забезпечення фізики, штучний інтелект, управління мережевою і файлової системами, а також потужний і багатофункціональний вбудований редактор UnrealEd. UnrealEd (UEd, Unreal Editor) – це інструмент редагування рівнів і ресурсів в реальному часі, на основі принципу моделювання твердих тіл – конструктивної блокової стереометрії (Constructive SolidGeometry, CSG). З останнього випливає те що:

– користувач за допомогою редактора може здійснювати маніпуляції з властивостями різних 3D-об'єктів, змінювати їх, програмуючи скріптові сценарії;

– всі зміни, що вносяться користувачем, такі як: розташування об'єктів, розподіл ефектів, поновлення текстур – видно через камеру редактора, що дозволяє користувачеві перевіряти можливі помилки в будівлях;

– редактор також дозволяє маніпулювати об'єктами, створеними за допомогою графічних пакетів, таких як 3DMax.

До іншої, важливою особливості Unreal варто віднести вбудований редактор BluePrint: це інтуїтивна система для створення логіки 3D-оточення. Простіше кажучи, користувач створює деякі типи об'єктів, які він згодом помістить на «сцену» свого проекту. Приклади роботи скриптів BluePrint в ігровому додатку: призначення гравців з відмінними рисами, установка правил і проведення гри, призначення кнопок для управління персонажем і т.д. При цьому, за допомогою «креслень» логіки рівня або гри користувач може створити, не написавши жодного рядка коду, що, безумовно, полегшить процес написання логіки тим людям, хто не сильний в програмуванні.

Останнє покоління рушія Unreal Engine містить в собі передові технології в області 3D-графіки, серед яких: система розрахунку освітлення Unreal Lightmass, система пост-обробки, реалістична симуляція фізики, створення

великих карт ландшафтів, інструменти анімації персонажів, створення кат-сцен і багато іншого. Простіше кажучи: на стороні рушія Unreal Engine 4, завдяки технологіям Epic Games, буде якість графіки, проте користувачам варто обзавестися неслабим «залізом» на борту комп'ютера, щоб побачити всю красу власними очима.

Переваги:

- безкоштовна версія;
- наявність потужного і інтуїтивно зрозумілого редактора;
- візуалізація елементів редактора в реальному часі;
- blueprints (креслення) зручні для реалізації базової логікі;
- інтеграція з C ++;
- відмінна якість графікі;
- універсальність коду для різних платформ;

Недоліки:

- брак документації за функціями рушія на мові C ++;
- можлива плутанина в «кресленнях» ;
- повільність мобільної розробки;
- вимога сучасного "заліза" ПК;

На рисунку 2.5 зображено графічний інтерфейс Unreal Engine 4.

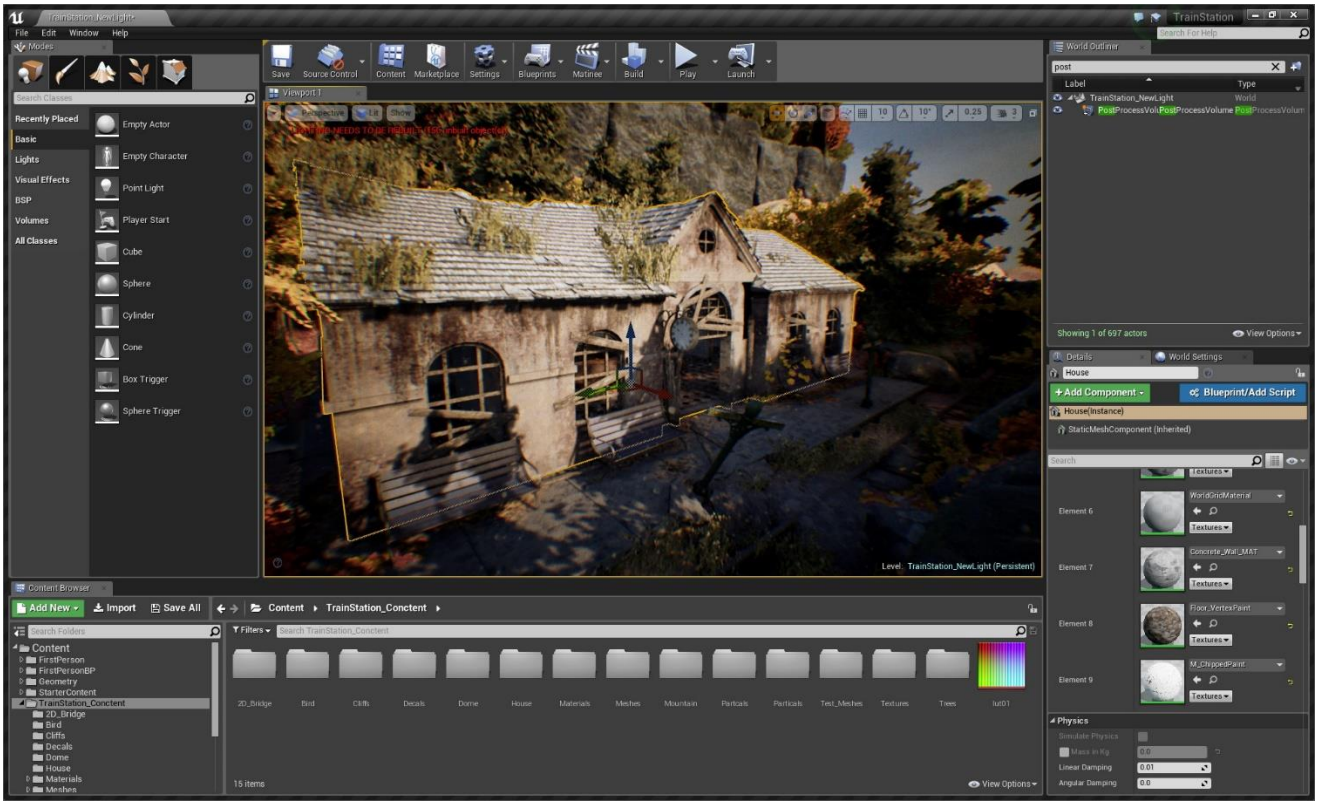


Рисунок 2.5 – Графічний інтерфейс Unreal Engine 4.

2.3.3 Game Maker Studio

GameMaker: Studio – один з найпопулярніших ігрових движків, що дозволяє розробляти додатки під безліч платформ.

GameMaker: Studio дозволяє писати розширення під безліч платформ на відповідних їм мовами. Підтримуються наступні типи файлів розширень: gml, dll-бібліотеки на Windows, Windows Phone, Xbox One, js-скрипти для HTML5, so-бібліотеки на Linux і Tizen, dylib-бібліотеки на Mac, prx на PS4, suprs на PSVita, а також спеціальні placeholder для iOS і Android. Є можливість створити проксі-розширення для проектів на багатьох платформах, що дозволяє використовувати однакові назви функцій в коді, але звертатися до бібліотек відповідної платформи.

Є функції для роботи з кодуваннями Base64, JSON, MD5, SHA-1, можливості розпакування ZIP-архівів, читання і записи .ini, текстових і

двійкових файлів, управління каталогами. Є можливість взаємодіяти з мережею: колективна гра по UDP, TCP, Bluetooth, відправка http-запитів, завантаження будь-яких файлів, взаємодія зі Steam API і Facebook.

З версії 1.1.1086 додана підтримка шейдерів – ефективного інструменту управління відображенням, додавання графічних ефектів і перетворень. GameMaker: Studio підтримує вершинні і фрагменти шейдери на мовах GLSL ES, HLSL9, HLSL11 і GLSL.

З пристроїв, крім миші і клавіатури, присутні функції для взаємодії з джойстиками і геймпадом, для обробки натискань і нахилу на смартфонах.

GameMaker: Studio містить безліч математичних функцій для роботи зі скалярними і векторними величинами, включаючи тригонометричні обчислення, знаходження ступенів, логарифмів, інтерполяцій, нормалей векторів, скалярних творів. Є вбудований фізичний движок Box2D, набір функцій для роботи з ним, демонстраційні проекти Angry Cats і Angry Cats Space.

На відміну від GameMaker, Studio використовує компілятор, а не інтерпретатор коду, що прибрало можливість виконання чистого коду "на льоту", але істотно збільшило продуктивність ігор. Також є додатковий модуль YoYoCompiler, що транслює GML-код в C ++ і оптимізує його. Проекти, зібрані з використанням цього компонента, демонструють хорошу продуктивність з сотнями ігрових елементів на рівні. Однак цей модуль доступний не для всіх платформ.

Є підтримка багатьох сервісів монетизації (таких як AdMob, Google Analytics, Google Play Licensing) з коду. Також є підтримка систем управління версіями – є приклади інтеграції з SVN, GIT, Mercurial [35].

Переваги:

– кроссплатформенность, підтримувані платформи: Windows, Mac OS X, Ubuntu, Android, iOS, Windows Phone, Tizen, Xbox, PlayStation;

– підтримка бібліотек і розширень, в тому числі на різних мовах;

– гнучка цінова категорія, Standard версія Game Maker: Studio абсолютно безкоштовна;

– власна спрощена мова програмування Game Maker Language (GML);

– інтеграція з декількома системами управління версіями;

– інтеграція з Steam, GooglePlay, AppStore.

Недоліком є незручна робота з 3D.

На рисунку 2.6 зображено графічний інтерфейс Game Maker: Studio

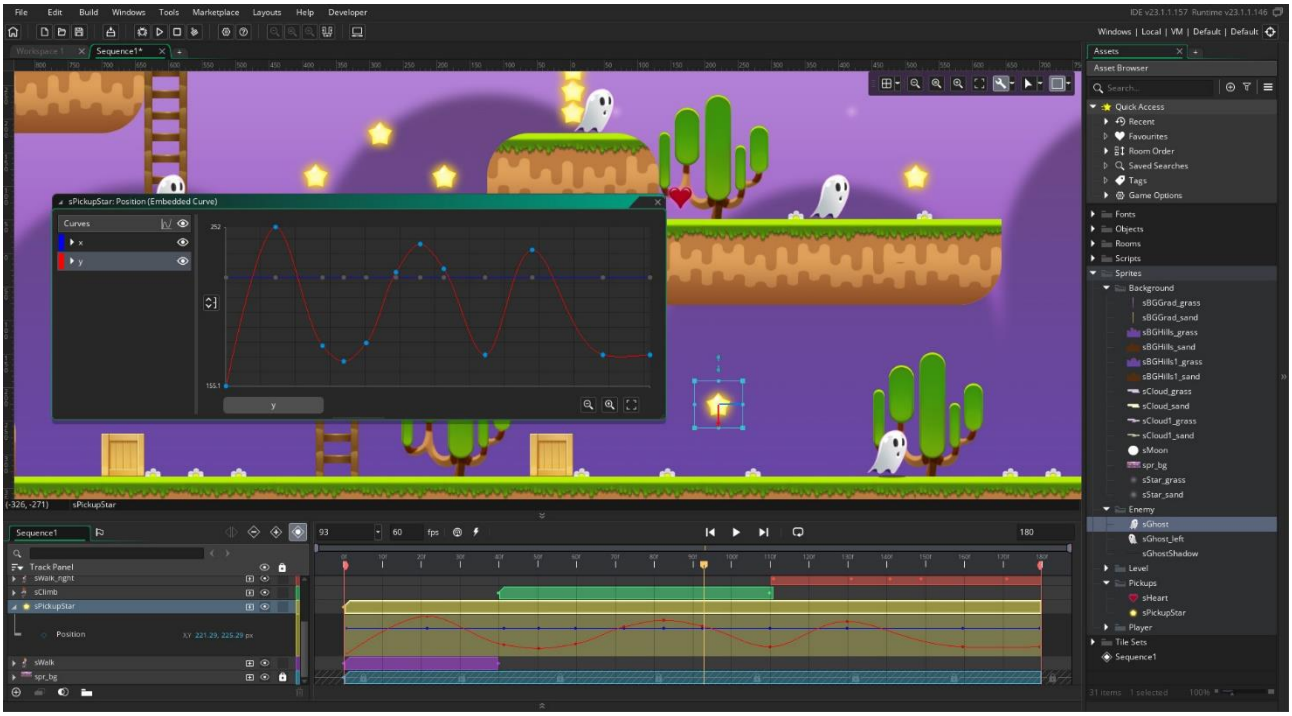


Рисунок 2.6. – Графічний інтерфейс Game Maker: Studio

Таблиця 2.1 – порівняння Unity3d, Unreal Engine 4 та Game Maker: Studio

Критерій	Unity3d	Unreal Engine 4	Game Maker: Studio
Якісна документація	5	4	4
Функціонал без додаткових розширень	4	5	5
Адаптивність	5	3	4
Функціонал	5	4	3
Швидкість роботи	5	3	5
Масштабованість	5	4	5
Просте вивчення	5	3	4

Продовження таблиці 2.1

Критерій	Unity3d	Unreal Engine 4	Game Maker: Studio
Кросплатформеність	5	4	5
Підтримка 3D	5	5	1
Безкоштовне використання	5	4	5

Проведено аналіз, який показав, що найкращим рішенням буде використання рушія Unity3d, оскільки він досить легкий у вивченні, в наявності безкоштовна версія, якісна технологія 3D візуалізації, якісна документація, а також легка побуда програми для мобільних ОС. Unreal Engine 4 уступає в данному вибірі через громіздкість (велика кількість непотрібного функціонала для виконання мети проекту), а також через складний процес портування гри на мобільні пристрої. Game Maker: Studio майже по всім критеріям підходить для виконання зазначего проекту, але в ньому немає якісної 3D візуалізації, що і призводить до вибору Unity3d, як інструменту для вирішення поставленої задачі.

2.4 Обґрунтування вибору Unity3d в якості ігрового рушія

Оскільки обрано Unity3d як рушій для розробки програмного продукту, система складатиметься зі спеціальних компонентів.

Проект в Unity ділиться на сцени – окремі файли, що містять свої ігрові світи зі своїм набором об'єктів, скриптів, і налаштувань. Сцени можуть містити в собі як, власне, об'єкти (моделі), так і порожні ігрові об'єкти – сутності, які не мають моделі («пустишки») (рисунок 2.7).

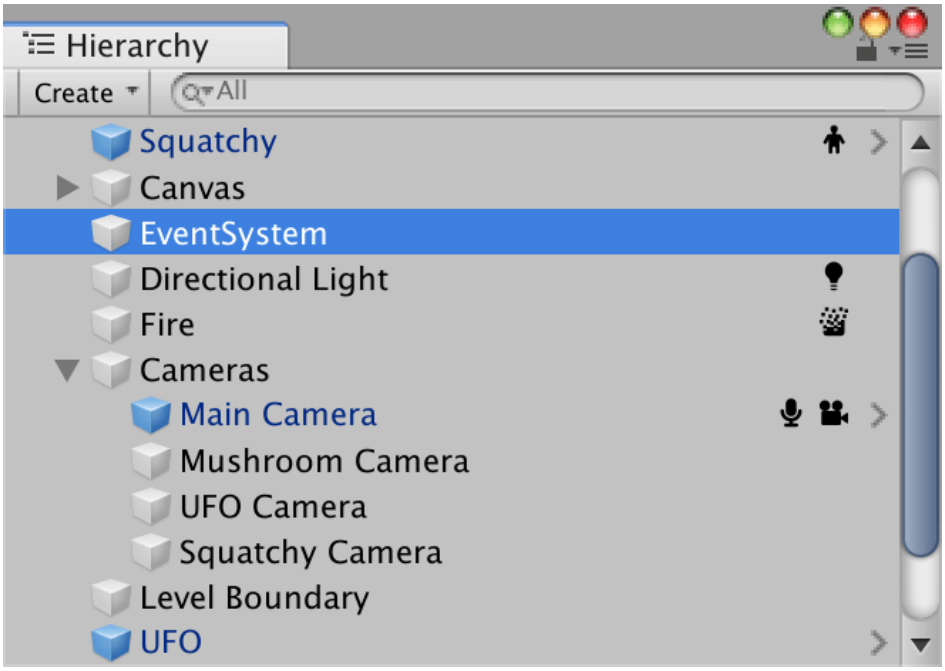


Рисунок 2.7 – Ієрархія об’єктів сцени

Об'єкти, в свою чергу містять набори компонентів, з якими і взаємодіють скрипти. Також у об'єктів є назва (в Unity допускається наявність двох і більше об'єктів з однаковими назвами), тег (мітка) і шар, на якому він повинен відображатися. Таким чином, у будь-якого об'єкта на сцені обов'язково присутній компонент Transform – він зберігає в собі координати місця розташування, повороту і розмірів об'єкта по всіх трьох осях. У об'єктів з видимої геометрією також за замовчуванням присутній компонент Mesh Renderer, що робить модель об'єкта видимою. До об'єктів можна застосовувати колізії (в Unity – collider), яких існує декілька типів: box collider, capsule collider, sphere collider, mesh collider [36]. Система компонентів зображена на рисунку 2.8.

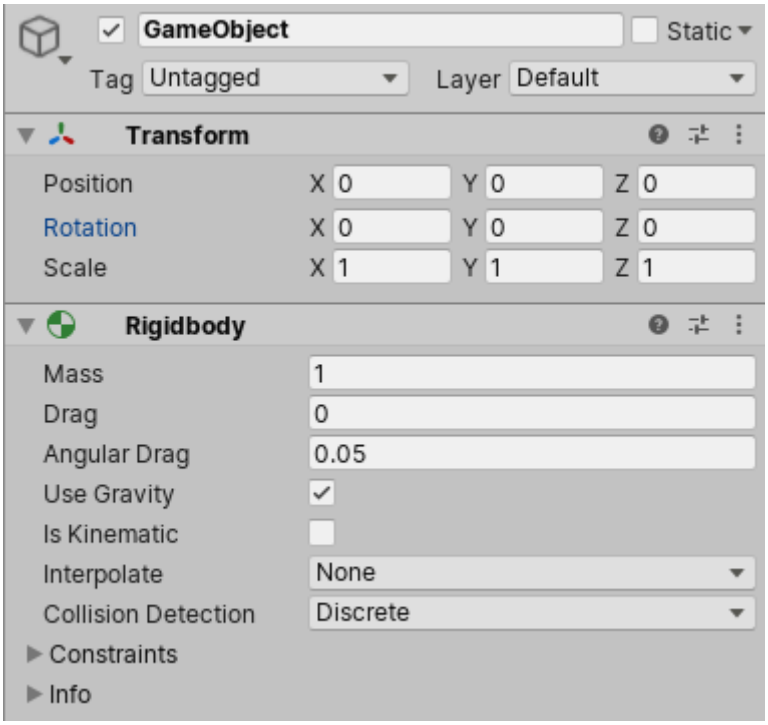


Рисунок 2.8 – Система компонентів об’єкту

У редакторі є система успадкування об'єктів: дочірні об'єкти будуть повторювати всі зміни позиції, повороту і масштабу батьківського об'єкта. Скрипти в редакторі прикріплюються до об'єктів у вигляді окремих компонентів.

При імпорті текстури в Unity можна згенерувати alpha-канал, mip-рівні, normal-map, light-map, карту відображень, проте безпосередньо на модель текстуру прикріпити не можна – буде створено матеріал, який складається з шейдеру, текстури і параметрів, і потім матеріал прикріпиться до моделі. Редактор Unity підтримує написання і редагування шейдерів. Редактор Unity має інструмент для створення анімації, але також анімацію можна створити попередньо в 3D-редакторі та імпортувати разом з моделлю, а потім розділити на файли.

Як правило, ігровий рушій надає безліч функціональних можливостей, які можуть бути задіяні в різних іграх, а саме: моделювання фізичних середовищ, карти нормалей, динамічні тіні і багато іншого. На відміну від багатьох ігрових рушіїв, у Unity є дві основні переваги: наявність візуального середовища розробки та міжплатформенної підтримки. Перший фактор включає не тільки

інструментарій візуального моделювання, а й інтегровану середу, цепочку збірки, які направлено на підвищення продуктивності розробників, зокрема, швидкості етапів створення прототипів і тестування. Під міжплатформеною підтримкою мається на увазі не тільки місця розгортання (установка на персональному комп'ютері, на мобільному пристрої, консолі тощо.), Але і наявність інструментарію розробки (інтегроване середовище може використовуватися під Windows і Mac OS).

Третьою перевагою вважається модульна система компонентів Unity, за допомогою якої відбувається конструювання ігрових об'єктів, коли останні є комбінованими пакетами функціональних елементів. На відміну від механізмів наслідування, об'єкти в Unity створюються за допомогою об'єднання функціональних блоків (в Unity3d – gameobjects), а не поміщення в вузли дерева наслідування. Такий підхід полегшує створення прототипів, що актуально при розробці ігор [33].

2.5 Вибір середовища програмування

2.5.1 Visual Studio Community

Visual Studio включає в себе редактор ісходного коду з підтримкою технології IntelliSense і можливістю найпростішого рефакторінга коду. Вбудований відладчик може працювати як відладчик ісходного коду, так і відладчик машинного. Решта вбудованих інструменів включають в себе редактор форм для спрощення створення графічного інтерфейсу додатку, веб-редактор, дизайнер класів і дизайнер схеми бази даних. Visual Studio дозволяє створювати і підключати сторонні плагіни для розширення функціональності практично на кожному рівні, включаючи додавання підтримки систем контролю версій ісходного коду (як, наприклад, Subversion і Visual SourceSafe), додавання нових наборів інструментів (наприклад, для редагування і візуального проектування коду на предметно-орієнтованих мовах програмування) або

інструментів для інших аспектів процесу розробки програмного забезпечення (наприклад, клієнт Team Explorer для роботи з Team Foundation Server) [37].

2.5.2 Jetbrains Rider

Rider допомагає розробляти додатки .NET, ASP.NET, .NET Core, Xamarin і Unity на Windows, macOS або Linux. Він забезпечує широкі можливості редагування та аналізу коду для мов, що використовуються в .NET-розробці, таких як C #, VB.NET, F #, підтримує синтаксис ASP.NET Razor, JavaScript, TypeScript, XAML, XML, HTML, CSS, SCSS, JSON і SQL.

Інтелектуальний редактор коду надає кілька видів автодоповнення, шаблони для написання типових конструкцій і постфіксні шаблони, посилання на контролери і дії в ASP.NET MVC, дозволяє редагувати код в декількох місцях одночасно і швидко переміщатися по ієрархії успадкування за допомогою іконок на полях. Rider імпортує відсутні простору імен, вставляє парні дужки, підсвічує границі блоків коду. Для перехода до рефакторингу, генерації коду, командам навігації і контекстним дій, потрібно натиснути всього пару клавіш.

Rider надає більше 2200 інспекцій, які допомагають знаходити помилки і проблеми в структурі коду. Для усунення виявлених проблем є понад 1000 автоматичних виправлень. Для глобального відстеження проблем в проектах в Rider передбачений інструмент аналізу помилок по всьому рішенням: він буде шукати помилки в кодової базі і повідомляти про них, навіть якщо проблемний файл не відкритий в редакторі.

Rider тісно взаємодії з рушієм Unity3d. На панелі інструментів розміщені кнопки запуску і зупинки ігрового процесу – Play, Pause і Step, які відповідають тим же кнопкам в редакторі Unity і виконують ті ж функції. Невелика іконка Unity на панелі станів показує стан редактора Unity: підключений, оновлює стан або знаходиться в режимі Play. Будь-які зміни, внесені в Rider в режимі Edit, будуть моментально передані в редактор Unity.

Rider автоматично декомпілює зовнішні бібліотеки, дозволяючи виконувати налагодження декомпільованого коду, входити у вкладені функції, встановлювати точки зупинки, переглядати і встановлювати локальні і інші змінні.

Rider оснащений безліччю спеціально призначених для Unity інспекцій та відповідних швидких виправлень. Він здатний розпізнавати шаблони низькопродуктивного коду і може рекомендувати для них автоматичні виправлення, наприклад використовувати перевантаження, кешування значень, або альтернативний API [38].

Таблиця 2.2 – Порівняння IDE

Критерій	Rider	Visual Studio
Можливість роботи на різних платформах	+	+
Безкоштовна ліцензія	-	+
Якісна документація	+	+
Інтеграція з консоллю Unity3d	+	-
Інтеграція з gameobjects Unity3d	+	-
Система контролю версій	+	+
Приємний дизайн	+	+
Оптимізованість	+	-
Декомпіляція зовнішніх бібліотек	+	-

Підбиваючи підсумки, для комфортної роботи та швидкої розробки проекту, Rider є найбільш оптимальною середою розробки, оскільки він краще взаємодії з редактором Unity3d, а також надає більше інструментів для рефакторингу коду.

2.6 Висновки до розділу 2

В даному розділі вирішено задачу з проектування програмного забезпечення. Завдяки проектуванню, визначено які архітектурні підходи та фреймворки використувати в розробці.

Виконано завдання визначити функціональні вимоги до програмного продукту. Перечислені вимоги є необхідними для того, щоб продукт мав привабливість, користувачеві було приємно і зручно грати, а також щоб тривалість ігрового сеансу була найбільшою.

Виконано завдання проаналізувати архітектурні підходи, які допоможуть найоптимальнішим способом реалізувати зазначені функціональні вимоги. Розроблено схему (рис.2.1), яка демонструє суть патерну MVC (Model View Controller), який розділяє програму на 2 великі частини: дані та представлення. Також розроблена схема, для більш наявного розуміння підходу Dependency Injection (рис 2.2), метою якого є – зробити явними всі залежності між програмними класами та об’єктами.

Виконано завдання розробити діаграму класів (рис.2.3), яка відображає використання двух архітектурних підходів. Згідно неї, були розроблені програмні класи.

Виконано завдання проаналізовати ігрові рушії: Unity3d, Unreal Engine 4, Game Maker: Studio. Побудована таблиця порівняння ігрових рушіїв (табл. 2.1), яка дозволяє наявно побачати переваги та недоліки кожного з метою вибору оптимального варіанту. Обрано рушій Unity3d для написання ігрової програми по таким причинам: легкий у вивченні, в наявності безкоштовна версія, якісна технологія 3D візуалізації, якісна докуменація, а також легка побуда програми для мобільних ОС.

Виконано завдання проаналізувати існуючі IDE (Integrated Development Environment). Побудована таблиця порівняння (табл. 2.2), яка демонструє переваги та недоліки кожного. Після чого, як середа програмування, на основі аналізу Visual Studio Community та Jetbrains Rider, обраний Jetbrains Rider, по таким причинам: можливість інтеграції з консоллю Unity3d, інтеграція з gameobjects Unity3d, оптимізованість, можливість декомпіляції зовнішніх бібліотек.

В наступному розділі необхідно описати конструювання програмного забезпечення: з чого складається ігровий процес, яка структура проекту, використані алгоритми та шаблони проектування.

3 КОНСТРУЮВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

В даному розділі описуюється програмна реалізація ігрової програми. Розділ присвячений наступним питанням: розробка ігрового процессу, опис структури проекту, опис реалізація алгоритму генерації башні, опис реалізації графічного інтерфейсу.

3.1 Розробка ігрового процесу

Ігровий процес (англ. Gameplay) — термін, яким називають особливості взаємодії людини із грою. Ці особливості створюються за допомогою правил, завдань та способів їх вирішення, які пропонує гра. Поряд з фабулою та технологією ігровий процес є однією з трьох основних складових будь-якої відеогри [39].

Основою ігрового процесу будь-якої відеогри є повторюваний ланцюжок дія-реакція-зворотній зв'язок.

3.1.1 Дія

Дія (англ. Action) — інформація, яку гравець надає грі з ігрових контролерів, як геймпад, клавіатура чи тачскрін (у данному випадку) [40].

Гравець керує головним героєм (шаром), використовуючи тачскрін. Свайп вліво повертає башню вліво, свайп вправо повертає башню вправо, що в свою чергу надає головному герою можливість оминути чорні блоки. Торкання будь-якої області екрану призводить до ускорення головного героя, що в свою чергу надає можливість рухатися вниз вздовж башні, оминаючи чорні блоки. Якщо герой ударяється в чорні блоки, то гра завершується і починається показ результатів.

3.1.2 Реакція

Реакція (англ. Reaction) — відповідь гри на дії гравця, сформована на основі наданої ним інформації, але ще не виражена для нього в доступній формі [40].

Під час свайпів викликається функція повороту башні. Під час тачу, гра змінює швидкість героя. Після того, як був зроблений першик зсув вниз вздовж башні, запускається таймер, який рахує кількість секунд з моменту початку гри. Під час успішного пересування до низу вздовж башні, гра визначає скільки шляху було пройдено до цих пір, відносно всього шляху.

3.1.2 Відгук

Відгук (англ. Feedback) — інформація, отримувана з гри гравцем. Поділяється на два основних види: явний (є очевидною відповіддю на дії гравця) і неявний (служить фоном чи інформує гравця, наприклад, довідка з гри). Зворотний зв'язок має свої різновиди в межах явного і неявного: візуальний (бачене гравцем на екрані), звуковий (чуте через пристрої відтворення звуку), дійовий (подія в самій грі — відповідь на дії гравця, зазвичай поєднання зображення зі звуком), неігрових персонажів (відповідь персонажів, що «населяють» гру, на дії гравця), акумулятивний (виражений певним чином прогрес гри), емоційний (емоції, які викликає гра), здійснення (вираження сенсу окремих завдань і всієї гри, їх завершення), інформативний (відомості, які гравець отримує в контексті грання). Отриманий зворотний зв'язок спонукає до нових дій і так до завершення гри [40].

Зверху на екрані відображається час, який почав відраховуватися з моменту першого дотику до екрана гравцем. А також графічна шкала, яка відображає довжину пройденного шляху відносного всього шляху. Після розбиття кожного блоку, зараховуються бали до головного рахунку, який

відображається після проходження рівня або після програшу. Під час пересування головного героя телефон легко вібрує, що надає користувачу приємне відчуття прогресу. Якщо героє стикається з чорними блоками, то програється анімація розбиття, також відображається меню програшу: поточний рівень, рахунок, кнопка «Replay», кнопка налаштувань, а також стрілки вліво та вправо, які надають можливість переключатися між рівнями.

3.2 Структура проекту

Проект скаладається з багатьох частин: скрипти (які і реалізують взаємодію між об’єктами на ігровому рівні), графічні моделі, ефекти (партикли), налаштування сцени (світло, скайбокс), графічний інтерфейс (через який користувач взаємодіє з грою). Завдяки використанню архітектурного патерну MVC, вдалося створити систему, в якій бізнес модель, тобто логіка програми, не залежить від візуальних компонентів. Це говорить про те, що можна підібрати будь-які графічні моделі та змінювати користувацький інтерфейс без редагування логіки програми. Далі в таблиці 3.1 приведено опис скриптів, завдяки яким і запрограмований ігровий процес «Дія-Реакція-Відгук».

Таблиця 3.1 – Опис файлів

Назва	Опис
1	2
GameRoot.cs	Містить в собі всі посилання та екземпляри задіяних класів. Грає роль контейнеру в архітектурному підході Dependency Injection.
PlayerController.cs	Реалізує логіку переміщення головного героя, прискорення, оновлення та включення/виключення графічних ефектів. Містить функції зчитування доторкань, свайпів та інших маніпуляцій, які походять від користувача.
LevelThemeManager.cs	Реалізує логіку зміни графічної теми башні при переході на новий рівень, а також під час проходження поточного рівня: зміна кольору, градієнту

Продовження таблиці 3.1

1	2
BallColisionDetection	Реалізує логіку перевірки, чи зіткнувся головний герой з чорним блоком чи ні. Якщо так, то викликається функція, яка змінює стан гри на програш та включає небхнідний графічний інтерфейс.
TowerGenerator.cs	Клас, який відповідає за генерацію башні.
CameraController.cs	Реалізує пересування камери відносно головного героя.
GameManager.	Клас, який є контроллером всього ігрового процесу в цілому. В цьому класі реалізовані такі функції: вибір рівня, replay, початок гри, запуск наступного рівня, програш, реініціалізація головного героя, рестарт, завершення гри.
UIScreenController.cs	Контроллер всіх ігрових екранів. Завдяки йому інші класи переключають ігровий екран на інший при необхідності.

На рисунку 3.1 зображена структура файлів проекту в цілому.

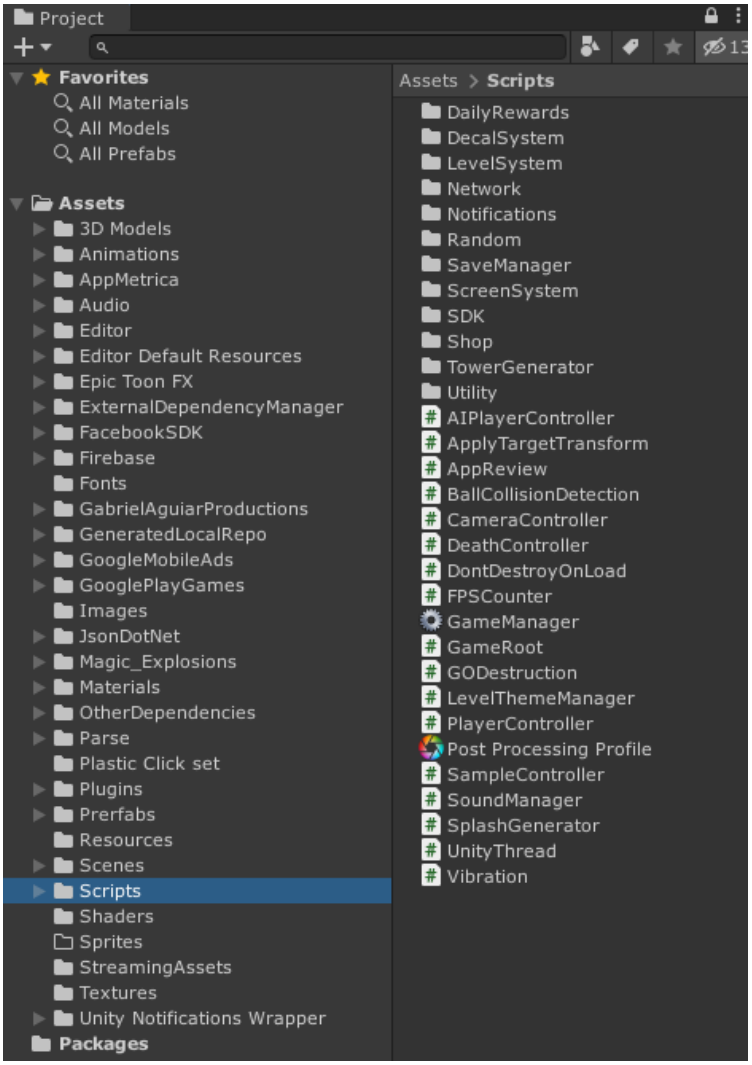


Рисунок 3.1 – Структура файлів проекту

На рисунку 3.2 зображена ієрархія об’єктів на сцені.

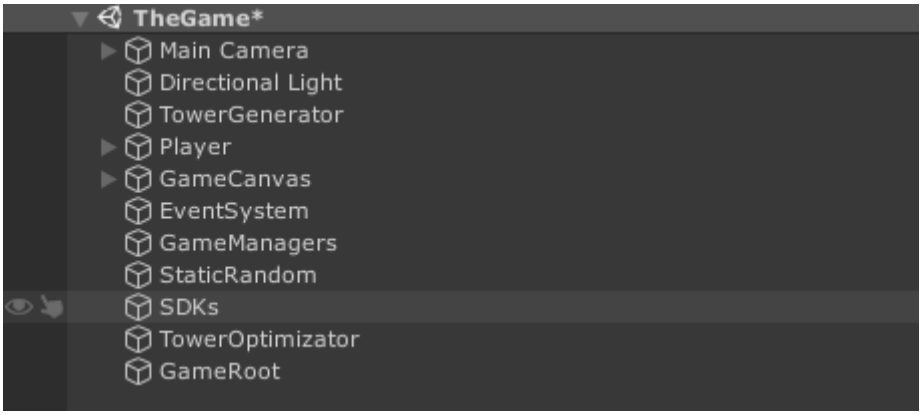


Рисунок 3.2 – Ієрархія об’єктів проекту

3.3 Опис алгоритму генерації башні

Для реалізації алгоритму генерації башні, використаний шаблон проектування – Об’єктний пул (англ. Object pool).

Пул об'єктів призначений для зберігання готових до використання об'єктів [41]. Спочатку об’єкти створюються і додаються до пула. Коли системі потрібен новий об'єкт, він надсилає запит до пула, перевіряючи чи є необхідний об’єкт, якщо так, то пул повертає об’єкт, який раніше уже був створений. Цей об’єкт буде переініціалізований системою для подальшого використання. А після використання повертається назад в пул замість знищення. Таким чином проспукається процес створення нового екземпляру, в результаті чого нова пам’ять не виділяється.

Шаблон застосовується для підвищення продуктивності, якщо:

– об'єкти часто створюються і знищуються;

– в системі існує обмежена кількість об'єктів типу, що зберігається в пулі;

– створення і / або знищення об'єкта є дуже витратними операціями.

Пул об'єктів може працювати як з інтерфейсами, так і з конкретними реалізаціями [40]. Все залежить від архітектури системи і вирішуваних завдань.

На рисунку 3.3 зображена діаграма використання пула.

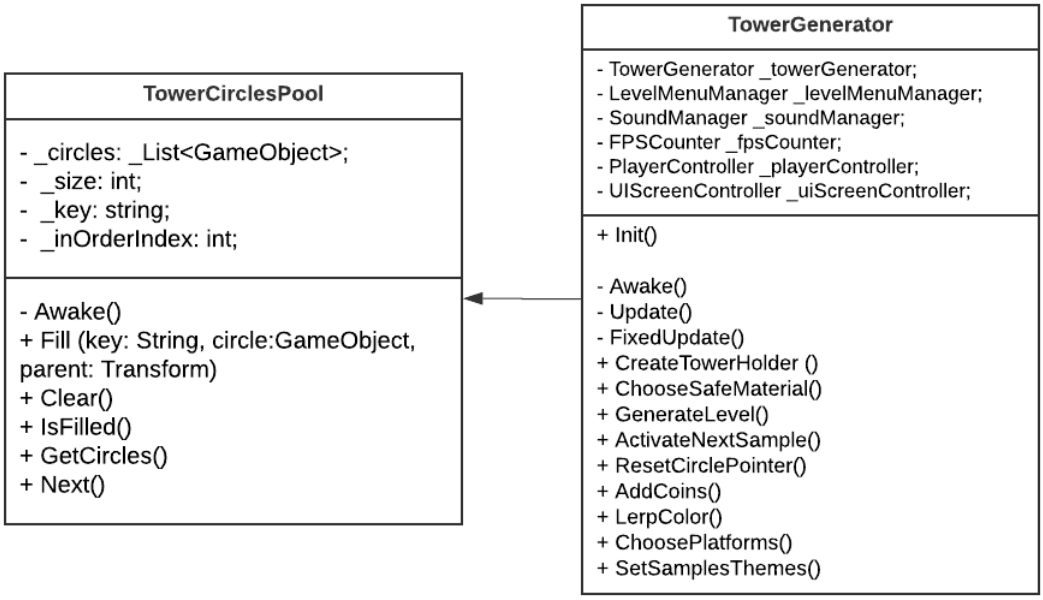


Рисунок 3.3 – Діаграма використання пула

3.4 Опис програмної реалізації графічного інтерфейсу

Для реалізації зручної системи керування ігровими екранами прийняте рішення створити єдиний контроллер, який будет мати посилання на всі ігрові екрани, задіяні в додатку, а також функції, котрі інші класи будуть використовувати для того, щоб включити або виключити необхідний екран.

Самі екрани представлені в Unity3d як GameObjects до яких прикріплений компонент, базовий клас якого є UIScreenBase. Використання функцій

поліморфізму надало можливість створити різні графічні екрани з єдиним базовим функціоналом.

В таблиці 3.2 приведений опис функцій UIScreenController.cs.

Таблиця 3.2 – Опис функцій UiScreenScreenController.cs

Метод	Опис
GetScreen<T>()	Повернення необхідного вікна зі списку всіх ігрових вікон.
T ShowScreen<T>(bool show)	Включення необхідного вікна
void HideAll()	Виключення всіх вікон одразу.

На рисунку 3.4 приведено всі існуючі ігрові екрани додатку.

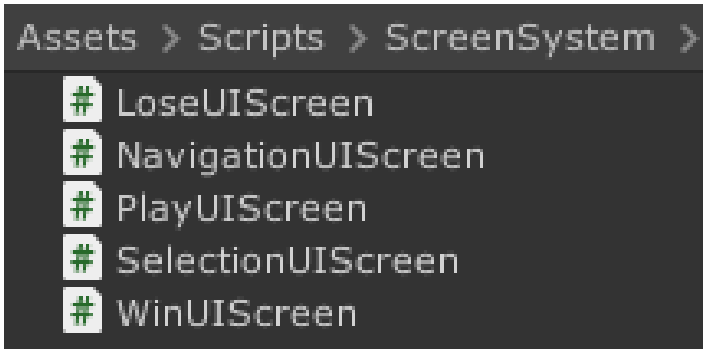


Рисунок 3.4 – Ігрові екрани

На рисунку 3.5 продемонстрована діаграма класів графічного користувацького інтерфейсу. Є базовий клас UIScreenBase, від якого наслідуються такі класи:

- NavigationScreen,
- PlayUIScreen,
- SelectionUIScreen,
- LoseUIScreen,
- WinUIScreen.

Кожен з вище перелічених класів реалізує функції відповідного екрану. В свою чергу UIScreenController містить список даних екранів, маніпулювання

якими відбувається за допомогою функцій: T GetScreen<T>(), T ShowScreen<T>(), void HideAll().

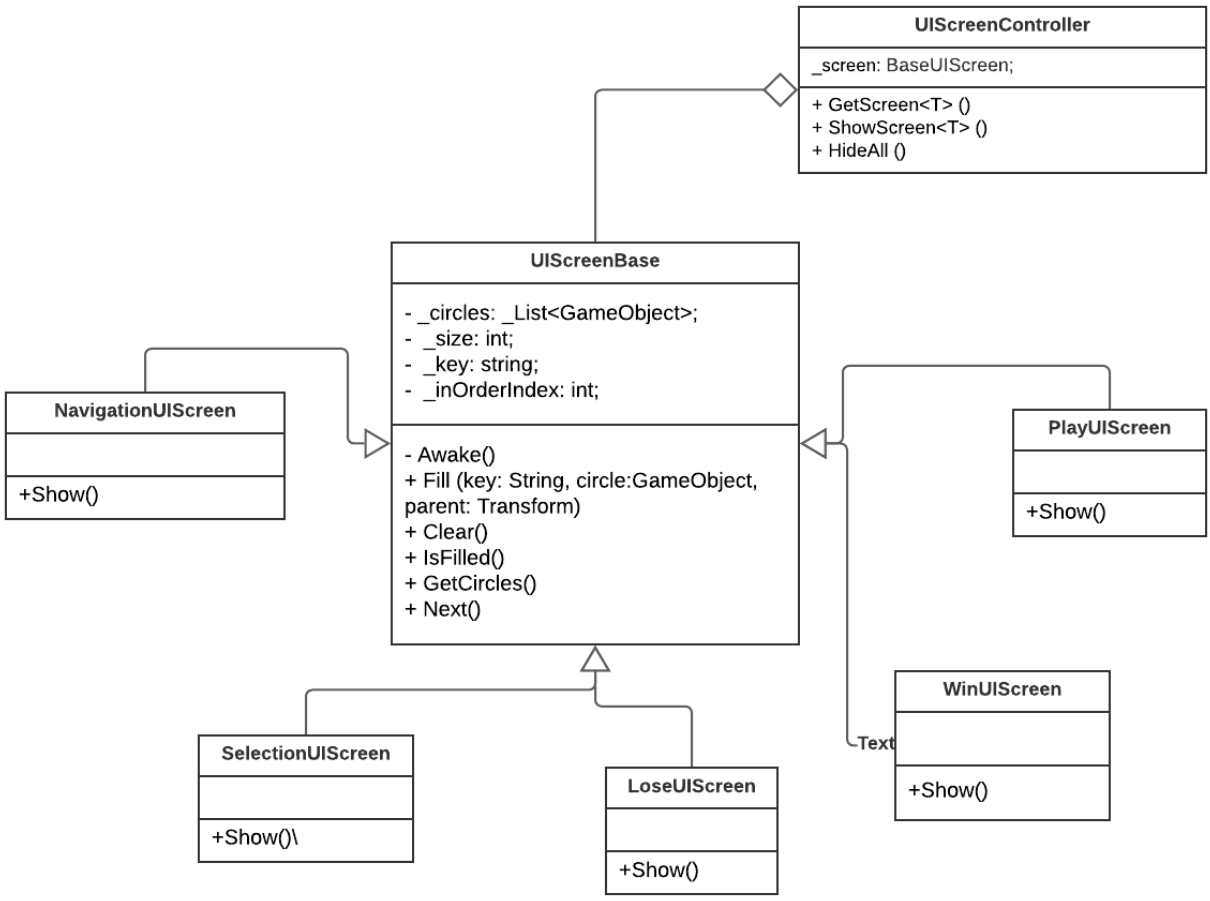


Рисунок 3.5 – Діаграма класів графічного користувацього інтерфейсу

3.5 Висновки до розділу 3

Вирішена задача конструювання програмного забезпечення. Завдяки цьому стало зрозумілим що саме необхідно розробити, а також, на які етапи поділити розробку.

Вирішено завдання дослідженя ігрового процесу. Визначено, які бувають етапи. Згідно цих етапів спроектований ігровий процес розроблюваного додатку.

Вирішено завдання розробки структури проекту. Структура складається з скриптів (які і реалізують взаємодію між об’єктами на ігровому рівні),

графічних моделей, ефектів (партикли), налаштування сцени (світло, скайбокс), графічного інтерфейсу (через який користувач взаємодіє з грою). Це допомогло визначитись з тим, які елементи програми необхідно розробити.

Вирішено завдання розробки алгоритму генерації бескінечної башні. Суть алгоритму полягає в використання пулу об’єктів. Розроблена діаграма (рис.3.3) для демонстрації архітектурної реалізації даного алгоритму. Завдяки його використанню, вдалося оптимізувати додадок та досягти найбільшої можливої швидкодії.

Вирішено завдання з програмної реалізації графічного інтерфейсу. Розроблено діаграму (рис. 3.5), яка демонструє архітектуру програмної реалізації. Завдяки обраному підходу (створення єдиного контроллера, керуючого усіма типами ігрових екранів), вдалося реалізувати зручний перехід між ігровими екранами, включення або виключення яких можливо з будь-якого классу-контроллеру. А також є можливість швидкого редагування існуючих екранів і додавання нових, без необхідності програмних змін в інших частинах програми.

В наступному розділі необхідно визначити системні вимоги, описати функціонал системи, інструкції з запуску системи, роботу системи.

4 ІНСТРУКЦІЇ З ВИКОРИСТАННЯ ПРОГРАМНОГО ПРОДУКТУ

В даному розділі описуються інструкції з використання програмного продукту. Розділ присвячений наступним питанням: системні вимоги, функціонал системи, інструкції з запуску системи, опис роботи системи.

4.1 Системні вимоги

Для роботи програми необхідний мобільний девайс з мінімальною версією ОС Android 6.0.

Для користування додатком необхідне: телефон з мінімальною версію Android 6.0;

4.2 Функціонал системи

Ігрова програма призначена для розваги користувачів.

Програмний продукт надає наступний функціонал:

а) запуск додатка;

б) відображення головного меню з кнопками, які мають такий функціонал:

1) початок ігрового раунда
2) відображення меню налаштувань;
3) вибір ігрового рівня з тих, які користувач вже пройшов, з метою перепроходження.

в) ігровий раунд, в якому користувач керує головним героєм, метою чого є – дійти до фінішу не торкнувшись чорних ігрових блоків; під час ігрового раунда відображаються такі складові ігрового процеса:

1) поточна кількість зароблених монет,
2) шкала проходження рівня,

3) таймер,

4) у випадку програшу, відображаєть вікно з підрахованою кількістю балів, а також кнопкою, при натисканні на яку, ігровий рівень починається заново;

г) вікно налаштувань з таким функціоналом:

1) інвератція управління,

2) включення/виключення звука,

3) включення/виключення вібрації.

4.3 Інструкції з запуску системи

Для запуску програми, необхідно перемістити .apk файл ігрового додатку на мобільний девайс, після чого відкрити цей файл. Після встановлення, іконка програми буде відображатися в головному меню девайса.

4.4 Опис роботи системи

Для використання ігрової програми, необхідно запустити додаток з головного меню мобільного девайсу. На рисунку 4.1 зображено головну сторінку з англійською мовою в якості стандартної.

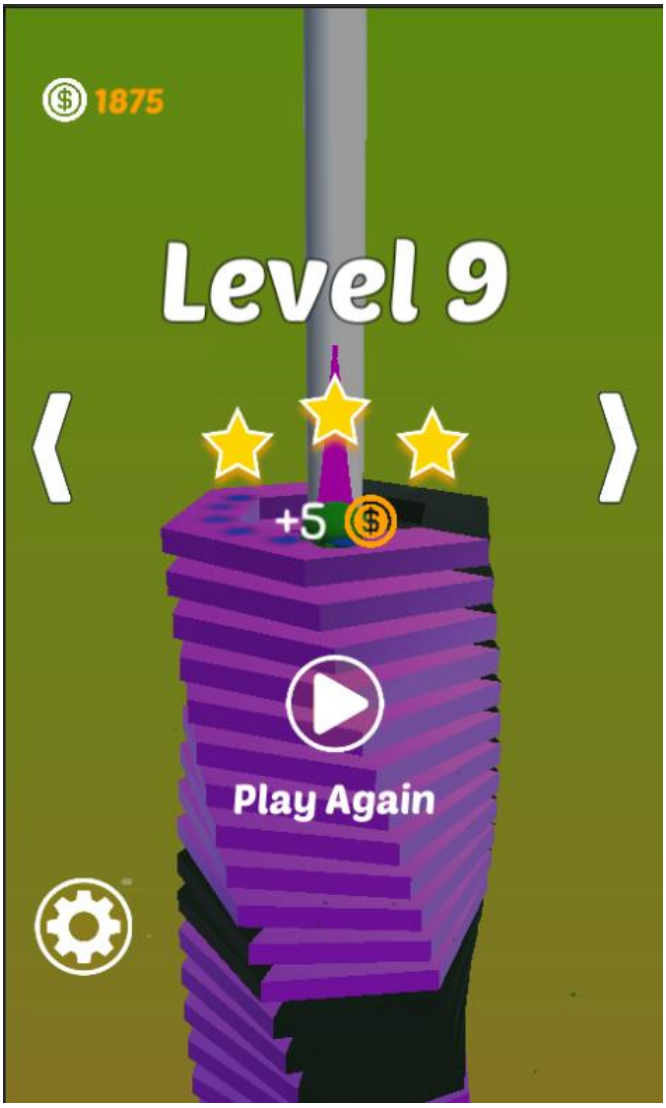


Рисунок 4.1 – Головне меню

Для того, щоб відкрити панель налаштувань, необхідно натиснути на кнопку з іконкою шестерні, після чого відкриється панель, яка зображена на рисунку 4.2.

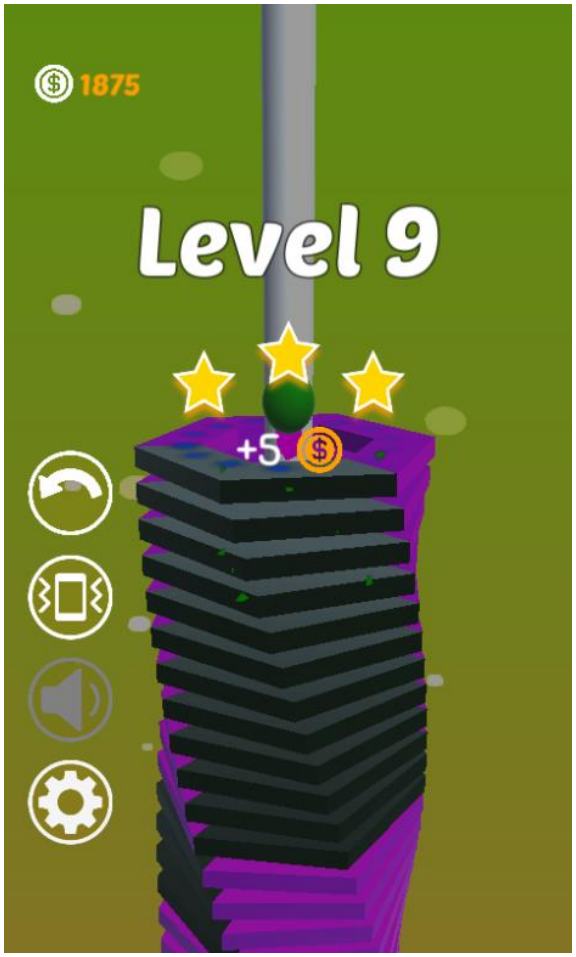


Рисунок 4.2 – Відкрита панель налаштувань

На панелі налаштувань є 3 кнопки, які реалізують такий функціонал:

– інвертація управління головним героєм;
– включення/виключення вібрації;
– включення/виключення звука.

Для вибору необхідної опції, потрібно натиснути на кнопку, після чого її іконка буде змінена на іншу, як показано на рисунку 4.3.

Рисунок 4.3 – Відключена вібрація

Для того, щоб змінити ігровий рівень, потрібно натиснути на стрілку «Вліво» або «Вправо», але це тільки в тому випадку, якщо кількість пройдених ігрових рівнів більша ніж 1.

Для початку гри, необхідно натиснути кнопку «Play», після чого буде відображено ігровий рівень (рис. 4.4).

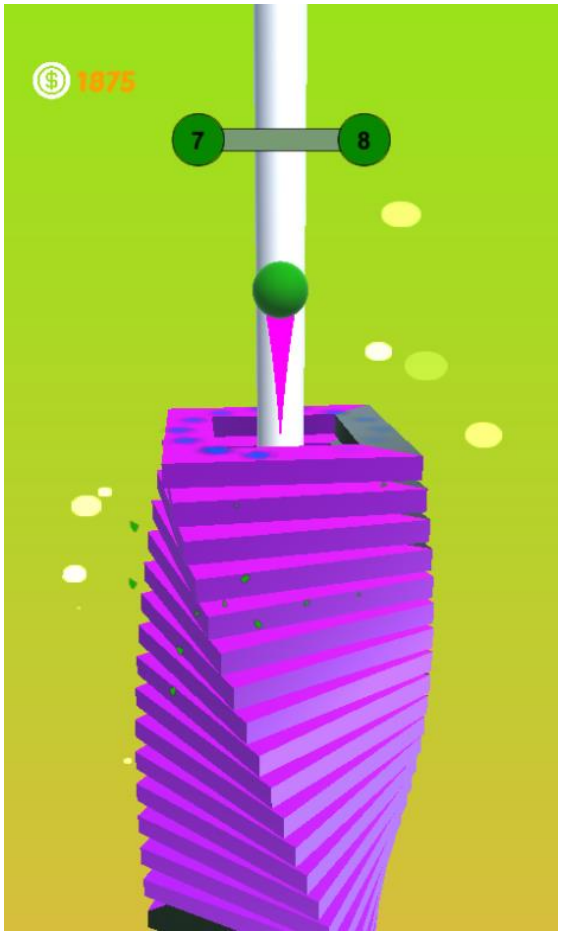


Рисунок 4.4 – Ігровий рівень

Для того, щоб пройти рівень, користувач повинен керувати героєм таким чином, щоб він не зіткнувся з чорними блоками. Після досягненні фінішу, відкриється вікно завершення рівня (рис. 4.5).

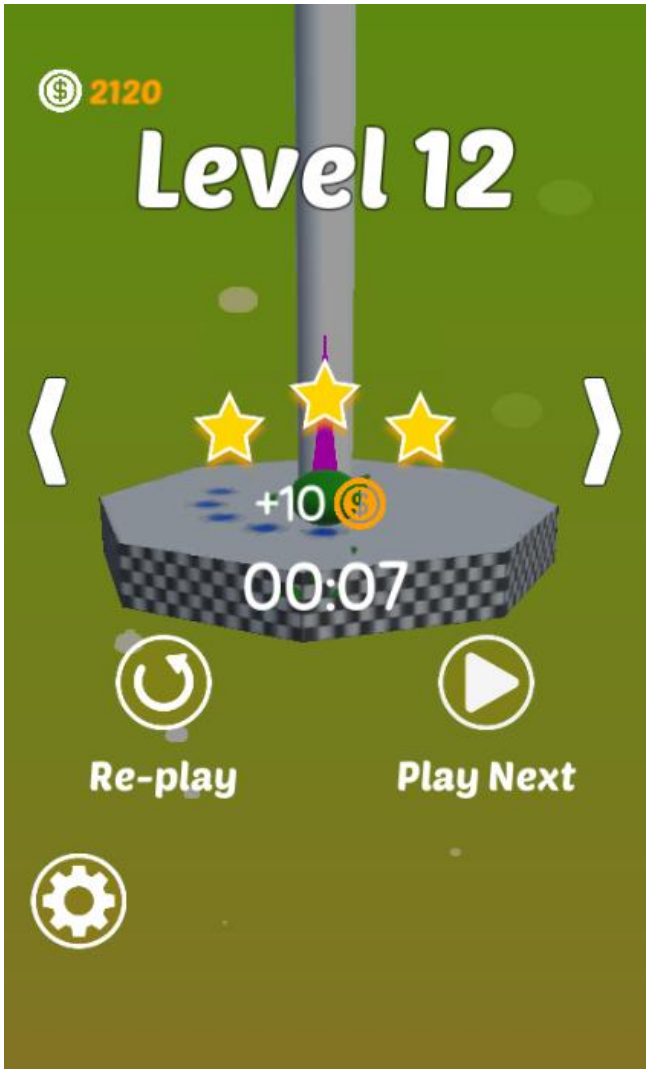


Рисунок 4.5 – Вікно завершення рівня

Вікно завершення рівня має такий функціонал:

– запуск цього ж рівня заново – при натисканні на кнопку з повернутою стрілкою;

– запуск наступного рівня – при натисканні на кнопку з трикутною стрілкою;

– відкриття вікна налаштувань – при натисканні на кнопку з шестернею.

На рисунку 4.5 зображено головного героя в прискорено режимі. Такий режим запускається лише в одному випадку, коли користувач без зупинки рухається до низу необхідну кількість секунд. Після виконання даної умови, швидкість героя збільшується, а також з’являється полум’я позаду нього. Також в такому режимі, герой розбиває чорні блоки, що надає можливість певний час рухатися без програшу.

Зверху відображається поточна кількість монет, яка збільшується, коли герой знаходиться в прискореному режимі.

Зелена шкала – є графічним відображенням прогресу проходження рівня. Зліва відображається номер поточного рівня, а справа номер наступного рівня.

Таймер відображає скільки часу затрачено на проходження даного рівня.

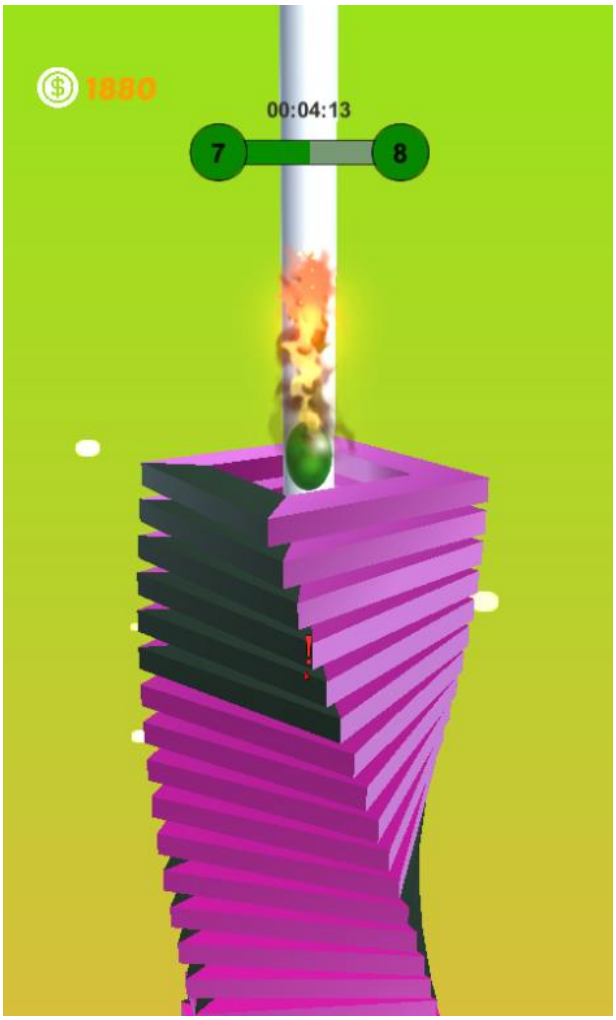


Рисунок 4.5 – Ігровий рівень в прискореному режимі

При зіткненні з чорними блоками відбувається поразка, після чого відображається вікно (рис. 4.6) з таким функціоналом:

– спробувати знову – при натисканні на кнопку зі стрілкою;
– налаштування – при натисканні на кнопку з шестернею;
– зміна рівня – при натисканні на стрілку вліво/вправо.

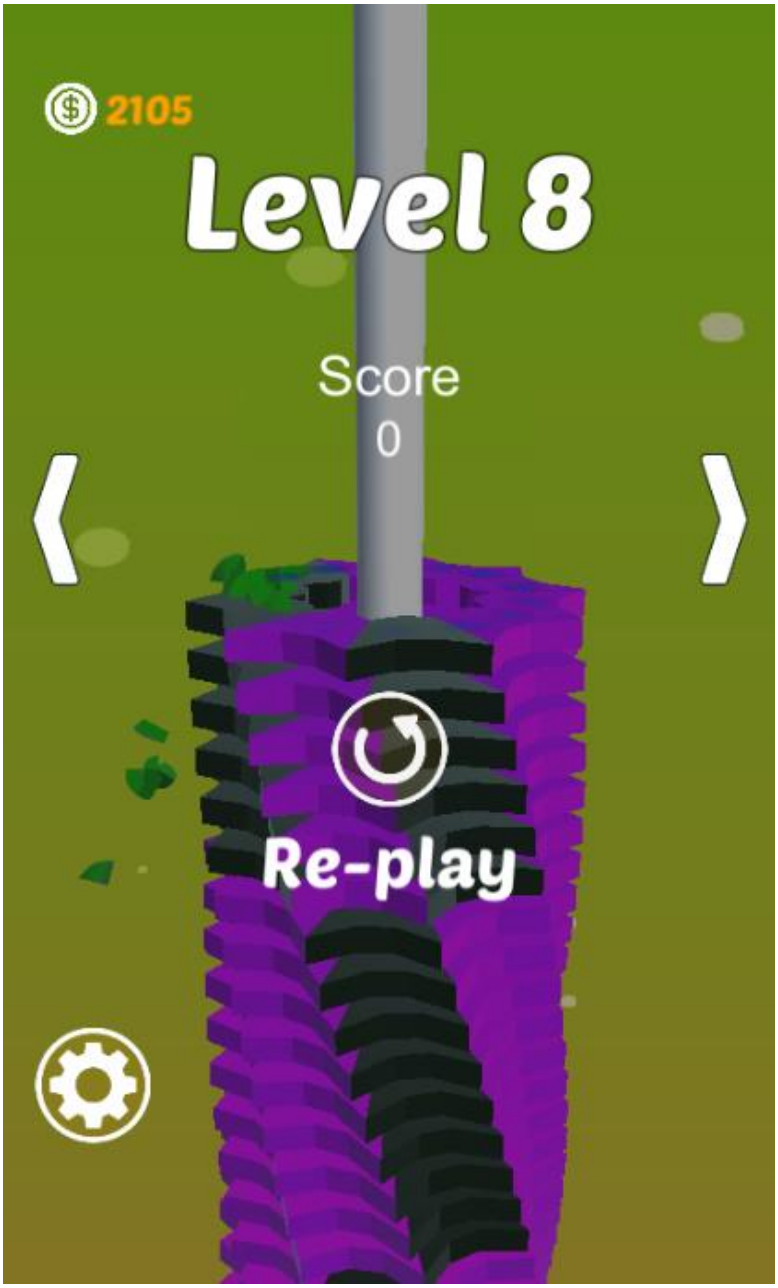


Рисунок 4.6 – Меню поразки

4.5 Висновки до розділу 4

Вирішена задача – розробити інструкції з використання програмного продукта. Це надає кінцевому користувачеві розуміння, яким чином взаємодіяти з ігровою програмою.

У розділі вирішено завдання визначити та описати системні вимоги для запуску та продуктивного функціонування ігрового додатку. Мінімальна версія Android 6.0

У розділі вирішено завдання детально описати функціонал програми, який повністю відповідає функціональним вимогам: пересуванню по ігровій локації зверху в вниз, логіці виграшу та програшу, генерації бескінечних рівнів, навігації по рівням складності, налаштуванню гри.

Описано інструкцію з запуску системи.

Детально описано як взаємодіяти з додатком.

ВИСНОВКИ

В результаті виконання даної дипломної роботи розглянуто та проаналізовані ігрові додатки жанру платформер. Були наведені сучасні аналоги розроблюваної програми «Tower Runner».

Були розглянуті та детально описані сучасні ігрові рушії, на яких створюються ігри подібного жанру.

Перевагами відомого аналогу Game Maker: Studio є:

– інтеграція з декількома системами управління версія;

– інтеграція з Steam, GooglePlay, AppStore.

Перевагами відомого аналогу Unreal Engine 4 є:

– blueprints (креслення) зручні для реалізації базової логікі;

– відмінна якість базової графіки.

Проте вони мають такі недоліки:

– великий об'єм непотрібного фунціоналу та складний процес потрування гри на мобільні пристрої в Unreal Engine 4;

– відсутність якісної 3D візуалізації в Game Maker: Studio.

Після кінцевого аналізу по зазначеним критеріям обрано саме Unity3d, оскільки він досить легкий у вивченні, в наявності безкоштовна версія, якісна технологія 3D візуалізації, якісна докуменація, а також легка побуда програми для мобільних ОС.

Були розглянуті сучасні середовища програмування (IDE – Integrated Development Kit) для написання програмного коду. В результаті був обраний JetBrains Rider, оскільки він краще взаємодії з редактором Unity3d, а також надає більше інструментів для рефакторингу коду

Були описані загальні складові ігрового процесу, такі як «Дія», «Реакція», «Відгук».

Описано структуру проекта та архітектурні шаблони, які використовувалися для проектування архітектури проекту. Серед зазначених шаблонів були: MVC (Model-View-Controller) та Dependency Injection.

Описано алгоритм, який використовувався для генерації безкінечної башні.

Була описана система графічного користувацького інтерфейсу.

Розглянуто призначення та умови використання ігрового додатку. Детально описано функції системи. Наведено інструкцію для користувачів як встановлювати та як використовувати цей програмний продукт.

В результаті була створено ігрова програма «Tower Runner».

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Платформер [Електронний ресурс]. – Режим доступу: https://uk.wikipedia.org/wiki/Платформер

2. Frogs [Electronic recource]. – Access mode: https://en.wikipedia.org/wiki/Frogs_(video_game)

3. Bloom A. Video Invaders / Bloom, Steve - Arco Publishing, 1982 – 29 p.

4. Pearl R. (June 1983). "Closet Classics" / Pearl R. – Electronic Games, 2015 –82 p.

5. Pitfall [Electronic recource]. – Access mode: https://ru.wikipedia.org/wiki/Pitfall!

6. Mario Bros [Electronic recource]. – Access mode: https://en.wikipedia.org/wiki/Mario_Bros

7. Manic Miner [Electronic recource]. – Access mode: https://en.wikipedia.org/wiki/Manic_Miner

8. Jet Set Willy [Electronic recource]. – Access mode: https://en.wikipedia.org/wiki/Jet_Set_Willy

9. Super Mario Bros [Electronic recource]. – Access mode: https://ru.wikipedia.org/wiki/Super_Mario_Bros.

10. Castlevania [Electronic recource]. – Access mode: https://ru.wikipedia.org/wiki/Castlevania

11. Lesser, P. The Role of Computers / K. Hartley // Dragon. — 1992. — December (no. 188). — P. 57–64.

12. Hazeldine, Julian. Epilogue / Speedrun: The Unauthorised History of Sonic The Hedgehog. — Lulu.com, 2014. — 127 p. — 142 p.

13. Temple Run [Electronic recource]. – Access mode: https://play.google.com/store/apps/details?id=com.imangi.templerun

14. Temple Run 2 [Electronic recource]. – Access mode: https://play.google.com/store/apps/details?id=com.imangi.templerun2&hl=ru&gl=US

15. Subway Surf [Electronic recource]. – Access mode: https://play.google.com/store/apps/details?id=com.kiloo.subwaysurf

16. Sonic Dash [Electronic recource]. – Access mode: https://play.google.com/store/apps/details?id=com.sega.sonicdash

17. Jungle Run [Electronic recource]. – Access mode: https://play.google.com/store/apps/details?id=com.pastagames.ro1mobile

18. Rodeo Safari [Electronic recource]. – Access mode: https://play.google.com/store/apps/details?id=com.yodo1.rodeo.safari

19. Showden Run 3D [Electronic recource]. – Access mode: https://en.wikipedia.org/wiki/Snowden_Run_3D

20. Jetpack [Electronic recource]. – Access mode: https://play.google.com/store/apps/details?id=com.halfbrick.jetpackjoyride

21. Donkey Kong [Electronic recource]. – Access mode: https://ru.wikipedia.org/wiki/Donkey_Kong_64

22. Crash Bandicoot [Electronic recource]. – Access mode: https://ru.wikipedia.org/wiki/Crash_Bandicoot

23. Spyro the Dragon [Electronic recource]. – Access mode: https://spyro.fandom.com/wiki/Spyro_the_Dragon_(video_game)

24. Helix Crush [Electronic recource]. – Access mode: https://play.google.com/store/apps/details?id=com.amanotes.pamahelixtiles&hl=ru&gl=US

25. Dancing Road: Color Ball Run! [Electronic recource]. – Access mode: https://play.google.com/store/apps/details?id=com.amanotes.pamadancingroad

26. Jump For Queen 2020: Rescue on the Helix [Electronic recource]. – Access mode: https://play.google.com/store/apps/details?id=com.PreTechX.queenjumpfantasyofthehelix

27. Helix Stack Ball [Electronic recource]. – Access mode: https://play.google.com/store/apps/details?id=com.smash.dropstackballfall&hl=ru&gl=US

28. Фриман А. Pro ASP.NET MVC 4, 4th edition / А. Фриман – М.: «Вильямс», 2013. – 688 с.

29. Robert C. Martin. Clean Code: A Handbook of Agile Software Craftsmanship. — Pearson Education, 2008. — P. 157.

30. Крэг Ларман. Применение UML 2.0 и шаблонов проектирования = Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development. — 3-е изд. — М.: Вильямс, 2006. — 736 с.

31. Гральний рушій [Електронний ресурс]. – Режим доступу: http://wikipedia.ua.nina.az/wiki/Unity_(гральний_рушій)

32. Хокинг, Джозеф. Unity – в действии. Мультиплатформенная разработка на C#: [рус.]. – 2. – СПб: Питер, 2016. — 336 с

33. Unity3d [Электронный ресурс]. – Режим доступа: http://wikipedia.org/wiki/Unity_(игровой_движок)

34. Unreal Engine 4 [Electronic recource]. – Access mode: https://www.unrealengine.com/en-US/.

35. Game Maker: Studio [Electronic recource]. – Access mode: https://www.yoyogames.com/en/gamemaker.

36. Unity3d [Электронный ресурс]. – Режим доступа: https://conf.ztu.edu.ua/wp-content/uploads/2017/11/9.pdf

37. Visual Studio [Electronic recource]. – Access mode: https://visualstudio.microsoft.com.

38. Rider JetBrains [Electronic recource]. – Access mode: https://www.jetbrains.com/rider/.

39. Rouse, Richard. Game Design: Theory & Practice: – 2. – Los Rios Boulevard, Plano, Texas, USA: Wordware Publishing, 2004. — 698 с.

40. Ігровий процес [Електронний ресурс]. – Режим доступу: https://uk.wikipedia.org/wiki/Ігровий_процес

41. Object pool pattern [Electronic recource]. – Access mode: http:// en.wikipedia.org/wiki/Object_pool_pattern

ДОДАТОК А
ТЕКСТ ПРОГРАМИ

A.1 GameRoot.cs

```
using UnityEngine;

public class GameRoot : MonoBehaviour
{
    //ViewControllers
    public RestClient RestClient;
    public NotificationsController NotificationsController;
    public DailyRewardsController DailyRewardsController;
    public PlayerController PlayerController;
    public AIPlayerController AIPlayerController;
    public CameraController CameraController;
    public BallCollisionDetection BallCollisionDetection;
    public ShopManager ShopManager;
    public TowerGenerator TowerGenerator;
    public GameManager GameManager;
    public LevelThemeManager LevelThemeManager;
    public StaticRandom StaticRandom;
    public FirebaseInit FirebaseInit;
    public FacebookManager FacebookManager;
    public AnalyticEvents AnalyticEvents;
    public LevelMenuManager LevelMenuManager;
    public IAP IAP;
    public AdMob AdMob;
    public AppReview AppReview;
    public GPGManager GpgManager;
    public SoundManager SoundManager;
    public FPSCounter FPSCounter;
    private SaveController _saveController;
    private Vibration _vibration;
    private UIScreenController _uiScreenController;

    //Models
    private ShopModel _shopModel;
    private SaveModel _saveModel;

    private void Awake()
    {
        InitModels();
        InitControllers();
    }
    private void InitModels()
    {
        _shopModel = new ShopModel();
        _saveModel = new SaveModel();
    }

    private void InitControllers()
    {
        _uiScreenController = new UIScreenController();
```

```
AnalyticEvents.Init(FirebaseInit,
    FacebookManager);

FacebookManager.Init();

_vibration = new Vibration();
_vibration.Init(AnalyticEvents);

AppReview.Init(LevelMenuManager);

NotificationsController.Init();

DailyRewardsController.Init(NotificationsController,
    _shopModel,
    SoundManager,
    GameManager);

FPSCounter.Init(AnalyticEvents);

SoundManager.Init(AnalyticEvents);

_saveController = new SaveController();
_saveController.Init(GameManager,
    _saveModel,
    LevelMenuManager);

GameManager.Init(TowerGenerator,
    _saveModel,
    ShopManager,
    AdMob,
    CameraController,
    LevelMenuManager,
    AppReview,
    GpgManager,
    SoundManager,
    FPSCounter,
    _shopModel,
    PlayerController,
    AIPlayerController,
    AnalyticEvents,
    _saveController,
    _uiScreenController,
    StaticRandom,
    RestClient);

StaticRandom.Init(GameManager, _saveModel);

TowerGenerator.Init(StaticRandom,
    LevelThemeManager,
    GameManager,
    SoundManager,
```

```
    _saveModel,
    PlayerController,
    LevelMenuManager);

LevelThemeManager.Init(TowerGenerator,
    PlayerController,
    GameManager);

PlayerController.Init(TowerGenerator,
    GameManager,
    ShopManager,
    LevelMenuManager,
    LevelThemeManager,
    GpgManager,
    SoundManager,
    _shopModel,
    CameraController);

// AIPlayerController.Init(TowerGenerator,
//     GameManager,
//     ShopManager,
//     LevelMenuManager,
//     LevelThemeManager,
//     GpgManager,
//     SoundManager,
//     PlayerController);

ShopManager.Init(_shopModel,
    _saveModel,
    PlayerController,
    _saveController,
    AdMob,
    LevelThemeManager,
    GpgManager,
    IAP,
    AnalyticEvents);

LevelMenuManager.Init(GameManager,
    ShopManager,
    AdMob,
    IAP,
    AppReview,
    GpgManager,
    SoundManager,
    _saveModel,
    _shopModel,
    _saveController,
    AnalyticEvents,
    _vibration,
    _uiScreenController);

FirebaseInit.Init(GameManager,
```

```
            LevelMenuManager,
            DailyRewardsController,
            _saveModel);

        IAP.Init(ShopManager,
            AdMob,
            LevelMenuManager,
            AnalyticEvents);

        AdMob.Init(ShopManager,
            IAP,
            _shopModel,
            AnalyticEvents,
            GameManager);

        BallCollisionDetection.Init(LevelThemeManager,
            SoundManager,
            _vibration);
    }
}
```

A.2 GameManager.cs

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

public class GameManager : MonoBehaviour
{
    public List<DifficultyLevel> difficultyLevels = new
List<DifficultyLevel>();
    public DifficultyLevel currentDifficultyLevel;

    public GameObject playerBall;
    public GameObject brokenBallPrefab;
    private GameObject brokenBall;
    private Rigidbody[] brokenBallParts;
    public float explosionPower;
    public float explosionRadius;

    public float locationIndicatorDistance = 13.0f;

    [SerializeField] private ParticleSystem[]
finishParticles;
    [SerializeField] private GameObject finishFireworks;

    [Header("UI")]
    [SerializeField] private GameObject duelPopup;
    [SerializeField] private GameObject loader;
    [SerializeField] private Text duelDescription;
    [SerializeField] private Text duelStartTimer;
```

```
        [SerializeField] private ContinueLevelView
_continueLevelView;

        private int frames;
        private string framesStr;
        private int seconds;
        private string secondsStr;
        private int minutes;
        private string minutesStr;
        private string currentTime;
        [HideInInspector]
        public bool startTheTimer;
        public bool IsGameOver { get; set; }

        private int brokenPlatforms;
        public int BrokenPlatforms
        {
            get { return brokenPlatforms; }
            set
            {
                if (_towerGenerator)
                {
                    // Analitic events
                    if (_saveModel.currentLevel == 0)
                    {
                        switch (value)
                        {
                            case 1:

_analyticEvents.ReportEvent("smashed_1");
                                break;
                            case 20:

_analyticEvents.ReportEvent("smashed_20");
```

```
                                break;
                            case 50:
_analyticEvents.ReportEvent("smashed_50");
                                break;
                        }
                    }

                    float progress = (float)value /
_towerGenerator.CirclesAmount;

                    _playUiScreen.UpdateProgressBar(progress);
                    _towerGenerator.LerpColor(progress);
                }

                brokenPlatforms = value;
            }
        }

        private int currentLevelCoins;
        public int CurrentLevelCoins
        {
            get { return currentLevelCoins; }
            set
            {
                _playUiScreen.UpdateCoins(_shopModel.Coins +
value, true);
                currentLevelCoins = value;
            }
        }

        public bool invertControls;
        public bool IsContinueLevel;
```

```
public int CurrentLvlTroughFrames()
{
    return frames + (seconds * 50) + minutes * 60 * 50;
}

public SaveModel _saveModel;

public GameObject loadingPanel;

private TowerGenerator _towerGenerator;
private LevelMenuManager _levelMenuManager;
private AppReview _appReview;
private GPGManager _gpgManager;
private SoundManager _soundManager;
private FPSCounter _fpsCounter;
private PlayerController _playerController;
private ShopModel _shopModel;
private AnalyticEvents _analyticEvents;
private RestClient _restClient;

private UIScreenController _uiScreenController;

//Screens
private SelectionUIScreen _selectionUiScreen;
private PlayUIScreen _playUiScreen;

[Header("Duel settings")]
public int duelBet = 20;
public int secondsToStartDuel = 5;

[Header("Continue Level settings")]
public int ContinueLevelCountdown = 5;
```

```
public RestClient.Track CurrentTrack;

private bool recordTrack;
private bool _isNoContinueLevel;

public GameMode gameMode;

public enum GameMode
{
    Single,
    Duel
}

public void Init(TowerGenerator towerGenerator,
    SaveModel saveModel,
    ShopManager shopManager,
    AdMob adMob,
    CameraController cameraController,
    LevelMenuManager levelMenuManager,
    AppReview appReview,
    GPGManager gpgManager,
    SoundManager soundManager,
    FPSCounter fpsCounter,
    ShopModel shopModel,
    PlayerController playerController,
    AIPlayerController aiPlayerController,
    AnalyticEvents analyticEvents,
    SaveController saveController,
    UIScreenController uiScreenController,
    StaticRandom staticRandom,
    RestClient restClient)
{
    _towerGenerator = towerGenerator;
```

```
            _saveModel = saveModel;
            _levelMenuManager = levelMenuManager;
            _appReview = appReview;
            _gpgManager = gpgManager;
            _soundManager = soundManager;
            _fpsCounter = fpsCounter;
            _shopModel = shopModel;
            _playerController = playerController;
            _analyticEvents = analyticEvents;
            _uiScreenController = uiScreenController;
            _restClient = restClient;

            Application.targetFrameRate = 60;

            switch
(Firebase.RemoteConfig.FirebaseRemoteConfig.GetValue("difficulty_l
evel").StringValue)
            {
                case "medium_level":
                    currentDifficultyLevel = difficultyLevels[1];
                    break;
                default:
                    currentDifficultyLevel = difficultyLevels[0];
                    break;
            }

            Thread.CurrentThread.CurrentCulture =
System.Globalization.CultureInfo.InvariantCulture;

            _uiScreenController.HideAll();
            _selectionUiScreen =
_uiScreenController.GetScreen<SelectionUIScreen>().Init(_saveModel
, _shopModel);
            _selectionUiScreen.Show(true);
            _selectionUiScreen.UpdateInfo();
```

```
_uiScreenController.GetScreen<NavigationUIScreen>().UpdateNavigati
onsButtons(_saveModel);


                CurrentTrack = new RestClient.Track();
            }


            private void Update()
            {
        #if UNITY_EDITOR
                if (startTheTimer && Input.GetKeyDown(KeyCode.Space))
                {
                    Finish();
                }
        #endif


                if
(!_levelMenuManager.controlsTutorial.activeInHierarchy)
                {
                    if (_playerController.isTouching &&
_playerController.enabled && !startTheTimer)
                    {
                        {
                            StartGame();
                        }
                    }


                    spentTime = (float)minutes * 60f + (float)seconds
+ (float)frames / 50f;
                }


                /*
                // Update location indicator
                if (gameMode == GameMode.Duel && startTheTimer)
```

```
            {
                if
(_aiPlayerController.playerTransform.position.y >
_playerController.playerTransform.position.y +
locationIndicatorDistance)
                {
                    _playUiScreen.ShowLocationUp();
                }
                else if
(_aiPlayerController.playerTransform.position.y <
_playerController.playerTransform.position.y -
locationIndicatorDistance)
                {
                    _playUiScreen.ShowLocationDown();
                }
                else
                {
                    _playUiScreen.HideLocation();
                }
            }
            */
        }

        void FixedUpdate()
        {
            if (startTheTimer)
            {
                ConvertFramesToTime();
            }
        }

        public void SelectLevel(int value)
        {
            _analyticEvents.ReportEvent("navigation_tap");
```

```
            ChangeLevel(value);

            _uiScreenController.HideAll();
            _selectionUiScreen.Show(true);
            _selectionUiScreen.UpdateInfo();

_uiScreenController.GetScreen<NavigationUIScreen>().UpdateNavigati
onsButtons(_saveModel);

            Restart();
        }

        public void Replay()
        {
            _analyticEvents.ReportEvent("play_again_button", new
Dictionary<string, object> { { "level", _saveModel.currentLevel +
1 } });

            ChangeLevel(0);

            var level =
_saveModel.GetLevel(_saveModel.currentLevel);
            if (level != null)
            {
                StaticRandom.Pointer = level.staticRandomPointer;
            }
            else
            {
                StaticRandom.Pointer = 0;
            }

            OpenPlayScreen();
            Restart();
        }
```

```
        public void PlayNextLevel()
        {
            ChangeLevel(1);

            OpenPlayScreen();
            Restart();
        }

        private void ChangeLevel(int value) // value = 1 or 0 or
-1
        {
            PlayFinishParticles(false);

            // Change level
            _saveModel.currentLevel += value;

            if (_saveModel.currentLevel <= 0)
            {
                _saveModel.currentLevel = 0;
            }
            else if (_saveModel.currentLevel >
_saveModel.levels.Count - 1)
            {
                _saveModel.currentLevel = _saveModel.levels.Count
- 1;
            }

            _levelMenuManager.CloseSettings();

            _saveModel.Save();
        }

        public void OpenPlayScreen()
        {
```

```
                gameMode = GameMode.Single;

                _saveModel.GetLevel(_saveModel.currentLevel).played =
true;

                _uiScreenController.HideAll();

                _playUiScreen =
_uiScreenController.ShowScreen<PlayUIScreen>(true);

_playUiScreen.UpdateLevelTexts(_saveModel.currentLevel + 1,
_saveModel.currentLevel + 2);

                //PlayerPrefs.DeleteKey("tutorial");
                if (!PlayerPrefs.HasKey("tutorial"))
                {
                    _playUiScreen.ShowTutorial(true);
                    PlayerPrefs.SetInt("tutorial", 1);
                    PlayerPrefs.Save();
                }
                else
                {
                    _playUiScreen.ShowTutorial(false);
                }

                if (PlayerPrefs.GetInt("firstPlay") <= 0)
                {
                    PlayerPrefs.SetInt("firstPlay", 1);
                    _analyticEvents.ReportEvent("first_play");
                }
                else
                {
                    _analyticEvents.ReportEvent("play_button", new
Dictionary<string, object> { { "level", _saveModel.currentLevel +
1 } });
```

```
            }


            CurrentTrack.level = _saveModel.currentLevel;
            CurrentTrack.skin = new RestClient.Track.Skin { group
= _shopModel.CurrentSkin.group, item = _shopModel.CurrentSkin.item
};
            CurrentTrack.effect = new RestClient.Track.Skin {
group = _shopModel.CurrentEffect.group, item =
_shopModel.CurrentEffect.item };
            CurrentTrack.moves.Clear();
        }


        public void ShowDuelPopup()
        {
            duelDescription.text = $"The price of participation
in a duel - {duelBet} coins. The winner receives {duelBet * 2}
coins.";


            duelPopup.SetActive(true);
        }


        public void AcceptDuel()
        {
            if (_shopModel.Coins >= duelBet)
            {
                _shopModel.Coins -= duelBet;
                duelPopup.SetActive(false);


                OpenDuelScreen();
            }
            else
            {
                // Not enough coins
                duelDescription.text = $"Not enough coins. The
price of participation in a duel - {duelBet} coins.";
            }
```

```
        }

        public void OpenDuelScreen()
        {
            gameMode = GameMode.Duel;

            _analyticEvents.ReportEvent("start_duel");

            _uiScreenController.HideAll();

            _playUiScreen =
_uiScreenController.ShowScreen<PlayUIScreen>(true);

_playUiScreen.UpdateLevelTexts(_saveModel.currentLevel + 1,
_saveModel.currentLevel + 2);

            if (!PlayerPrefs.HasKey("tutorial"))
            {
                _playUiScreen.ShowTutorial(true);
                PlayerPrefs.SetInt("tutorial", 1);
                PlayerPrefs.Save();
            }
            else
            {
                _playUiScreen.ShowTutorial(false);
            }

            StopCoroutine("StartCountdown");
            StartCoroutine("StartCountdown");
        }

        WaitForSeconds countdownDelay = new WaitForSeconds(1.0f);

        IEnumerator StartCountdown()
```

```
{
    int count = secondsToStartDuel;

    duelStartTimer.text = "";

    while (count > 0)
    {
        duelStartTimer.text = count.ToString();
        count--;

        yield return countdownDelay;
    }

    duelStartTimer.text = "Start!";

    StartGame();

    yield return countdownDelay;

    duelStartTimer.text = "";
}

IEnumerator StartContinueLevelCountdown()
{
    int count = ContinueLevelCountdown;

    _continueLevelView.UpateTimerText(count.ToString());

    while (count > 0)
    {
        count--;

        yield return countdownDelay;
```

```
_continueLevelView.UpateTimerText(count.ToString());

                if (IsContinueLevel || _isNoContinueLevel)
                {
                    _isNoContinueLevel = false;
                    IsContinueLevel = false;

                    yield break;
                }
            }

            StartCoroutine(EndLevel());
        }

        // Sync players and start
        public void StartDuel()
        {

        }

        public void StartGame()
        {
            PlayFinishParticles(false);

            if (gameMode == GameMode.Single)
            _playUiScreen.ShowTutorial(false);

            startTheTimer = true;
            _fpsCounter.report = true;
            _playUiScreen.ShowTimer(true);
```

```
            minutes = 0;
            seconds = 0;
            frames = 0;

            // Analitic events
            switch (_saveModel.currentLevel + 1)
            {
                case 2:
_analyticEvents.ReportEvent("2_level_launched");
                    break;
                case 10:
_analyticEvents.ReportEvent("10_level_launched");
                    break;
                case 50:
_analyticEvents.ReportEvent("50_level_launched");
                    break;
            }
        }

        public void SetCoinsCounterColor(Color color)
        {
_uiScreenController.GetScreen<PlayUIScreen>().SetCoinsCounterColor
(color);
        }

        public void UpdateCoins(int value, bool animate)
        {
_uiScreenController.GetScreen<PlayUIScreen>().UpdateCoins(value,
animate);
        }
```

```
        private void ConvertFramesToTime()
        {
            if (minutes < 100)
            {
                if (frames < 50)
                {
                    frames++;
                }
                else
                {
                    frames = 0;
                    if (seconds < 60)
                    {
                        seconds++;
                    }
                    else
                    {
                        seconds = 0;
                        minutes++;
                    }
                }
            }
            framesStr = frames < 10 ? "0" + frames.ToString() :
frames.ToString();
            secondsStr = seconds < 10 ? "0" + seconds.ToString()
: seconds.ToString();
            minutesStr = minutes < 10 ? "0" + minutes.ToString()
: minutes.ToString();
            currentTime = minutesStr + ":" + secondsStr + ":" +
framesStr;

            if(_playUiScreen != null)
            {
                _playUiScreen.UpdateTimerText(currentTime);
            }
```

```
        }

        public void GameOver()
        {
            // Start game over after count down finished
            _towerGenerator.rotSpeed = 50;
            OnGameOver();
        }

        public void ReInitPlayer(Vector3 spawnPos)
        {
           _playerController.powerAmount = 0;
           _playerController.ReCreatePlayer(spawnPos);
        }

        void OnGameOver()
        {
            _playerController.TempPlayerForRecreating =
_playerController.ballAnim.gameObject;

            _shopModel.Coins = _shopModel.Coins +
currentLevelCoins;
            CurrentLevelCoins = 0;

            //Too hard for one finger
            _gpgManager.UnlockAchivement(5);

            _fpsCounter.report = false;

            _soundManager.PlaySound(2);

            DeathController deathController =
GameObject.FindGameObjectWithTag("SkinGFX").GetComponent<DeathCont
roller>();
```

```
            MeshRenderer gfx =
playerBall.GetComponentInChildren<MeshRenderer>();
            if (gfx.transform.childCount > 1)
            {
                deathController.BreakSkin(gfx.material,
explosionPower, explosionRadius);
            }
            else
            {
                deathController.BreakSkin();
            }

            playerBall.SetActive(false);
            startTheTimer = false;

            _playerController.enabled = false;
            _playerController.PowerAmount = 0;

            IsGameOver = true;

            StartCoroutine(EndLevel());
        }

        IEnumerator EndLevel()
        {
            _continueLevelView.gameObject.SetActive(false);

            _uiScreenController.ShowScreen<PlayUIScreen>(false);

            _levelMenuManager.CloseSettings();

            var loseScreen =
_uiScreenController.ShowScreen<LoseUIScreen>(true);
```

```
_uiScreenController.GetScreen<NavigationUIScreen>().UpdateNavigati
onsButtons(_saveModel);


                loseScreen.UpdateInfo(brokenPlatforms, _saveModel,
_shopModel);
                BrokenPlatforms = 0;


                yield return new WaitForEndOfFrame();
            }


            public void ContinueLevelShowReward()
            {
                IsContinueLevel = true;


                _analyticEvents.ReportEvent("continue_game");
            }


            public void ContinueLevelRewardEarned()
            {

ReInitPlayer(_playerController.playerTransform.position);
                _continueLevelView.gameObject.SetActive(false);
                startTheTimer = true;


                _analyticEvents.ReportEvent("rewarded_banner");
            }


            public void NoContinueLevel()
            {
                _isNoContinueLevel = true;
                StartCoroutine(EndLevel());


                _analyticEvents.ReportEvent("break_game");
            }
```

```
        public void Restart()
        {
            _fpsCounter.report = false;

            StaticRandom.Pointer =
_saveModel.GetLevel(_saveModel.currentLevel).staticRandomPointer;

            ReInitPlayer(Vector3.zero);

            _towerGenerator.ResetCirclePointer();
            _towerGenerator.GenerateLevel();
        }

        public void Finish()
        {
            StartCoroutine(FinishGame());
        }

        IEnumerator FinishGame()
        {
            _playerController.EnableEffect(false);

            _fpsCounter.report = false;

            switch (_saveModel.currentLevel + 1)
            {
                case 1:
                    //Half done
                    _gpgManager.UnlockAchivement(1);
                    break;
                case 100:
                    //I can go on like this all day
```

```
                    _gpgManager.UnlockAchivement(3);
                    break;
                }

                _playerController.OnPointerUp(null);

                _soundManager.PlaySound(5); //complete level sound

                _playerController.PowerAmount = 0;

_playerController.fireBallLoading.gameObject.SetActive(false);
                startTheTimer = false;
                _playerController.isTouching = false;

                // Update level data
                var level =
_saveModel.GetLevel(_saveModel.currentLevel);
                level.completed = true;
                level.coins = CurrentLevelCoins;
                level.time = CurrentLvlTroughFrames();

                int rating = (int)RateLevel();

                if (rating > level.rating) level.rating = rating;

                if (level.staticRandomPointer <= 0)
level.staticRandomPointer = StaticRandom.Pointer;

                // Create next level
                if (_saveModel.GetLevel(_saveModel.currentLevel + 1)
== null)
                {
                    var newLevel = new SaveModel.LevelData();
                    newLevel.theme =
_saveModel.GetCurrentLevel().theme + 1;
```

```
                if (newLevel.theme >
_towerGenerator.themeColors.Count - 1)
                    newLevel.theme = 0;

                newLevel.id = _saveModel.currentLevel + 1;
                newLevel.staticRandomPointer =
StaticRandom.Pointer;

                _saveModel.levels.Add(newLevel);
            }

            _saveModel.Save();

            _shopModel.Coins += CurrentLevelCoins;

            CurrentLevelCoins = 0;
            BrokenPlatforms = 0;

            //_playUiScreen.PlayFireworks();
            PlayFinishParticles(true);

            yield return new WaitForSeconds(1.0f);

            _appReview.RepeatCounter++;
            yield return new WaitForEndOfFrame();



            _levelMenuManager.CloseSettings();
            _uiScreenController.ShowScreen<PlayUIScreen>(false);

            int seconds = Mathf.CeilToInt(spentTime);
            var timespan = TimeSpan.FromSeconds(seconds);
```

```
            _uiScreenController.ShowScreen<WinUIScreen>(true).
                UpdateInfo(timespan.ToString(@"mm\:ss"),
_saveModel, _shopModel);

            _selectionUiScreen.UpdateInfo();

            _restClient.AddTrack(CurrentTrack, result => { });

_uiScreenController.GetScreen<NavigationUIScreen>().UpdateNavigati
onsButtons(_saveModel);
        }

        public void PlayFinishParticles(bool play)
        {
            foreach (var p in finishParticles)
            {
                if (play)
                {
                    p.Play();
                }
                else
                {
                    p.Stop();
                    p.Clear();
                }
            }
        }

        public TowerGenerator towerGenerator;

        private float spentTime;
        public enum Rate
        {
```

```
            None,
            One,
            Two,
            Three
        };

        private Rate RateLevel()
        {
            Rate value = Rate.None;

            float goldTime = towerGenerator.CirclesAmount *
0.07f;

            //Debug.Log($"Spent time: {spentTime} Gold time:
{goldTime} Rating:{Mathf.CeilToInt(((spentTime / goldTime) *
100f))}%");

            int result = Mathf.CeilToInt(((spentTime / goldTime)
* 100f));

            if (result <= 100)
            {
                value = Rate.Three;
            }
            else if (result > 100 && result < 200)
            {
                value = Rate.Two;
            }
            else
            {
                value = Rate.One;
            }

            return value;
        }
```

```
        [Serializable]
        public class DifficultyLevel
        {
            public string name = "simple";
            public List<Difficulty> difficulty = new
List<Difficulty>();
        }

        [Serializable]
        public class Difficulty
        {
            public int difficulty;
            public int level;
            public string values;
        }
    }
```

A.3 PlayerController.cs

```
using System;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.EventSystems;
using UnityEngine.Rendering;
using UnityEngine.UI;

public class PlayerController : MonoBehaviour,
IPointerDownHandler, IPointerUpHandler, IDragHandler
{
    public float fallSpeed;
    public Rigidbody ballRg;
    private SphereCollider ballCollider;
    public Animation ballAnim;
```

```
[HideInInspector]
public bool isTouching;
[HideInInspector]
public bool isReadyToDestroy;
[HideInInspector]
public GameObject TempPlayerForRecreating;
public GameObject SkinGFX;
private float timer = 0f;

[Header("Power Mode")]
public float powerModeDuration = 3.0f;
public int enablePowerAmount = 20;
public float reducePower = 0.1f;
public float reducePowerInPowerMode = 0.25f;

public Image fireBallLoading;
public GameObject warningSign;
public ParticleSystem ignitionTrail;
public ParticleSystem fireTrail;
public TrailRenderer mainTrail;

public float spinSpeed = 15;

public Transform playerTransform;

public float moveShifting = 0.5f;

public float powerAmount;
public float PowerAmount
{
    get { return powerAmount; }
    set
    {
```

```
            if (value != powerAmount)
            {
                powerAmount = value;

                if (value >= 100)
                {
                    if (!IsPowerMode)
                    {
                        IsPowerMode = true;
                        fireBallLoading.color = Color.red;

                        Ignite();

                        _soundManager.PlaySound(3); //ignition
sound
                        _soundManager.PlaySound(4); //burning
sound
                    }
                } else if (value <= 30 && value > 0 &&
IsPowerMode)
                {
                    warningSign.SetActive(true);
                }
                else if (value <= 0)
                {
                    if (IsPowerMode)
                    {
                        IsPowerMode = false;
                        fireBallLoading.fillAmount = 0f;
                        fireBallLoading.color = Color.white;
                        warningSign.SetActive(false);

                        EnableEffect(false);
```

```
                        _soundManager.StopSound(4); //burning
sound
                    }
                }
            }
        }
    }

    private void Ignite()
    {
        StopCoroutine("Ignition");
        StartCoroutine("Ignition");
    }

    WaitForSeconds ignitionDelay = new WaitForSeconds(0.5f);

    IEnumerator Ignition()
    {
        ignitionTrail.Stop();
        ignitionTrail.Play();

        yield return ignitionDelay;

        EnableEffect(true);
    }

    public float playerPitchValue = 1;

    bool isPowerMode;
    public bool IsPowerMode {
        get { return isPowerMode; }
        set
        {
```

```
            if (value != isPowerMode)
            {
                if (value)
                {
                    //Like a Johnny Storm
                    _gpgManager.UnlockAchivement(2);

                    if (_towerGenerator.collectCoinsMode ==
TowerGenerator.CollectCoinsMode.PowerMode)
                    {

_levelMenuManager.collectCoinAnimator.Play("CollectedCoin2D", -1,
0);
                        _gameManager.CurrentLevelCoins += 5;
                        _soundManager.PlaySound(6);
                    }
                }

                _levelThemeManager.skinTrail.emitting = !value;
            }

            isPowerMode = value;
        }
    }

    private float hitSoundTimer;
    public float hitSoundDelay = 0.5f;

    private BallCollisionDetection ballCollisionDetection;

    public float checkDragTimer;
    private float checkDragDelay = 0.1f;

    public bool drag;
```

```
private float dragDelta;
private Vector3 _startPlayerPosRg;
private Vector3 _startPlayerPosAnim;

private TowerGenerator _towerGenerator;
private GameManager _gameManager;
private ShopManager _shopManager;
private LevelMenuManager _levelMenuManager;
private LevelThemeManager _levelThemeManager;
private GPGManager _gpgManager;
private SoundManager _soundManager;
private ShopModel _shopModel;
private CameraController _cameraController;

private long moveDelay;
private long startMoveDelay;
private long endMoveDelay;

Vector3 startMovePos;
Vector3 endMovePos;

public void Init(TowerGenerator towerGenerator,
    GameManager gameManager,
    ShopManager shopManager,
    LevelMenuManager levelMenuManager,
    LevelThemeManager levelThemeManager,
    GPGManager gpgManager,
    SoundManager soundManager,
    ShopModel shopModel,
    CameraController cameraController)
{
    _towerGenerator = towerGenerator;
```

```
        _gameManager = gameManager;
        _shopManager = shopManager;
        _levelMenuManager = levelMenuManager;
        _levelThemeManager = levelThemeManager;
        _gpgManager = gpgManager;
        _soundManager = soundManager;
        _shopModel = shopModel;
        _cameraController = cameraController;

        fireBallLoading.fillAmount = 0f;

        ballCollider = ballRg.GetComponent<SphereCollider>();
        ballCollisionDetection =
ballRg.GetComponent<BallCollisionDetection>();

        _startPlayerPosRg = ballRg.transform.position;
        _startPlayerPosAnim = ballAnim.transform.localPosition;
        TempPlayerForRecreating = ballAnim.gameObject;
    }

    public void OnPointerDown(PointerEventData eventData)
    {
        SkinGFX.transform.localPosition =
_levelThemeManager.SkinShifting(moveShifting);

        isTouching = true;

        ballCollisionDetection.isCollidedUnbreakable = false;
        ballCollisionDetection.deathTime = 0f;
        ballCollisionDetection.isSampleAnimPlayed = false;

        startMoveDelay =
DateTimeOffset.Now.ToUnixTimeMilliseconds();
        startMovePos = playerTransform.position;
```

```
    }

    public void OnPointerUp(PointerEventData eventData)
    {
        SkinGFX.transform.localPosition =
_levelThemeManager.skinLocalPosition;

        playerPitchValue = 1;

        ballAnim.Play();

        hitSoundTimer = 0;

        ballCollider.isTrigger = false;
        isTouching = false;
        isReadyToDestroy = false;
        timer = 0f;

        drag = false;

        _towerGenerator.rotSpeed = 50;

        endMoveDelay =
DateTimeOffset.Now.ToUnixTimeMilliseconds();

        RecordMove();

        moveDelay = DateTimeOffset.Now.ToUnixTimeMilliseconds();
        endMovePos = playerTransform.position;
    }

    private void RecordMove()
    {
```

```
        float delay = moveDelay > 0 ? (startMoveDelay - moveDelay)
* 0.001f : 0;
        float time = (endMoveDelay - startMoveDelay) * 0.001f;


        _gameManager.CurrentTrack.moves.Add(new
RestClient.Track.Move
        {
            delay = delay,
            startY = startMovePos.y,
            endY = endMovePos.y,
        });
    }

    public float rotationSpeed;
    public float rotationDamping;
    private float _rotationVelocity;

    public void OnBeginDrag(PointerEventData eventData)
    {
        drag = true;
    }

    public void OnDrag(PointerEventData eventData)
    {
        _rotationVelocity = eventData.delta.x * rotationSpeed;
        _towerGenerator.towerHolder.transform.Rotate(Vector3.up,
_gameManager.invertControls ? -_rotationVelocity :
_rotationVelocity, Space.Self);
    }

    public void OnEndDrag(PointerEventData eventData)
    {
        drag = false;
    }
```

```
public void ReCreatePlayer(Vector3 spawnPos)
{
    isReadyToDestroy = false;
    isTouching = false;
    _gameManager.IsGameOver = false;

     timer = 0;

     if (spawnPos == Vector3.zero)
     {
         spawnPos = _startPlayerPosRg;
     }

     ballRg.transform.position = spawnPos;
     ballRg.gameObject.SetActive(true);
     fireBallLoading.gameObject.SetActive(true);

     _shopManager.ApplySkin(true);
     _shopManager.ApplyEffect();

    _cameraController.SetTarget(ballRg.transform);

    enabled = true;
}

void Update()
{
    // Can be used to add inertia to drag
    /*
    if (!drag && !Mathf.Approximately(_rotationVelocity, 0))
    {
        float deltaVelocity = Mathf.Min(
```

```
                Mathf.Sign(_rotationVelocity) * Time.deltaTime *
rotationDamping,
                Mathf.Sign(_rotationVelocity) * _rotationVelocity
            );
            _rotationVelocity -= deltaVelocity;

TowerGenerator.Instance.towerHolder.transform.Rotate(Vector3.up, -
_rotationVelocity, Space.Self);
        }
        */

        fireBallLoading.fillAmount = PowerAmount / 100f;

        if (isTouching && !isReadyToDestroy)
        {
            playerTransform.localScale = new Vector3(1, 0.5f, 1);

            timer += Time.deltaTime;
            if(timer >= 0.1f)
            {
                DestroyTower();
            }
        }
    }

    void FixedUpdate()
    {
        if (IsPowerMode)
        {
            PowerAmount = Mathf.Clamp(PowerAmount -=
reducePowerInPowerMode * Time.fixedDeltaTime, 0, 100);
        }
        else
        {
```

```
            PowerAmount = Mathf.Clamp(PowerAmount -= reducePower *
Time.fixedDeltaTime, 0, 100);
        }
    }

    public void EnableEffect(bool enable)
    {
        if (_levelThemeManager.effectGFX != null)
        {
            // If standart fire effect
            if (_shopModel.CurrentEffect.group <= 0 &&
_shopModel.CurrentEffect.item <= 0)
            {
                if (enable) fireTrail.Play();
                else fireTrail.Stop();

                if
(_levelThemeManager.effectGFX.TryGetComponent(out ParticleSystem
p))
                {
                    if (enable) p.Play();
                        else p.Stop();
                }
            }
            // Other effects
            else
            {
                foreach (Transform t in
_levelThemeManager.effectGFX.transform)
                {
                    if (t.TryGetComponent(out ParticleSystem p))
                    {
                        if (enable) p.Play();
                        else p.Stop();
                    }
```

```
                    if (t.TryGetComponent(out TrailRenderer tr))
                    {
                        tr.enabled = enable;
                    }
                }
            }
        }

    private void DestroyTower()
    {
        if (_gameManager.startTheTimer)
        {
            ballAnim.Stop();


            ballAnim.transform.SetParent(null);
            ballRg.transform.position = new
Vector3(ballRg.transform.position.x,
ballAnim.transform.position.y, ballRg.transform.position.z);
            ballAnim.transform.SetParent(ballRg.transform);


            ballCollider.isTrigger = true;
            ballRg.isKinematic = false;
            ballRg.AddForce(0f, fallSpeed * 100f, 0f);
            isReadyToDestroy = true;
        }
    }


    public void AddPower()
    {
        Mathf.Clamp(PowerAmount += 100 / enablePowerAmount, 0,
100);
    }
```

```
}
```

A.3 UIScreenController.cs

```
using System;
using UnityEngine;

public class UIScreenController
{
    private BaseUIScreen[] _uiScreens;

    public BaseUIScreen CurrentScreen { get; private set; }

    public UIScreenController()
    {
        _uiScreens = GameObject.FindObjectsOfType<BaseUIScreen>();
    }

    public T GetScreen<T>()
    {
        BaseUIScreen value = null;

        foreach (var screen in _uiScreens)
        {
            if (screen.GetType() == typeof(T))
            {
                value = screen;
            }
        }

        return (T) (object) value;
    }

    public T ShowScreen<T>(bool show)
```

```
    {
        BaseUIScreen value = null;

        foreach (var screen in _uiScreens)
        {
            if (screen.GetType() == typeof(T))
            {
                value = screen;
                value.Show(show);
            }
        }

        CurrentScreen = value;

        return (T) (object) value;
    }

    public void HideAll()
    {
        CurrentScreen = null;

        foreach (var screen in _uiScreens)
        {
            screen.Show(false);
        }
    }
}
```

A.5 LevelMenuManager.cs

```
using System;
using UnityEngine;
using UnityEngine.EventSystems;
using UnityEngine.UI;

public class LevelMenuManager : MonoBehaviour
```

```
{
    public GameObject splashLoader;

    public Image[] ratingStars;

    public GameObject controlsTutorial;
    public Animator collectCoinAnimator;

    //public Text highScore;

    [SerializeField] private Button[] noAdsButtons;

    public Text currentLvlTxt;
    public Text endLvlText;
    public GameObject closePauseMenuButton;

    public GameObject rewardedVideoLoader;

    public EventSystem eventSystem;

    private GameManager _gameManager;
    private ShopManager _shopManager;
    private AdMob _admob;
    private IAP _iap;
    private AppReview _appReview;
    private GPGManager _gpgManager;
    private SoundManager _soundManager;
    private SaveModel _saveModel;
    private ShopModel _shopModel;
    private SaveController _saveController;
    private AnalyticEvents _analyticEvents;
    private Vibration _vibration;
    private UIScreenController _uiScreenController;

    public void Init(GameManager gameManager,
        ShopManager shopManager,
        AdMob adMob,
        IAP iap,
        AppReview appReview,
        GPGManager gpgManager,
        SoundManager soundManager,
        SaveModel saveModel,
        ShopModel shopModel,
        SaveController saveController,
        AnalyticEvents analyticEvents,
        Vibration vibration,
        UIScreenController uiScreenController)
    {
        _gameManager = gameManager;
        _shopManager = shopManager;
        _admob = adMob;
        _iap = iap;
```

```
        _appReview = appReview;
        _gpgManager = gpgManager;
        _soundManager = soundManager;
        _saveModel = saveModel;
        _shopModel = shopModel;
        _saveController = saveController;
        _gameManager.invertControls =
Convert.ToBoolean(PlayerPrefs.GetInt("invertControls"));
        _shopModel = shopModel;
        _analyticEvents = analyticEvents;
        _vibration = vibration;
        _uiScreenController = uiScreenController;

        _gameManager.invertControls =
Convert.ToBoolean(PlayerPrefs.GetInt("invertControls"));
    }

    public GameObject mainCam;

    public SettingsSubmenuView settingsSubmenu;
    public bool isSettingsSubmenuOpened = false;

    public Image vibrationIcon;
    public Image soundIcon;
    public Image invertIcon;
    public Image tutorialInvertIcon;
    public Sprite[] invertIcons;

    public void UpdateSettingsIcons()
    {
        if (_vibration.vibration)
        {
            vibrationIcon.color = Color.white;
        }
        else
        {
            vibrationIcon.color = Color.gray;
        }

        if (_soundManager.sound)
        {
            soundIcon.color = Color.white;
        }
        else
        {
            soundIcon.color = Color.gray;
        }

        tutorialInvertIcon.sprite =
invertIcons[Convert.ToInt32(!_gameManager.invertControls)];
        invertIcon.sprite =
invertIcons[Convert.ToInt32(!_gameManager.invertControls)];
```

```
    }

    public void OnSkinsButtonClicked()
    {
        _shopManager.Open();

        CloseSettings();
    }

    public void OnNoAdsClicked()
    {
        _analyticEvents.ReportEvent("no_ads_on");

        CloseSettings();

#if !UNITY_EDITOR
        _iap.BuyNonConsumable(0);
#endif
    }

    public void CloseSettings()
    {
        isSettingsSubmenuOpened = false;
        settingsSubmenu.Show(false);

        UpdateOptions();
        UpdateNavigation();
    }

    private void UpdateNavigation()
    {
        var navigationUIScreen =
_uiScreenController.GetScreen<NavigationUIScreen>();

        if (!isSettingsSubmenuOpened)
        {

navigationUIScreen.UpdateNavigationsButtons(_saveModel);
        }
    }

    public void OnUnlockAllClicked()
    {
#if UNITY_EDITOR
        //IAP.Instance.UnlockAll = true;
#endif

        _analyticEvents.ReportEvent("unlock_all_on");

#if !UNITY_EDITOR
        _iap.BuyNonConsumable(1);
#endif
```

```
    }

    public void OnSettingsButtonClicked()
    {
        isSettingsSubmenuOpened = !isSettingsSubmenuOpened;

        settingsSubmenu.Show(isSettingsSubmenuOpened);

        UpdateOptions();
        UpdateNavigation();
        UpdateSettingsIcons();
    }

    private void UpdateOptions()
    {

_uiScreenController.GetScreen<SelectionUIScreen>().ShowOptions(!is
SettingsSubmenuOpened);

_uiScreenController.GetScreen<WinUIScreen>().ShowOptions(!isSettin
gsSubmenuOpened);
    }

    public void OnSoundButtonClicked()
    {
        _soundManager.ToggleSound();
        UpdateSettingsIcons();
    }

    public void OnVibroButtonClicked()
    {
        _vibration.ToggleVibration();
        UpdateSettingsIcons();
    }

    public void OnInvertClicked()
    {
        _gameManager.invertControls =
!_gameManager.invertControls;
        PlayerPrefs.SetInt("invertControls",
Convert.ToInt16(_gameManager.invertControls));

        UpdateSettingsIcons();
    }

    public void OnGameRatingButtonClicked()
    {
        OnSettingsButtonClicked();
        UpdateRatingStars();

        _appReview.OpenReviewPanel();
    }
```

```
    public void UpdateRatingStars()
    {
        foreach (var img in ratingStars)
        {
            img.color = Color.gray;
        }

        for (int i = 0; i < _appReview.rating; i++)
        {
            ratingStars[i].color = Color.white;
        }
    }

    public void OnInfoButtonClicked()
    {
        Application.OpenURL("https://break-tower.club/terms-and-
conditions");
    }

    public void OnAchivementsButtonClicked()
    {
        _gpgManager.ShowAchivements();
    }
}
```

A.6 TowerGenerator.cs

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


public class TowerGenerator : MonoBehaviour
{
    private const int POOL_SIZE = 40;


    public CollectCoinsMode collectCoinsMode;
    public enum CollectCoinsMode
    {
        PowerMode,
        CollectOnlevel
    }
```

```
    public GameObject coinPrefab;

    public bool globalRotation = true;

    [Header("0-100%")][Range(0, 100)] public int coinsPerLevel =
0;
    public float collectCoinRange = 3.0f;

    public float colorLerpSpeed = 0.1f;

    //Empty which is parent of all tower samples
    [HideInInspector]
    public GameObject towerHolder;
    //Tower samples prefabs
    public GameObject[] towerSamplePrefabs;
    private GameObject chosenTowerSample;

    public TowerCirclesPool towerCirclesPool;
    public int CirclesAmount { get; private set; }
    public float finishOffset;
    public float posOffset;
    public float rotOffset;
    public float rotSpeed;

    //Finish platform prefab - support platform
    public GameObject finishPlatform;
    private GameObject finishPlatformGenerated;
    //Support cylinder
    public GameObject supportCylinder;
    private GameObject supportCylinderGenerated;

    public List<ThemeColor> themeColors = new List<ThemeColor>();
    public ThemeColor CurrentThemeColor;
```

```
public Material safeMat;
public Material dangerousMat;

public SpriteRenderer backroundRenderer;

//Amount of unique themes in the tower:
[HideInInspector]
public int uniqThemes;

private StaticRandom _staticRandom;
private LevelThemeManager _levelThemeManager;
private SoundManager _soundManager;
private GameManager _gameManager;
private PlayerController _playerController;
private SaveModel _saveModel;
private LevelMenuManager _levelMenuManager;

private int _currentCirclePointer;

public void Init(StaticRandom staticRandom,
    LevelThemeManager levelThemeManager,
    GameManager gameManager,
    SoundManager soundManager,
    SaveModel saveModel,
    PlayerController playerController,
    LevelMenuManager levelMenuManager)
{
    _staticRandom = staticRandom;
    _levelThemeManager = levelThemeManager;
    _soundManager = soundManager;
    _gameManager = gameManager;
    _playerController = playerController;
```

```
        _saveModel = saveModel;
        _levelMenuManager = levelMenuManager;

        towerCirclesPool = new TowerCirclesPool(POOL_SIZE);

        StaticRandom.Pointer =
_saveModel.GetLevel(_saveModel.currentLevel).staticRandomPointer;

        CreateTowerHolder();
        GenerateLevel();
    }

    public static float Clamp0360(float eulerAngles)
    {
        float result = eulerAngles - Mathf.CeilToInt(eulerAngles /
360f) * 360f;
        if (result < 0)
        {
            result += 360f;
        }
        return result;
    }

    private void CreateTowerHolder()
    {
        towerHolder = new GameObject("TowerHolder");
        towerHolder.transform.position = Vector3.zero;
        towerHolder.transform.rotation = Quaternion.identity;
        towerHolder.tag = "TowerHolder";
    }

    private void ChooseSafeMaterial()
    {
```

```
        CurrentThemeColor =
themeColors[_saveModel.GetCurrentLevel().theme];
        safeMat.color = CurrentThemeColor.startColor;
    }

    public void GenerateLevel()
    {
        ChoosePlatforms();

        towerCirclesPool.Clear();

towerCirclesPool.Fill(chosenTowerSample.name,chosenTowerSample,
towerHolder.transform);
        towerCirclesPool.SetOrderIndexToZero();

        CirclesAmount = 70 +
Mathf.RoundToInt((_saveModel.currentLevel + 2) * 70f * 0.05f);
        _currentCirclePointer = POOL_SIZE;

        ChooseSafeMaterial();

        _levelThemeManager.choosenThemeColor =
CurrentThemeColor.startColor;

        // Init circles positions
        for (int i = 0; i < towerCirclesPool.GetCircles().Count;
i++)
        {
            var circle = towerCirclesPool.GetCircles()[i];
            var samples =
circle.GetComponentsInChildren<SampleController>();
            foreach (var sample in samples)
            {
                sample.Init(this, _gameManager,
_levelThemeManager, _soundManager, _playerController);
            }
```

```
            //Set the position and rotation for generated sample
relative to the previous one
            if (i > 0)
            {
                circle.transform.position =
towerCirclesPool.GetCircles()[i - 1].transform.position - new
Vector3(
                    0f,
                    towerCirclesPool.GetCircles()[i -
1].transform.localScale.y * posOffset,
                    0f);
                circle.transform.rotation *=
towerCirclesPool.GetCircles()[i - 1].transform.rotation *
Quaternion.Euler(0f, rotOffset, 0f);
            }
        }


        if (finishPlatformGenerated == null)
        {
            finishPlatformGenerated = Instantiate(finishPlatform,
Vector3.zero, Quaternion.identity, towerHolder.transform);
        }


        var pos =
towerCirclesPool.GetCircles()[towerCirclesPool.GetCircles().Count
- 1].transform.position;
        finishPlatformGenerated.transform.position = new
Vector3(pos.x, pos.y - finishOffset, pos.z);


        //if (supportCylinderGenerated == null)
        //{
        //    supportCylinderGenerated =
Instantiate(supportCylinder, Vector3.zero, Quaternion.identity,
towerHolder.transform);
        //}
        //
        //supportCylinderGenerated.name = "supportCylinder";
```

```
//supportCylinderGenerated.transform.SetParent(finishPlatformGener
ated.transform);
        //supportCylinderGenerated.transform.localScale = new
Vector3(supportCylinder.transform.localScale.x,
chosenTowerSample.transform.localScale.y * CirclesAmount,
        //    supportCylinder.transform.localScale.z);
        //supportCylinderGenerated.transform.localPosition =
Vector3.zero;


        SetSamplesThemes();


        if(collectCoinsMode == CollectCoinsMode.CollectOnlevel)
            AddCoins();


        var l = _saveModel.GetLevel(_saveModel.currentLevel);
        if (_saveModel.currentLevel == 0 && l.staticRandomPointer
== 0)
        {
            l.staticRandomPointer = StaticRandom.Pointer;
            GenerateLevel();
        }
    }


    public void ActivateNextSample()
    {
        if (_currentCirclePointer == CirclesAmount)
        {
            var circleController =
towerCirclesPool.GetElementInOrder().GetComponent<CircleController
>();
            circleController.SetParent();
            return;
        }


        GameObject previousLast = null;
```

```
        var circle = towerCirclesPool.Next(out previousLast);


        circle.transform.position =
previousLast.transform.position - new Vector3(
            0f,
            previousLast.transform.localScale.y * posOffset,
            0f);
        circle.transform.eulerAngles =
previousLast.transform.eulerAngles + new Vector3(0, rotOffset, 0);


        circle.GetComponent<CircleController>().Reset();


        var pos =
towerCirclesPool.GetCircles()[towerCirclesPool.GetCircles().Count
- 1].transform.position;
        finishPlatformGenerated.transform.position = new
Vector3(pos.x, pos.y - finishOffset, pos.z);


        //finishPlatformGenerated.transform.position =
previousLast.transform.position - new Vector3(
        //    0f,
        //    previousLast.transform.localScale.y * finishOffset,
        //    0f) * 2;


        _currentCirclePointer++;
    }


    public void ResetCirclePointer()
    {
        _currentCirclePointer = 0;
    }


    private void AddCoins()
    {
        System.Random r = new System.Random();
```

```
        foreach (GameObject s in towerCirclesPool.GetCircles())
        {
            SampleController[] sc =
s.transform.GetComponentsInChildren<SampleController>(true);

            foreach (var i in sc)
            {
                if (r.Next(0, 100) < coinsPerLevel)
                {
                    i.AddCoin();
                }
            }
        }
    }

    void Update()
    {
        //Tower rotation
        if(towerHolder != null &&
!_levelMenuManager.controlsTutorial.activeInHierarchy)
        {
            if (_playerController == null ||
!_playerController.drag)
            {
                towerHolder.transform.Rotate(0f, rotSpeed *
Time.deltaTime, 0f);
            }
        }
    }

    public void LerpColor(float value)
    {
        safeMat.color = Color.Lerp(CurrentThemeColor.startColor,
CurrentThemeColor.endColor, value);
```

```
    }

    private void ChoosePlatforms()
    {
        int[] tempRandomValuesSample = new
int[towerSamplePrefabs.Length];
        int tempActualValueSample = 0;

        for (int r = 0; r < towerSamplePrefabs.Length; r++)
        {
            tempRandomValuesSample[r] =
_staticRandom.StaticRange();
        }
        for (int a = 0; a < tempRandomValuesSample.Length; a++)
        {
            if (tempRandomValuesSample[a] > tempActualValueSample)
            {
                tempActualValueSample = tempRandomValuesSample[a];
            }
        }
        for (int f = 0; f < tempRandomValuesSample.Length; f++)
        {
            if (tempActualValueSample ==
tempRandomValuesSample[f])
            {
                chosenTowerSample = towerSamplePrefabs[f];
                break;
            }
        }
    }

    //Sets sample themes using static random:
    void SetSamplesThemes()
    {
```

```
        int setSamples = 0;
        List<bool> theme = new List<bool>();
        List<bool> tempTheme = new List<bool>();
        bool isThemeActive = false;
        int lastSampleEdited = 0;


        while (setSamples <= towerCirclesPool.GetCircles().Count)
        {
            int sameThemeSamples =
_staticRandom.StaticRange10();//Random.Range(1,
Mathf.RoundToInt(towerSamples.Count / 6f) + 1);
            for (int i = 0; i < sameThemeSamples; i++)
            {
                if (towerCirclesPool.GetCircles()[i +
lastSampleEdited].transform.childCount > 1)
                {
                    if (!isThemeActive)
                    {
                        //Debug.Log(savingManager.CurrentLvl);


                        int sampleChildCount =
towerCirclesPool.GetCircles()[i +
lastSampleEdited].transform.childCount;
                        for (int index = 0; index <
sampleChildCount; index++)
                        {
                            bool themeElement =
_staticRandom.StaticRange() % 2 == 0;
                            theme.Add(themeElement);
                        }
                        for (int currentElement = 0;
currentElement < theme.Count; currentElement++)
                        {
                            if (!theme[currentElement])
                            {
```

```
tempTheme.Add(theme[currentElement]);
                            }
                        }
                        if (theme.Count == tempTheme.Count)
                        {
                            int[] tempRandomValues = new
int[theme.Count];
                            int tempActualValue = 0;
                            for(int r = 0; r < theme.Count; r++)
                            {
                                tempRandomValues[r] =
_staticRandom.StaticRange();
                            }
                            for(int a = 0; a <
tempRandomValues.Length; a++)
                            {
                                if(tempRandomValues[a] >
tempActualValue)
                                {
                                    tempActualValue =
tempRandomValues[a];
                                }
                            }
                            for(int f = 0; f <
tempRandomValues.Length; f++)
                            {
                                if(tempActualValue ==
tempRandomValues[f])
                                {
                                    theme[f] = true;
                                    break;
                                }
                            }
                        }
                        isThemeActive = true;
```

```
                    }


                    if (towerCirclesPool.GetCircles()[i +
lastSampleEdited].transform.childCount == theme.Count)
                    {
                        for (int themeElement = 0; themeElement <
theme.Count; themeElement++)
                        {
                            if (theme[themeElement])
                            {
                                //towerSamples[i +
lastSampleEdited].transform.GetChild(themeElement).GetComponent<Me
shRenderer>().material = ChoosenSafeMaterial;
                                towerCirclesPool.GetCircles()[i +
lastSampleEdited].transform.GetChild(themeElement).GetComponent<Me
shRenderer>().material = safeMat;
                                towerCirclesPool.GetCircles()[i +
lastSampleEdited].transform.GetChild(themeElement).tag =
"Breakable";
                            }
                            else
                            {
                                towerCirclesPool.GetCircles()[i +
lastSampleEdited].transform.GetChild(themeElement).GetComponent<Me
shRenderer>().material = dangerousMat;
                                towerCirclesPool.GetCircles()[i +
lastSampleEdited].transform.GetChild(themeElement).tag =
"UnBreakable";
                            }
                        }
                    }
                }


                if (i == sameThemeSamples - 1)
                {
                    isThemeActive = false;
                    for (int themeI = theme.Count - 1; themeI >=
0; themeI--)
```

```
                    {
                        theme.RemoveAt(themeI);
                    }
                    for (int tempThemeI = tempTheme.Count - 1;
tempThemeI >= 0; tempThemeI--)
                    {
                        tempTheme.RemoveAt(tempThemeI);
                    }
                    lastSampleEdited += sameThemeSamples;
                }
                setSamples++;
                if (setSamples ==
towerCirclesPool.GetCircles().Count)
                {
                    uniqThemes++;
                    return;
                }
            }
            uniqThemes++;
        }
    }

    [Serializable]
    public class ThemeColor
    {
        public Color startColor;
        public Color endColor;
        public Color backgroundStartColor;
        public Color backgroundEndColor;
        public Gradient endEffectColor;
    }
}
```

A.7 CameraController.cs

```
using UnityEngine;

public class CameraController : MonoBehaviour
{
    [SerializeField] Transform ballTransform;
    private Vector3 distance;
    private Vector3 dampCurrentVelocity;
    public float smoothTime;

    public float offsetSpeed;
    public bool enableDownOffset;

    private void Awake()
    {
         distance = transform.position - ballTransform.position;
    }

    public void SetTarget(Transform target)
    {
        ballTransform = target;
    }

    void LateUpdate()
    {
        transform.position =
Vector3.SmoothDamp(transform.position, ballTransform.position +
distance, ref dampCurrentVelocity, smoothTime);
    }

    public void ApplyOffset()
    {
```

```
        distance = new Vector3(distance.x, distance.y +
offsetSpeed, distance.z);
    }
}
```

A.8 CameraController.cs

```
using UnityEngine;

public class CameraController : MonoBehaviour
{
    [SerializeField] Transform ballTransform;
    private Vector3 distance;
    private Vector3 dampCurrentVelocity;
    public float smoothTime;

    public float offsetSpeed;
    public bool enableDownOffset;

    private void Awake()
    {
         distance = transform.position -
ballTransform.position;
    }

    public void SetTarget(Transform target)
    {
        ballTransform = target;
    }

    void LateUpdate()
    {
        transform.position =
Vector3.SmoothDamp(transform.position, ballTransform.position +
distance, ref dampCurrentVelocity, smoothTime);
```

```
        }

        public void ApplyOffset()
        {
            distance = new Vector3(distance.x, distance.y +
offsetSpeed, distance.z);
        }
    }
```

A.9 BallColisionDetection.cs

```
using UnityEngine;

public class BallCollisionDetection : MonoBehaviour
{
    private Rigidbody ballRg;
    public PlayerController playerCtrl;
    private SphereCollider ballCollider;

    [HideInInspector]
    public bool isCollidedUnbreakable;
    [HideInInspector]
    public float deathTime;
    [HideInInspector]
    public bool isSampleAnimPlayed;
    private Animation sampleAnim;
    public GameManager gameManager;
    private LevelThemeManager _levelThemeManager;
    private SoundManager _soundManager;
    private Vibration _vibration;

    public void Init(LevelThemeManager levelThemeManager,
        SoundManager soundManager,
```

```
        Vibration vibration)
    {
        _levelThemeManager = levelThemeManager;
        _soundManager = soundManager;
        _vibration = vibration;
    }

    void Start()
    {
        ballRg = GetComponent<Rigidbody>();
        ballCollider = GetComponent<SphereCollider>();
    }
    private void FixedUpdate()
    {
        if (ballRg.velocity.y <= 0f)
        {
            ballRg.velocity = new Vector3(ballRg.velocity.x,
Mathf.Clamp(ballRg.velocity.y, -23f, 0f), ballRg.velocity.z);
        }
        if (isCollidedUnbreakable && playerCtrl.isReadyToDestroy)
        {
            if (!playerCtrl.IsPowerMode)
            {
                deathTime += Time.fixedDeltaTime;
                if (deathTime >= 0.2f)
                {
                    playerCtrl.playerTransform.localScale = new
Vector3(1, 1, 1);

                    gameManager.GameOver();
                }
                else if (deathTime >= 0.05f &&
!isSampleAnimPlayed)
```

```
                {
                    sampleAnim.Play();
                    isSampleAnimPlayed = true;
                }
            }
            else
            {
                ballRg.velocity = new Vector3(ballRg.velocity.x,
Mathf.Clamp(ballRg.velocity.y, -23f, 0f), ballRg.velocity.z);
            }
        }
    }
    private void OnCollisionEnter(Collision collision)
    {
        if (collision.collider.CompareTag("Breakable") ||
collision.collider.CompareTag("UnBreakable"))
        {
            transform.position = new Vector3(transform.position.x,
collision.collider.bounds.max.y + ballCollider.radius + (-
ballCollider.center.y), transform.position.z);
            ballRg.isKinematic = true;
        }
        else if (collision.collider.CompareTag("FinishPlatform")
&& !gameManager.IsGameOver)
        {
            gameManager.Finish();
        }
    }


    private void OnTriggerEnter(Collider other)
    {
        if(_vibration != null) _vibration.Vibrate(5); //<100ms not
work on all devices
```

```
        if (!playerCtrl.isTouching)
            _soundManager.PlaySound(0); //jump sound

        if (!playerCtrl.IsPowerMode)
        {
            if (other.CompareTag("UnBreakable"))
            {
                ballCollider.isTrigger = false;
                transform.position = new
Vector3(transform.position.x, other.bounds.max.y +
ballCollider.radius + (-ballCollider.center.y),
transform.position.z);

                sampleAnim =
other.gameObject.transform.parent.GetComponent<Animation>();
                isCollidedUnbreakable = true;

                playerCtrl.SkinGFX.transform.localPosition =
_levelThemeManager.skinLocalPosition;
            }
        }
        if (other.CompareTag("FinishPlatform"))
        {
            ballCollider.isTrigger = false;
            transform.position = new Vector3(transform.position.x,
other.bounds.max.y + ballCollider.radius + (-
ballCollider.center.y), transform.position.z);
        }
    }
}
```

ДОДАТОК Б
СЛАЙДИ ПРЕЗЕНТАЦІЇ

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЗАПОРІЗЬКА ПОЛІТЕХНІКА»

Дипломна кваліфікаційна робота на тему:

РОЗРОБКА ІГРОВОЇ ПРОГРАМИ «TOWER RUNNER» ДЛЯ МОБІЛЬНОГО ТЕЛЕФОНУ

DEVELOPMENT OF THE TOWER RUNNER GAME SOFTWARE FOR A MOBILE PHONE

Виконав: ст. гр. КНТ-217 Є.С. Лизя

Керівник: проф. С.О Субботін

Рисунок Б.1 – Титульний слайд

МЕТА ТА ЗАВДАННЯ РОБОТИ 2

Мета роботи - програмна реалізація ігрової програми в жанрі платформеру на операційну систему Android.

ЗАДАЧІ РОБОТИ
аналіз вимог до системи;
розробка архітектури системи;
розробка дизайну клієнтської частини;
вибір інструментарію розробки;
програмна реалізація віртуального середовища;
розробка інструкцій з використання програми.

Рисунок Б.2 – Мета та завдання роботи

АНАЛІЗ ФУНКЦІОНАЛЬНИХ ВИМОГ 3

Розроблювана ігрова програма в жанрі платформеру повинна бути оптимізованою для мобільних девайсів, зручною у використанні та цікава в процесі використання.

Програмний продукт повинен містити наступний функціонал:
Пересування по ігровій локації зверху в вниз
Логіка виграшу та програшу
розробка дизайну клієнтської частини
Генерація бескінечних рівнів
Навігація по рівням складності
Налаштування гри

Рисунок Б.3 – Аналіз функціональних вимог

ДОДАТКИ АНАЛОГИ 4

Helix Crush

Dancing Road: Color Ball Run!

Jump for Queen 2020: Rescue on the Helix

Helix Stack Ball

Рисунок Б.4 – Додатки аналоги

Порівняння аналогів

Додаток	Графіка	Управління	Користувацький інтерфейс	Музикальний супровід
Helix Stack Ball	4	5	5	3
Jump for Queen 2020: Rescue on the Helix	2	5	3	3
Dancing Road: Color Ball Run!	5	5	5	5
Helix Crush	4	5	5	5

Рисунок Б.5 – Порівняння аналогів

Порівняння Unity3d, Unreal Engine 4 та Game Maker: Studio

Критерій	Unity3d	Unreal Engine 4	Game Maker: Studio
Якісна документація	5	4	4
Функціонал без додаткових розширень	4	5	5
Адаптивність	5	3	4
Функціонал	5	4	3
Швидкість роботи	5	3	5
Масштабованість	5	4	5
Просте вивчення	5	3	4
Кросплатформеність	5	4	5
Підтримка 3D	5	5	1
Безкоштовне використання	5	4	5

Рисунок Б.6 – Вибір фреймворку для розробки

ВИБІР СЕРЕДОВИЩА ПРОГРАМУВАННЯ 6

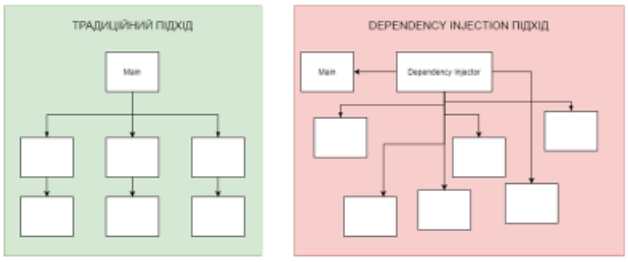
Порівняння IDE

Критерій	Rider	Visual Studio
Можливість роботи на різних платформах	+	+
Безкоштовна ліцензія	-	+
Якісна документація	+	+
Інтеграція з консоллю Unity3d	+	-
Інтеграція з gameobjects Unity3d	+	-
Система контролю версій	+	+
Приємний дизайн	+	+
Оптимізованість	+	-
Декомпіляція зовнішніх бібліотек	+	-

Рисунок Б.7 – Вибір середовища програмування

ПРОЕКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ / АРХІТЕКТУРА СИСТЕМИ 7

Схема MVC



Dependency Injection

Рисунок Б.8 – Проектування програмного забезпечення / Архітектура системи

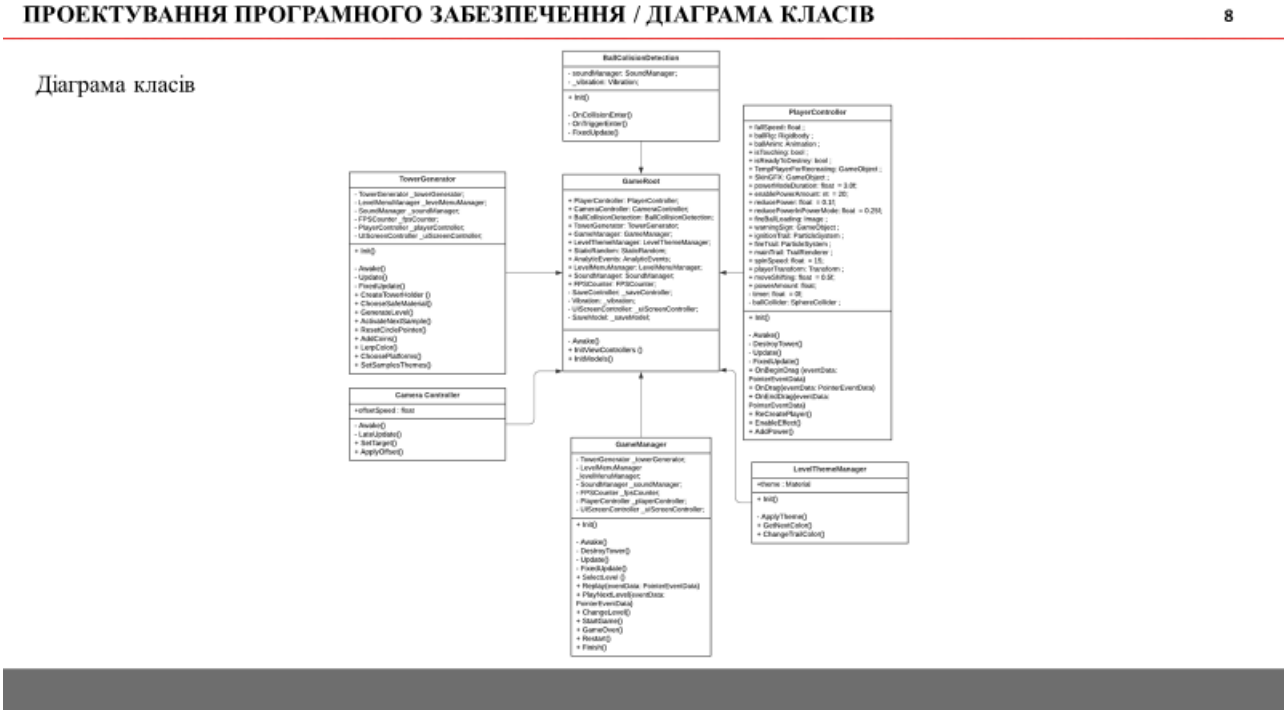


Рисунок Б.9 – Проектування програмного забезпечення / Діаграма класів

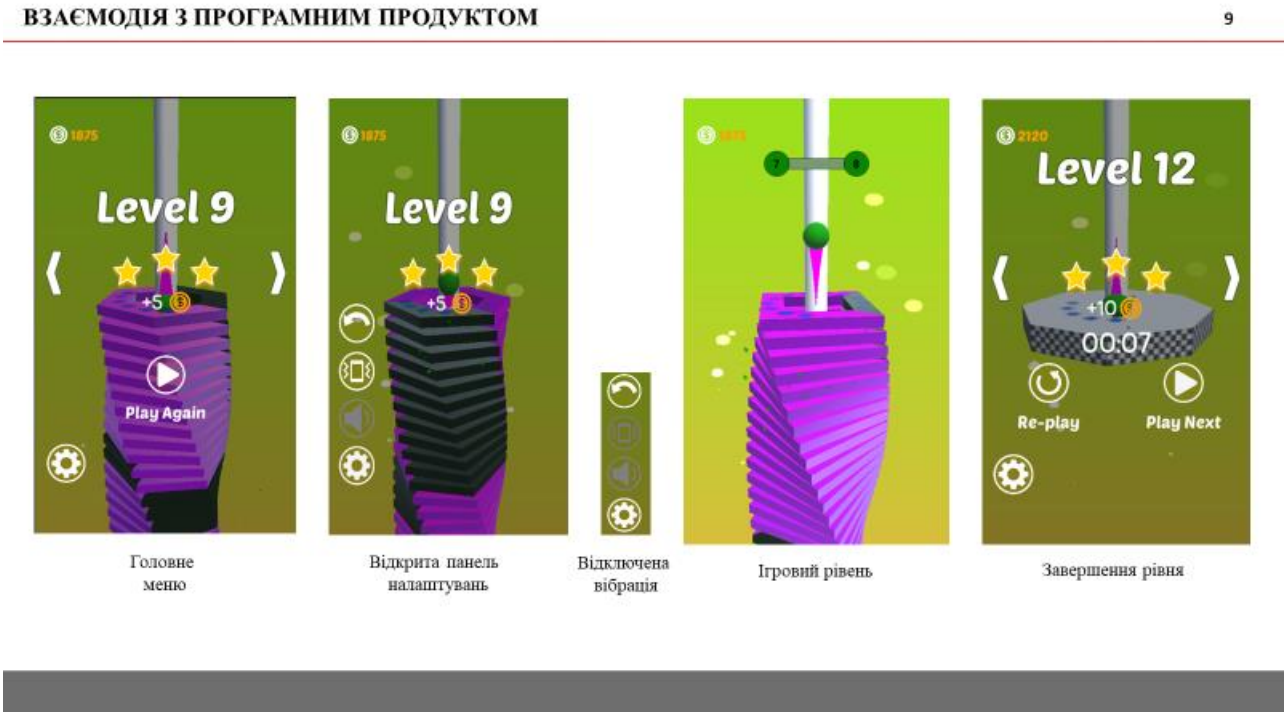


Рисунок Б.10 – Взаємодія з програмним продуктом

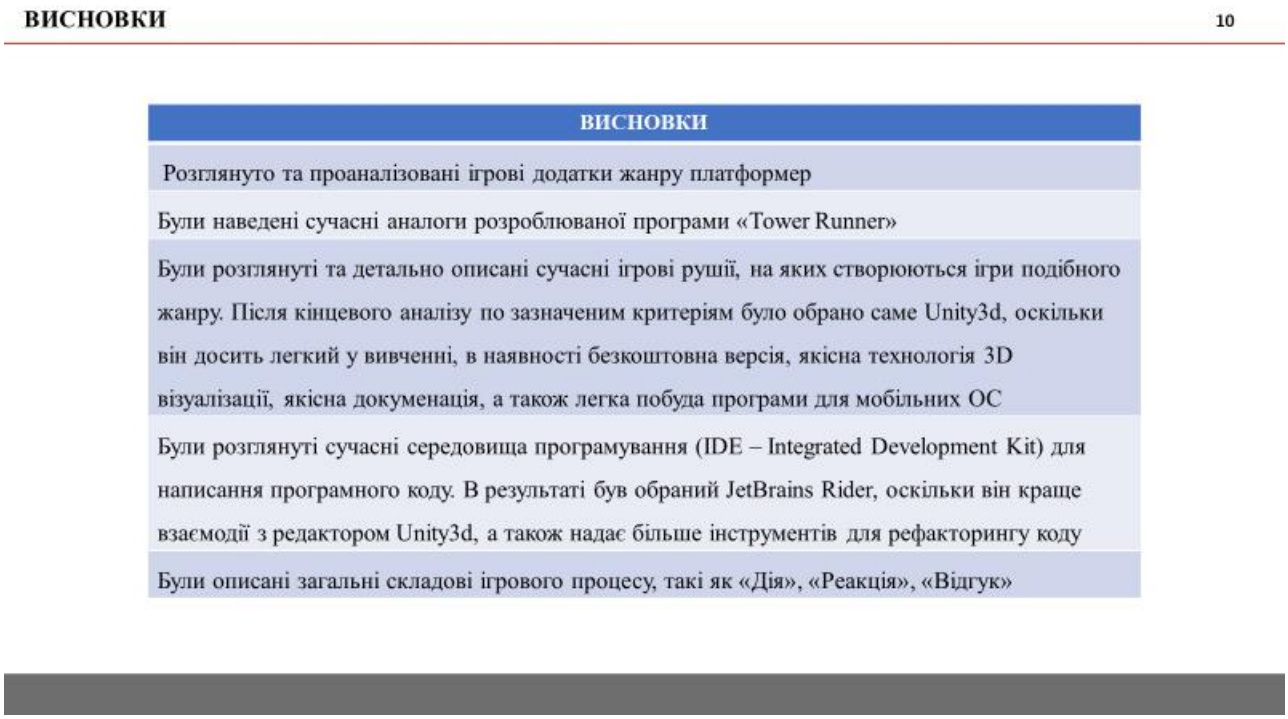


Рисунок Б.11 – Висновки