РЕФЕРАТ

Пояснювальна записка: ___ с., ___ рис., ___ табл., __ дод., ___ джерел.

Об'єкт розробки: програмний додаток для реалізації комп’ютерної гри - платформера.

Мета кваліфікаційної роботи: аналіз доступних технологій та методів розробки для створення ігрових програм та розробка власної комп’ютерні 3D ігри в середовищі UNITY 3D.

У вступі розглядається аналіз та сучасний стан проблеми, конкретизується мета кваліфікаційної роботи та галузь її застосування, наведено обґрунтування актуальності теми та уточнюється постановка завдання.

У першому розділі проведено аналіз предметної галузі, визначено актуальність завдання та призначення розробки, розроблена постановка завдання, задані вимоги до програмної реалізації, технологій та програмних засобів.

У другому розділі виконано аналіз існуючих рішень, обрано платформу для розробки, виконано проєктування і розробка програми, наведено опис алгоритму і структури функціонування програми, визначені вхідні і вихідні дані, наведені характеристики складу параметрів технічних засобів, описаний виклик та завантаження програми, описана робота програми.

В економічному розділі визначено трудомісткість розробленої інформаційної системи, проведений підрахунок вартості роботи по створенню програми та розраховано час на його створення.

Практичне значення розробленої роботи полягає в створенні повноцінного працеспроможного додатка, що надає можливість реалізувати комп’ютерну гру, в якій використовується сучасний графічний дизайн, невибагливі правила гри, зміна налаштувань під конкретного гравця, та який призначений для розваги користувачів.

Актуальність даного програмного продукту полягає в тому, що в наш час є доцільним використання комп’ютерних відеоігор для розваг, а саме ця програма за допомогою сучасних методів розробки в середовищі UNITY 3D реалізує гру популярного жанру в тривимірному просторі, а ідеї, закладені в її основу, також можуть бути в подальшому використані для створення нових ігор аналогічного спрямування.

Список ключових слів: ГРА, КОМПОНЕНТ, ПЕРСОНАЖ, ПРОЕКТУВАННЯ, ПРОГРАМУВАННЯ, ДИЗАЙН, ПРАВИЛА ГРИ, АЛГОРИТМ, ДІАГРАМА, НАЛАШТУВАННЯ.

ABSTRACT

The note is explained: ___ p., ___ fig., ___ table., __ dod., ___ dzherel.

Object of distribution: software add-on for the implementation of a computer gri - platformer.

Meta of high-quality robots: analysis of available technologies and methods of rooting for launching games programs rooting power computers 3D games in the middle of UNITY 3D.

At the entrance, the analysis of that current problem is examined, the meta of the quality robots and the fixation is concretized, the definition of the relevance is brought up by those that specify the statement of the problem.

At the first distribution, an analysis of the subject hall was carried out, the relevance of the design and the designation of the distribution was determined, the setting of the factory was broken, the settings were set before the software implementation, technologies and software inputs.

In another section, a description of the analysis of existing solutions, a platform for processing, a description of the processing and distribution of programs, a description of the algorithm and structure of the functions of the programs, the values of the input and output parameters of the data,

In the economic distribution, the labor is assigned to the breakdown of the information system, the carrying out of work on the part of the robot and the program is insured for an hour at the end.

The practical significance of the broken robotic field is in the opening of the main legal support tool, so we have the ability to implement the computer group, in which the victorious graphic design can be used, the simple rules for the concrete presentation

The relevance of this software product is due to the fact that, in our hour, we are the assistant of the computer video games for rozvag, and the program itself for the additional modern methods of rosetting in the middle of UNITY 3D, realizing the gross popular triple genre - the shooter ïï basis, it can also be found in a local vicarystani for the establishment of new igors of an analogous way.

Keywords: GRA, COMPONENT, CHARACTER, PROJECT, PROGRAM, DESIGN, GRI RULES, ALGORITHM, DIAGRAM, NALASHTUVANNYA.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

ІС – інформаційна система;

ОС – операційна система;

ПЗ – програмне забезпечення;

ПК – персональний комп’ютер;

ІТ – інформаційні технології.

ЗМІСТ

ВСТУП

Ігри можуть нести не тільки розважальний характер, але і змушувати задуматися, переживати, піднімати серйозні глобальні та психологічні питання. Тому ігри можна назвати сучасним видом мистецтва, часом вони дарують емоцій не менше, ніж перегляд кінофільму або театральної постановки.

Розробка відеогри – складний процес, який вимагає широкого спектру умінь і включає в себе багато етапів, починаючи з аналізу та планування майбутньої гри, закінчуючи оптимізацією і просуванням користувачеві.

Метою кваліфікаційної роботи є аналіз доступних технологій та методів розробки для створення ігрових програм та розробка власної комп'ютерні 3D ігри в середовищі UNITY 3D.

Для досягнення поставленої мети, розробнику необхідно дослідити наступні завдання:

1. Проаналізувати жанри відеоігор.
2. Детально розібратися з властивостями аркадних комп'ютерних ігор.
3. Висунути вимоги до власної гри.
4. Створити дизайн ігрового оформлення.
5. Створити програму реалізацію власної комп'ютерної гри «Break your hands».

В результаті виконання роботи має бути реалізована 3D гра під назвою «Break your hands» в середовищі UNITY 3D.

Для її реалізації необхідно виконати наступні етапи проектування:

– візуалізація ігрового процесу;
– взаємодія з користувачем;
– логіка гри.

Практичне значення розробленої роботи полягає в створенні повноцінного працеспроможного додатка, що надає можливість реалізувати комп'ютерну гру, в якій використовується сучасний графічний дизайн,

невибагливі правила гри, зміна налаштувань під конкретного гравця, та який призначений для розваги користувачів.

Актуальність даного програмного продукту полягає в тому, що в наш час є доцільним використання комп’ютерних відеоігор для розваг, а саме ця програма за допомогою сучасних методів розробки в середовищі UNITY 3D реалізує гру популярного жанру в тривимірному просторі, а ідеї, закладені в її основу, також можуть бути в подальшому використані для створення нових ігор аналогічного спрямування.

Дана програма - гра може використовуватись користувачами для розважальних заходів та приємного відпочинку за комп’ютером.

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1. Загальні відомості з предметної галузі

Задля розваги люди почали створювати все більше і більше ігор. У деяких з них брався за основу перевірений роками спортивний принцип, де люди змагалися між собою. Але спочатку використовувалися не справжні гравці, а так звані NPC – об’єкти, наділені певною логікою розробником. Ці NPC почали створювати з метою перешкоди гравцю, для створення певного інтересу проходження гри. Таким чином, розпочалась історія ігор жанру шутер, котрий став основним у ігровій індустрії.

Платформер (англ. Platformer, platform game) – жанр комп'ютерних ігор, в яких основною рисою ігрового процесу є стрибання по платформах, лазіння по сходах, збирання предметів, зазвичай необхідних для завершення рівня. Деякі предмети, звані пауер-апамі (англ. Power-up), наділяють керованого гравцем персонажа особливою силою, яка зазвичай вичерпується згодом (наприклад: силове поле, прискорення, збільшення висоти стрибків). Колекційні предмети, зброю і «пауеер-ап» збираються зазвичай простим дотиком персонажа і для застосування не вимагають спеціальних дій з боку гравця. Рідше предмети збираються в «інвентар» героя і застосовуються спеціальної командою (така поведінка більш характерно для аркадних головоломок). Подібний жанр комп'ютерних ігор сайд-скроллер.

Локація. (Від лат. Locatio – розміщення, розподіл). Більшість сучасних тлумачних словників і енциклопедій визначають локацію в самому широкому сенсі як визначення місця розташування об'єкта. Нас цікавить більш вузьке значення терміна, яке використовується програмістами, що спеціалізуються на створенні комп'ютерних ігор. В цьому сенсі, локація – це окрема частина ігрового світу, яка має свої особливості – споруди, рельєф і природу. Вона дає можливість гравцеві здійснювати певні дії, які виконуються для проходження

сюжету, позасюжетного завдань або просто заради розваги. Вона ніколи не може бути окремою або ізольованою (за рідкісним винятком секретних або бонусних), адже локація – це невід'ємна частина ігрового світу. В залежності від значимості для сюжету локація може мати такі частини, як міста, поселення, бути безводної пустелею або густим і квітучим лісом. Одним словом, як реальний світ, так і ігровий різноманітний, і чимало різноманітності вносять саме локації.

Оскільки локація – це тривимірний простір, який повинен жити певним життям і так чи інакше реагувати на дії персонажів, то для її створення використовується ряд програм. Це не тільки генератори текстур і персонажів, це також скрипти, симулятори погодних умов, життєвих і природних процесів і, звичайно ж, процеси і операції, завдання яких – створення інтерактивності і можливості взаємодії.

1.2. Призначення розробки та область застосування

Останнім часом все частіше можна почути про збільшення масштабів розробки програмного забезпечення розважального характеру. До числа таких продуктів відносяться відеоігри. Через десятки років індустрія відеоігор зайняла фіксоване місце на ринку поряд з іншими розвагами сфери мультимедіа, таких як кіно, мультиплікація, музика. Це багатомільярдні компанії, що займаються розробкою розважальних продуктів та програмних забезпечень, що складаються з мільйонів рядків програмного коду і файлів мультимедіа.

Ігри стали охоплювати величезну аудиторію по всьому світу, з'являючись на різних ігрових пристроях. Вони стали складніше і масштабніше: покращилася обробка візуального простору, обробка фізики об'єктів і штучний інтелект. Але напрямна методологія розробки не змінилася. Сучасні комп'ютерні ігри – великі і складні програмні комплекси. Витрати тільки на програмну розробку часто вимірюються сотнями людино-місяців. За обсягом

задіяних технологій та залучених коштів і рівня професійної підготовки розробників ігрова індустрія давно зайняла аж ніяк не останнє місце в світі IT.

Розробка апаратної бази в секторі персональних комп'ютерів вже досить довго стимулюється саме ігровою індустрією. Розвиток багатьох програмних технологій, зокрема, всіх мислимих способів відтворення тривимірних зображень, йде саме завдяки іграм.

В наш час є доцільним використання комп’ютерних відеоігор для розваг, а саме ця програма за допомогою сучасних методів розробки в середовищі UNITY 3D реалізує гру популярного жанру – шутера в тривимірному просторі, а ідеї, закладені в її основу, також можуть бути в подальшому використані для створення нових ігор аналогічного спрямування.

Дана програма-гра може використовуватись користувачами для розважальних заходів та приємного відпочинку за комп’ютером.

1.3. Підстава для розробки

Підставою для розробки кваліфікаційної роботи бакалавра на тему «Розробка інтерактивної гри «Break Your Hands» у середовищі UNITY 3D» є наказ по Національному технічному університету «Дніпровська політехніка» від __.__. 2021р. № ____-_.

1.4. Постановка завдання

Метою кваліфікаційної роботи є аналіз доступних технологій та методів розробки для створення ігрових програм та розробка власної комп’ютерні 3D ігри в середовищі UNITY 3D.

Завданням даної роботи є спроектувати та розробити інтерактивну гру «Break Your Hands», жанру платформер у піксельній стилістиці з тривимірним оточенням для платформи Windows.

Для досягнення поставленої мети, необхідно дослідити наступні завдання:

1. Проаналізувати жанри відеоігор.
2. Детально розібратися з властивостями аркадних комп'ютерних ігор.
3. Висунути вимоги до власної гри.
4. Створити дизайн ігрового оформлення.
5. Створити програму реалізацію власної комп'ютерної гри «Break your hands».

В результаті виконання роботи має бути реалізована 3D гра під назвою «Break your hands» в середовищі UNITY 3D.

Для її реалізації необхідно виконати наступні етапи проектування:

– візуалізація ігрового процесу;
– взаємодія з користувачем;
– логіка гри.

Ігровий процес повинен візуально транслювати користувачеві максимально чутливе і плавне управління персонажем та ігровими об'єктами.

На етапі реалізації завдання розробки відеогри важливо обміркувати кожну ігрову механіку в деталях і її взаємодію з іншими; врахувати обмеження платформи, та дивитися на фінальний продукт очима користувача для уникнення непорозумінь.

На етапі проектування програми важливо вірно розробити основні функції відеогри, щоб потім «нарощувати» допоміжні на правильно працюючий код. Необхідно розумно розпоряджатися ресурсами платформи для якої ведеться розробка; по-мінімуму виділяти нову динамічну пам'ять і очищувати непотрібну та використовувати найшвидші методи для реалізації функції гри.

При тестуванні гри необхідно перевіряти всі функції і можливі варіанти дій користувача; більше за все слід перевіряти переміщення персонажа, так, як він використовується на протязі усієї гри; також штучний інтелект потребує особливої уваги. Використання платформ теж стоїть не на останній позиції, бо

не вірно працююча платформа може стати перешкодою в успішному проходженні рівня.

1.5. Вимоги до програми або програмного виробу

1.5.1. Вимоги до функціональних характеристик

Відеогра повинна мати два режими: для одного гравця – на клавіатурі і двох – проходження гри розподіляється між гравцем на клавіатурі та іншим на ігровому контролері (геймпаді). Основною ціллю є проходження рівнів, минаючи перешкоди і ворогів, з найвищою кількістю балів, водночас керуючи персонажем та ігровими об’єктами.

Вимоги до функціоналу гри:

1. Переміщення персонажа відбуватиметься в двомірному просторі, хоч оточення і складається з тривимірних об’єктів, можна рухатися тільки вліво, вправо, вгору та вниз. Персонаж та віртуальна камера повинна переміщуватися плавно.

2. Гравець повинен мати шкалу здоров'я, якщо воно закінчиться, то персонаж відродиться на старті або Checkpoint (точка збереження позиції), в такому випадку час продовжує йти і вороги не відновлюються.

3. Передбачити перезапуск рівня, шляхом обнулення прогресу. Контрольні точки активуються при досягненні ними гравця і працюють як точка відродження.

4. Висота стрибка повинна залежати від часу натискання клавіші (максимум пів секунди). Деякі об'єкти можуть підкидати та запускати гравця.

Система рейтингу:

1. Під час проходження рівня гравець повинен збирати лійки та уникати доторкання до дизлайків. Після проходження рівня формується рейтинг рівня: від зібраних лайків віднімаються дизлайки та коефіцієнт часу (відсоток від затрачених хвилин, секунд і мілісекунд) та множиться на множник складності рівня.

2. Рейтинг кожного рівня додається до спільного рейтингу гравця, який виводиться в головному меню, причому він перераховується при оновлені, тобто якщо при повторному проходженні рівня враховується тільки один, найкращий результат. Рейтинг повинен зберігатися на комп’ютер та завантажуватися при заходженні в гру.

Рівні складаються з наступних складових:

1. Поверхні (можуть бути розміщені на платформах):

– основа: матеріал, що становить рівень (підлога, стіни, стеля, платформи);

– лава: при падінні робот гине, а персонаж відроджується;

– шипи: наносять смертельну шкоду;

2. Платформи (мають індивідуальну швидкість та відображають кнопку, яка приводить їх до дії):

– лінійна: затиснута кнопка рухає вздовж шляху і назад;

– механічна: натискання кнопки підкидає гравця;

– обертаюча: затискання кнопки обертає на певний градус;

– висувна: затиснута кнопка висуває зі стіни.

Противниками в грі виступають роботи, вони повинні знаходитись на рівні та заважати гравцеві. При розміщені на поверхні повинні патрулювати її і не падати з неї та не застрявати в об’єктах. Якщо гравець потрапить до зони їх видимості – повинні прискорюватись та нападати на нього, завдаючи шкоди. Вони також мають запас міцності і так, як персонаж не має зброї, він може стрибати по роботах, завдаючи їм шкоди, або вбивати за допомогою особливостей рівня (скидати на шипи або лаву).

Вимоги до графічного інтерфейсу:

1. Управління кожним меню може проводитися з клавіатури або з геймпада та при необхідності замінювати один-одного без налаштувань. Якщо необхідні драйвера встановлені, гра повинна помічати підключений геймпад без перезапуску.

2. При проходженні рівня повинна відображатися кількість здоров’я

персонажа, зібрані лайки і дизлайки, та затрачений час. В любий момент гра може бути призупинена викликом меню паузи, що дозволяє переглянути клавіші управління, почати рівень заново та вийти до головного меню.

3. Головне меню в свою чергу повинно містити кнопки виходу з гри та ігрових налаштувань. Також в головному меню повинні міститися кнопки завантаження рівнів, причому якщо попередній рівень не пройдений, то наступний не доступний. Як спільний рейтинг, так і рейтинг рівня (при наведенні на кнопку, що його запускає) повинен бути відображений в меню.

Управління грою має відбуватися відповідними клавішами.

1.5.2. Вимоги до інформаційної безпеки

Надійність роботи програмного забезпечення залежить від надійності операційної системи, під управлінням якої вона буде функціонувати, а також надійності розроблюваного програмного забезпечення.

Для надійної роботи програмного забезпечення зі сторони операційної системи необхідно дотримуватися таких факторів:

- використання ліцензійного ПЗ;
- захист від зловмисних програм;
- надійність роботи веб-серверу;
- архівація даних на сервері;
- встановлення блоків безперебійного живлення.

Для надійності роботи програми зі сторони розроблюваного програмного забезпечення:

- перешкоджання несанкціонованого доступу такими методами як авторизація та інші;
- перевірка введених даних користувачем для виключення можливих зловмисних ін’єкцій.

1.5.3. Вимоги до складу та параметрів технічних засобів

Для запуску відеогри необхідна конфігурація ПК не гірше мінімальної (табл. 1.1), для комфортної гри не гірше рекомендованої (табл. 1.2).

Такий вибір компонентів є оправданим тому, що:

– процесори підтримують інструкції SSE2, необхідні для запуску;

– відеокарти підтримують версії DirectX, які використовуються;

– кількості і швидкості ОЗП вистачає для процесів;

– пам’яті ЖД вистачає для зберігання ігрових файлів;

– материнські плати підтримують підключенні компоненти;

– блоків живлення вистачає для живлення всіх компонентів. Також для одного гравця потрібна клавіатура і миша, а для другого – геймпад. Для виводу картинки потрібен монітор.

Таблиця 1.1

Мінімальна конфігурація для запуску гри

Тип	Модель	Характеристики
Процесор	Intel Core Duo E6600	2 ядра по 2.40GHz
Відеокарта	NVIDIA GeForce 9500 GT	CF 550, GDDR3 128-bit 1024mb
ОЗП	GR1600S3V64L11	DDR3, 2048MB, 1600 MHz
Жорсткий диск	WD3200AAJS	320GB, 7200RPM, 8MB, 3.5
Материнська плата	GA-G41MT-D3P -S2 -S2P	Socket 775, Intel G41
Блок живлення	GameMax GM	450W, ATX 20+4pin

Таблиця 1.2

Рекомендована конфігурація для запуску гри

Тип	Модель	Характеристики
Процесор	Intel Core i5-2500K	4 ядра по 3.3GHz
Відеокарта	GeForce GTX750 Ti	CF 1072, GDDR5 128-bit2048mb
ОЗП	GR2400D464L17S	DDR4, 4096MB, 2400 MHz
Жорсткий диск	WD10EZEX	1TB, 7200RPM, 64MB, 3.5
Материнська плата	ASUS P8B75-V	Socket 1155, Intel B75
Блок живлення	GameMax GM	600W, ATX 20+4pin

1.5.4. Вимоги до інформаційної та програмної сумісності

Реалізація додатку повинно бути виконано на платформі Unity, рекомендована мова програмування для написання скриптів та ігрової логіки – C#.

Програма має бути сумісною з операційними системами Microsoft Windows XP та її пізнішими версіями.

Програма повинна являти собою самостійний виконуваний модуль, бути структурована і закоментована.

РОЗДІЛ 2
ПРОЄКТУВАННЯ ТА РОЗРОБКА ПРОГРАМНОГО ПРОДУКТУ

2.1. Функціональне призначення програми

Комп’ютерна гра «Break your hands» розроблена для платформи Windows, жанр: платформер у піксельній стилістиці з тривимірним оточенням. Реалізація виконана на платформі Unity, мова програмування для написання скриптів та ігрової логіки – C#.

Кожен рівень спроектовано таким чином, що користувач повинен минаючи перешкоди і ворогів, з найвищою кількістю балів дійти до кінця рівня, водночас керуючи персонажем та ігровими об’єктами. Ігровий процес візуально транслює користувачеві максимально чутливе і плавне управління персонажем та ігровими об’єктами.

Переміщення персонажа відбувається в двомірному просторі, хоч оточення і складається з тривимірних об’єктів, можна рухатися тільки вліво, вправо, вгору та вниз.

Гравець має шкалу здоров'я, якщо воно закінчується, то персонаж відродиться на старті або Checkpoint (точка збереження позиції), в такому випадку час продовжує йти і вороги не відновлюються.

Виконується перезапуск рівня, шляхом обнулення прогресу. Контрольні точки активуються при досягненні ними гравця і працюють як точка відродження.

Висота стрибка залежить від часу натискання клавіші. Деякі об'єкти можуть підкидати та запускати гравця.

Під час проходження рівня гравець збирає лайки та уникає доторкання до дизлайків. Після проходження рівня формується рейтинг рівня: від зібраних лайків віднімаються дизлайки та коефіцієнт часу (відсоток від затрачених хвилин, секунд і мілісекунд) та множиться на множник складності рівня.

Рейтинг кожного рівня додається до спільного рейтингу гравця, який виводиться в головному меню, причому він перераховується при оновлені, тобто якщо при повторному проходженні рівня враховується тільки один, найкращий результат. Рейтинг зберігається на комп’ютер та завантажується при заходженні в гру.

Противниками в грі виступають роботи, вони знаходяться на рівні та заважають гравцеві. Якщо гравець потрапить до зони їх видимості – вони прискорюються та нападають на нього, завдаючи шкоди. Вони також мають запас міцності і так, як персонаж не має зброї, він може стрибати по роботах, завдаючи їм шкоди, або вбивати за допомогою особливостей рівня (скидати на шипи або лаву).

Управління кожним меню проводиться з клавіатури або з геймпада та при необхідності замінює один-одного без налаштувань. Якщо необхідні драйвера встановлені, гра помічає підключений геймпад без перезапуску.

При проходженні рівня відображається кількість здоров’я персонажа, зібрані лайки і дизлайки, та затрачений час. В любий момент гра може бути призупинена викликом меню паузи, що дозволяє переглянути клавіші управління, почати рівень заново та вийти до головного меню.

Головне меню в свою чергу містить кнопки виходу з гри та ігрових налаштувань. Також в головному меню містяться кнопки завантаження рівнів, причому якщо попередній рівень не пройдений, то наступний не доступний.

2.2. Опис застосованих математичних методів

Під час проходження рівня гравець збирає лайки та уникає доторкання до дизлайків. Після проходження рівня формується рейтинг рівня: від зібраних лайків віднімаються дизлайки та коефіцієнт часу (відсоток від затрачених хвилин, секунд і мілісекунд) та множиться на множник складності рівня. Це єдині математичні операції, що виконуються в даній програмі.

2.3. Опис використаної архітектури та шаблонів проектування

Створення гри починають з того, що придумують сюжет і ідею гри, яку хочуть створити. Створення гри процес виснажливий і вимагає дуже багато часу. Без певних знань створити гру не вийде. Необхідно знати хоча б ази мов програмування, скриптових мов, моделювання.

Далі потрібно вибрати формат створення гри – 2D або 3D. Легше розробити 2D, ніж 3D: вони не навантажують комп'ютер, а потрібну кількість програм, необхідних для створення гри, зводиться до мінімуму. Але навіть для створення 2D-ігор потрібно добре вміти малювати. Можна також користуватися вже готовими заготовками локацій, персонажів і т.д.

Одним з плюсів 3D-ігор можна назвати красу і видовищність. Буде потрібно знання різних мов програмування. Це найскладніша частина створення 3D-гри.

Існують спеціальні конструктори для створення ігор. З готових деталей, які даються в конструкторі, поступово створюється гра. Вони підходять як для 3D-ігор, так і для 2D-ігор. Якщо не вистачає готових деталей, то можна додати свої і користуватися ними. Щоб змусити щось рухатися, потрібно буде привласнювати об'єктам дії, використовуючи готові логічні операції. При нестачі стандартних дій на допомогу прийдуть скриптові мови. Існують конструктори, які включають в себе загальні мови програмування, вони більш функціональні, але в їх роботу складніше вникнути. Конструктори зазвичай розбиті по жанрам, але є і загальні, які підходять для створення ігор різних жанрів.

Створення комп'ютерної гри – комплексний процес, найважливіша частина якого це проектування. Необхідно попередньо створити план гри, сценарій, сюжет, вибрати потрібну мову програмування, продумати можливість технічної реалізації заданого. Немає єдиного способу написання гри, оскільки її створення – творчий процес.

Умовно процес створення гри можна розбити на кілька етапів:

– опрацювання тематики і жанру майбутньої гри. Спочатку необхідно створити ідею і оформити її. Створити майбутніх героїв, продумати сюжет, кожну його складову. Звести всі зібрані дані в один документ дизайну проекту, де буде міститися інформація про сюжет;

– вибір мови програмування, на якій буде реалізація проекту. Залежно від масштабів гри, слід враховувати особливості мови. Наприклад, безліч сучасних ігор пишеться на C ++, проте існує безліч інших мов програмування, які були до написання;

– вибір движка, на базі якого буде будуватися ігровий проект. Движок є керуючою системою, яка відповідає за відображення графічних елементів, визначення функцій, управління звуком і т.п. Він безпосередньо пов'язаний з графічним інтерфейсом програмування додатків (API). Якщо буде використовуватися готовий движок, слід задуматися про бюджет проекту, оскільки придбання програмного коду, 3D, графічних і аудіо редакторів може обійтися в серйозну суму;

– для написання серйозних проектів необхідно набрати собі команду, яка буде складатися з 3D-модельєра, графічного редактора, дизайнера, верстальника і музиканта. Кількість необхідних профільованих фахівців залежить від складності проекту.

– створивши план, вибравши движок, можна приступати до технічної реалізації задуманого. Робота розбивається на етапи, гра пишеться поступово, реалізуючи спочатку основний функціонал, а потім створюючи все нові можливості. Код повинен бути максимально ефективним.

Ігрові движки надають засоби розробки, які можуть бути використані програмістами, щоб спростити їх роботу. Надають інструменти та функціональні можливості для розробки гри.

2.4. Опис використаних технологій та мов програмування

Unity3d – це сучасна платформа для створення ігор та додатків в реальному часі, розроблена Unity Technologies. За допомогою платформи можна розробляти не тільки додатки для комп'ютерів, але і для мобільних пристроїв, ігрових приставок та інших девайсів. Інтерфейс платформи (рис. 2.1).

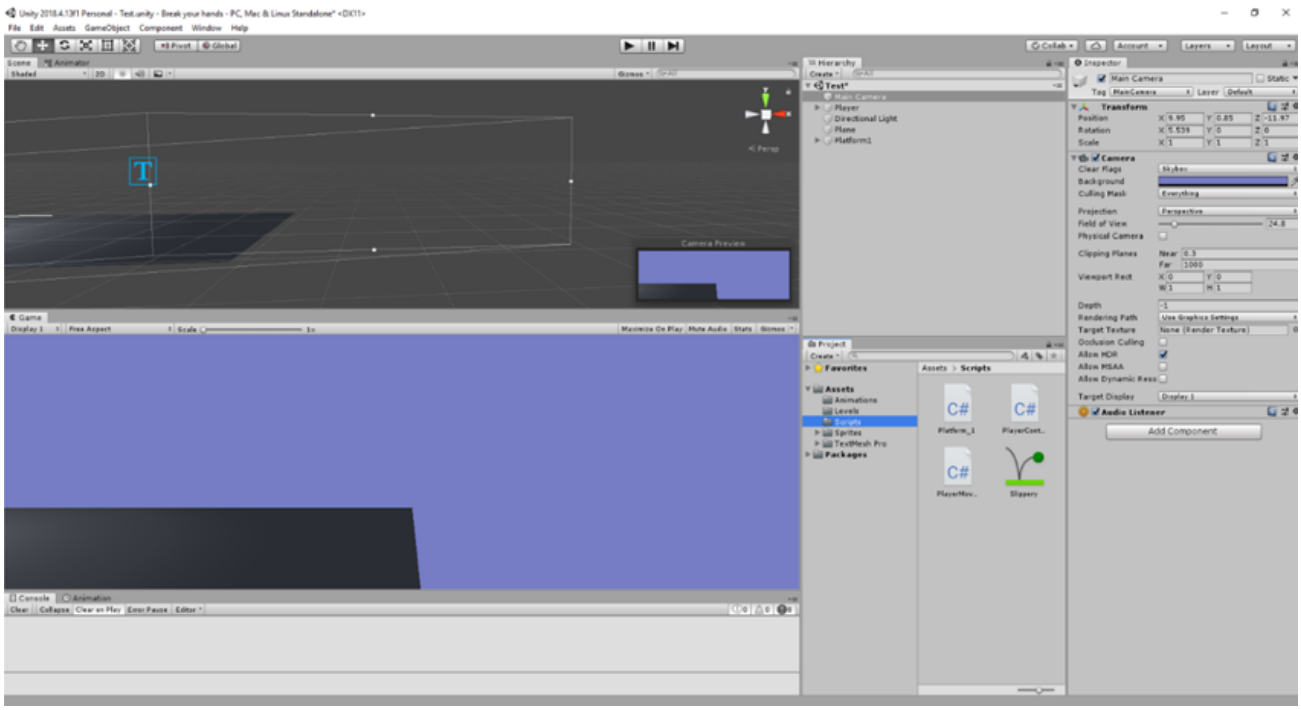

Рис. 2.1. Інтерфейс платформи Unity

Unity дозволяє протестувати свою гру не виходячи з редактора. Підтримує імпорт велику кількість файлових форматів, що дозволяє розробнику гри конструювати самі моделі в більш зручному додатку, а Unity використовувати за прямим призначенням – розробка ігрового продукту. Написання ігрової логіки та програмних сценаріїв (скриптів) здійснюється на C#.

Ігри в Unity будуються на основі трьох фундаментальних блоків: об'єкти GameObject, компоненти та змінні.

Будь-який об'єкт в Unity є GameObject: персонажі, джерела світла, спецефекти, декорації і все інше. Ігрові об'єкти самі по собі не мають ніякої поведінки. Для того, щоб об'єкт змінював свій стан, йому потрібні атрибути, що зберігаються в компонентах, котрі і визначають поведінку ігрових об'єктів, до яких вони прикріплені. Простий приклад – створення джерела світла, що включає прикріплення компонента Light до GameObject. Таким же чином можна додати компонент Rigidbody до об'єкта, щоб він мав фізичні властивості.

Компоненти мають ряд властивостей або змінних, які можна налаштувати як у вікні Inspector редактора Unity, так і за допомогою скрипта. В наведеному вище прикладі до властивостей джерела світла відносяться дальність, колір та інтенсивність.

Unity надає багато влаштованих компонентів, але можна створити власний для реалізації специфічних алгоритмів. Це можна зробити за допомогою створення користувацького скрипта та прикріплення його до необхідного об’єкта. Кожен скрипт зв'язується з внутрішніми механізмами Unity шляхом реалізації класу, похідного від вбудованого класу MonoBehaviour.

Компоненти на основі скриптів дозволяють запускати ігрові події, перевіряти об'єкт на предмет колізій, застосовувати фізичні властивості, програмувати реакцію на управління користувача і багато іншого [1].

C# – мова програмування, що поєднує об'єктно-орієнтовані і контекстно-орієнтовані концепції, відноситься до сім'ї мов з C-подібним синтаксисом, з них синтаксис найбільш схожий на C++ і Java. Мова має строгу статичну типізацію, підтримує поліморфізм, перевантаження операторів, вказівники на функції-члени класів, атрибути, події, властивості, винятки і коментарі у форматі XML.

Розробка сучасних додатків все більше тяжіє до створення програмних компонентів у формі автономних і самописних пакетів, що реалізують окремі функціональні можливості. Важлива особливість таких компонентів – це модель програмування на основі властивостей, методів і подій. Кожен компонент має атрибути, які надають декларативні відомості про компоненти, а

також вбудовані елементи документації. C# надає мовні конструкції, які безпосередньо підтримують таку концепцію роботи.

У C# всі типи даних, включаючи типи-примітиви, такі як int і double, успадковують від одного кореневого типу object. Таким чином, всі типи використовують загальний набір операцій і значення будь-якого типу можна зберігати, передавати і обробляти схожим чином.

Крім того, C# підтримує призначені для користувача посилальні типи і типи значень, дозволяючи як динамічно виділяти пам'ять для об'єктів, так і зберігати спрощені структури в стек.

Мова програмування забороняє звернення до змінних, що не були ініційовані, що виключає можливість виконання безконтрольного приведення типів або виходу за межі певного масиву даних.

Для роботи додатків на C# необхідно встановити і налаштувати платформу NET Framework. Платформа вбудована в інсталяційний па-кет Windows, при необхідності її також можна скачати та інсталювати окремо. Існують також версії для Linux і MAC.

В рамках платформи до обробки виконуваного коду підключається середа CLR – єдиний об'єднаний набір бібліотек і класів, який був розроблений Майкрософт і є реалізацією світового стандарту Common Language Infrastructure (CLI).

Основний механізм CLR має вигляд бібліотеки під назвою mscoree.dll (називається загальним механізмом виконання виконуваного коду об'єктів – Common Object Runtime Execution Engine). При додаванні посилання на збірку для її використання, завантаження бібліотеки mscoree.dll здійснюється автоматично і потім, в свою чергу, призводить до завантаження необхідної збірки в пам'ять.

Механізм виконуючого середовища відповідає за виконання цілого ряду завдань. Перш за все, що найбільш важливо, він відповідає за визначення місця розташування збірки і виявлення запитуваного типу в двійковому файлі за

рахунок зчитування збережених метаданих. Потім він розміщує тип в пам'яті, перетворює CIL-код у відповідні платформі інструкції, ініціює будь-які необхідні перевірки на предмет безпеки і після цього, нарешті, безпосередньо виконує сам запитуваний програмний код. В результаті код C # вважається керованим, тобто він компілюється в двійковий вид на призначеному для користувача пристрої з урахуванням особливостей встановленої системи [2].

Microsoft Visual Studio – це повнофункціональне інтегроване середовище розробки (IDE) з підтримкою популярних мов програмування, таких як C, C ++, VB.NET, C #, F #, JavaScript, Python.

За допомогою інтеграції Visual Studio в Unity можна редагувати створені скрипти для об’єктів GameObject, при збереженні скрипта він оновлюється в платформі Unity. При цьому компілятор середи перевіряє код на помилки в реальному часі без необхідності переключатися на платформу. Інтерфейс середи (рис. 2.2).

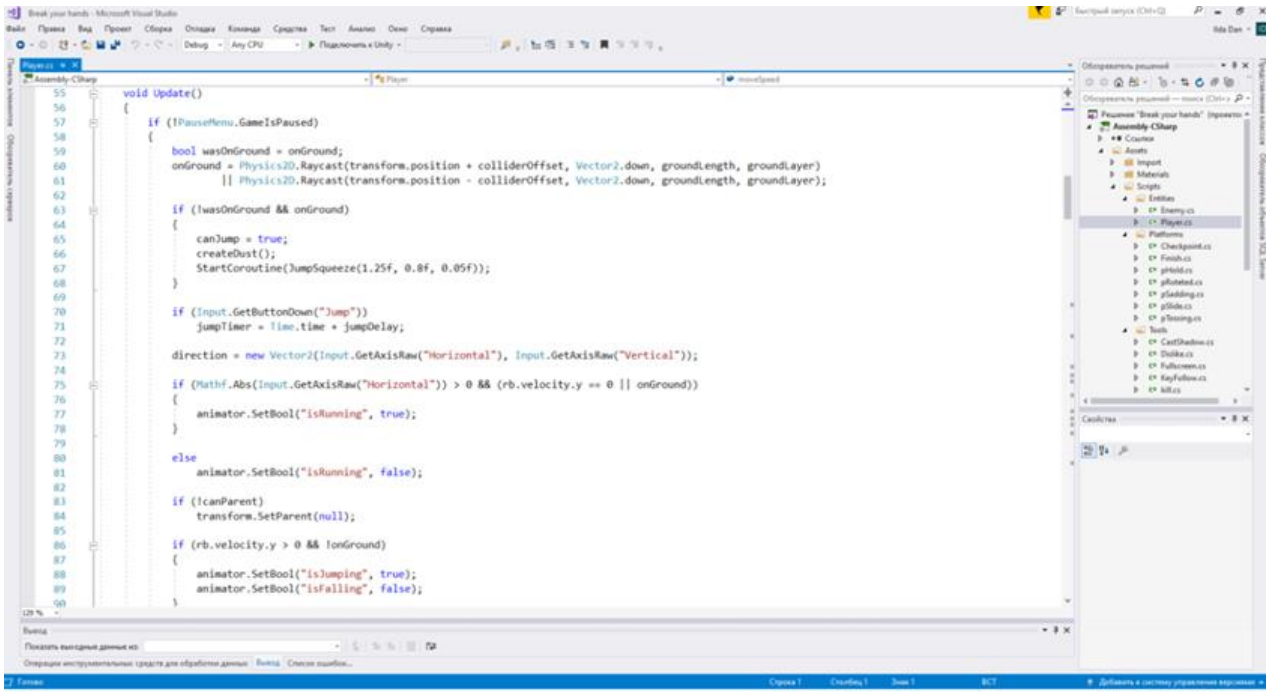

Рис. 2.2. Інтерфейс середи Visual Studio

Існує три випуски Visual Studio 2019: Community, Professional і Enterprise.

У MS Visual Studio кожне окреме застосування є рішенням (solution), що складається з одного чи декількох проектів (project). Одночасно можна відкрити тільки одне рішення (з розширенням. sln), при роботі над кількома рішеннями одночасно можна запустити декілька вікон програми.

Visual Studio надає шаблони для проектів найбільш поширених типів. Використання проектів і їх шаблонів дозволяє користувачеві зосередитися на реалізації окремої функції, в той час, як проект буде виконувати загальне управління та завдання побудови.

Функціональність Visual Studio охоплює всі етапи розробки програмного забезпечення, надаючи сучасні інструменти для написання коду, проектування графічних інтерфейсів, збірки, налагодження і тестування програм. Можливості Visual Studio можуть бути доповнені шляхом підключення необхідних розширень.

Рефакторинг включає в себе такі операції, як інтелектуальне перейменування змінних, витягування однієї або декількох рядків коду в новий метод, зміна порядку параметрів методів і багато іншого.

Редактор коду Visual Studio підтримує підсвічування синтаксису, вставку фрагментів коду, відображення структури і пов'язаних функцій. Істотно прискорити роботу допомагає технологія IntelliSense – автозавершення коду під час введення.

Вбудований відладчик Visual Studio використовується для пошуку і виправлення помилок у вихідному коді, в тому числі на низькому апаратному рівні. Інструменти діагностики дозволяють оцінити якість коду з точки зору продуктивності і використання пам'яті.

Visual Studio надає комплекс інструментів для автоматизації тестування додатків в частині перевірки роботи інтерфейсів, модульного і навантажувального тестування.

Система налагодження Visual Studio дозволяє переглядати код з кроком в одну інструкцію, перевіряючи значення змінних. Можна задати точки зупину,

які зупиняють виконання коду на певному рядку. Таким чином можна побачити як значення змінної змінюється по мірі виконання коду [3].

Blender – це безкоштовне програмне забезпечення для створення і редагування тривимірної графіки.

В роботі використовується для створення моделей оточення гри, які в подальшому будуть розміщені на ігрових рівнях за допомогою інструментів проектування платформи Unity.

З огляду на кросплатформеність, відкритий вихідний код, доступность і функціональність пакет отримав заслужену популярність не тільки серед новачків, а й серед професіоналів. По мірі розвитку програми її вибирають в якості робочого інструменту для все більш складних проектів. По суті, цей додаток практично не поступається за кількістю можливостей і функціоналу більш просунутим і дорогим пакетам 3D графіки. Інтерфейс програми (рис. 1.3).

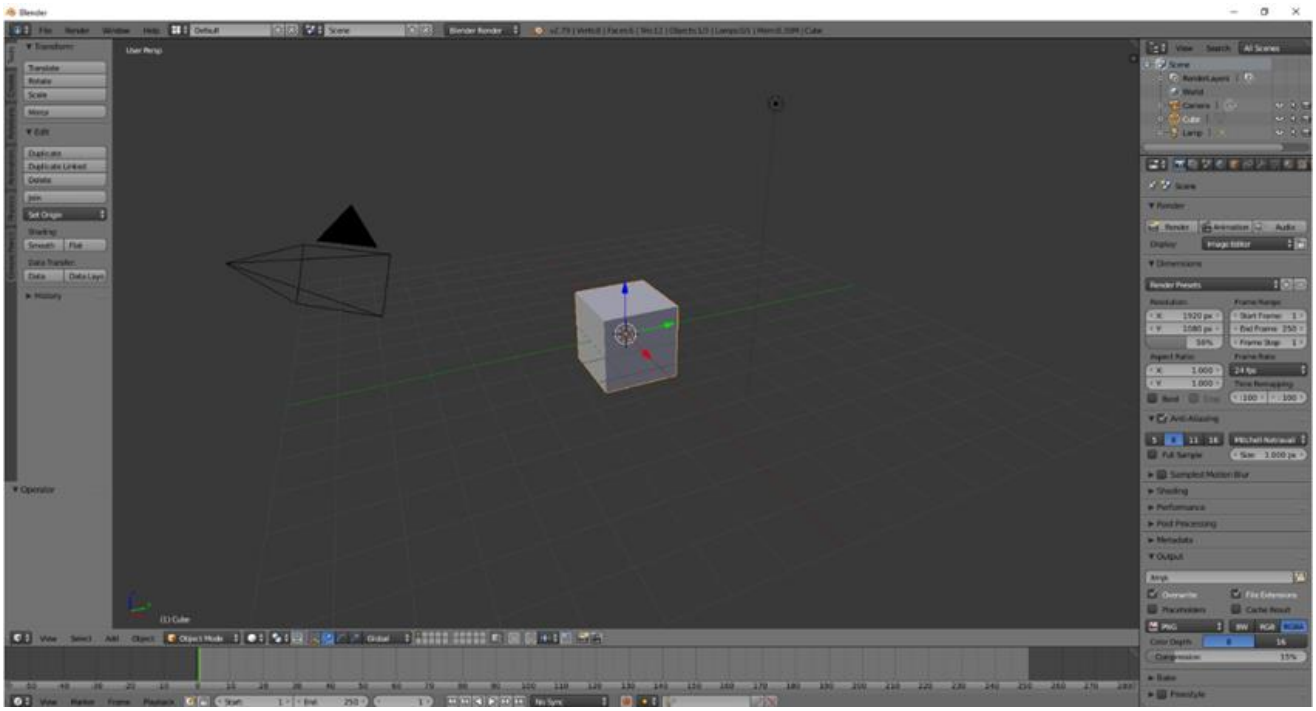

Рис. 2.3. Вікно програми Blender

Blender позиціонується як додаток для створення і редагування

тривимірної графіки, візуалізації, анімації, створення комп'ютерних ігор і навіть скульптинга. Аналогічні програми вимагають багато ресурсів від комп’ютера, але інстальований Blender займає близько 300 мегабайт і оптимізований для слабких ПК. Додатковою перевагою є стабільний і стрімкий розвиток пакета завдяки професійній команді розробників.

Як і у всіх програмах моделювання, користувач працює в свого роду сцені, або в’юпорті. Тут 3D модель безпосередньо створюється і редагується, а також обертається, переміщується, масштабується, тощо. Також тут відображаються всі зміни, пов'язані з процесами анімації, текстурування та візуалізації.

3D моделювання представлено практично всіма існуючими способами створення і роботи з об'ємними моделями. Доступно проектування об'єктів на основі примітивів, полігонів, NURBS-кривих, кривих Безьє, метасфер, булевих операцій, Subdivision Surface і базових інструментів для скульптінга. Як і в 3Ds Max, програма пропонує велику кількість різних модифікаторів, які застосовуються до моделі.

Для анімації у розпорядженні користувача такі інструменти, як ріггінг (скелетна анімація), інверсна кінематика, сіткова деформація, обмежувачі, ключові кадри, редагування вагових коефіцієнтів вершин, тощо. Відмінно реалізована динаміка твердих і м'яких тіл, а також анімація частиць.

Програма дозволяє накладати кілька текстур на один об'єкт, і оснащена рядом інструментів для текстурування, включаючи UV-маппінг і часткове настроювання текстур. Ряд налаштовуємих шейдерів додає гнучкості в роботі з матеріалами.

Програма для 3D моделювання надає можливість створювати начерки різними типами кистей прямо у вікні програми. Поточне призначення такої функції – допомога у створенні 2D анімації, для чого ця функція також оснащена можливістю гнучкого налаштування, зокрема, роботи з шарами.

Пакет оснащений декількома вбудованими інструментами візуалізації, а

також підтримує інтеграцію з різними зовнішніми рендерерами. Також в Blender є вбудований відео-редактор, що не настільки потужний, як спеціалізоване ПЗ для цих цілей, але вельми непоганий [4].

Відеогра повинна повноцінно функціонувати в операційній системі Windows.

Основні характеристики Windows:

– графічний інтерфейс: всі елементи, які користувач бачить на ек-рані (вікна, кнопки, смуги скролінгу), знаходяться в файлах ОС. Існує безліч бібліотек, які містять функції управління цими елементами, що дозволяє програмісту не відволікатися на створення інтерфейсу програми і стандартизує зовнішній вигляд додатків;

– багатозадачність: Windows підтримує одночасне виконання декількох програм на рівнях процесів (програм) і потоків (всередині програмний паралелізм), забезпечення яких відбувається за участю апаратного рівня. ОС містить велику кількість функцій, що забезпечують управління паралельно виконуваними програмами, синхронізацію, поділ адресного простору пам'яті між додатками тощо;

– апаратно-незалежне програмування: використання функцій операційної системи в прикладній програмі дозволяє програмісту не піклуватися про сумісність програми з апаратурою. ОС сама (за допомогою драйверів) виконує узгодження введення-виведення. Таким чином, програма, створена на одній конфігурації буде працювати на інший;

– події і механізм повідомлень: в більшості випадків програма починає працювати тільки при спрацьовуванні деякої події в системі (запуск програми, необхідність перемальовування вікна програми, сигнал від таймера тощо). В інший час програма або взагалі не виконується, або виконує невеликий код в фоновому режимі. Більш того, ОС не прямо викликає деяку процедуру програми для відпрацювання реакції на якусь подію, а посилає відповідне повідомлення в програму, і вона сама вирішує, що їй робити;

– віконна середа: частіше всього в кожній програмі є одне або декілька вікон, які мають стандартний вигляд і містять обов'язкові елементи (наприклад, іконку, рядок заголовка, схоже меню, кнопки закриття, мінімізації та максимізації).

Для запуску гри необхідна версія Windows не нижче 7 SP1, але для кращої стабільності необхідно використовувати більш сучасні версії.

2.5. Опис структури програми та алгоритмів її функціонування

2.5.1. Проектування програми

Відеогра має два режими: для одного гравця – на клавіатурі і двох – проходження гри розподіляється між гравцем на клавіатурі та іншим на ігровому контролері (геймпаді). Основною ціллю гри є проходження рівнів, минаючи перешкоди і ворогів, з найвищою кількістю балів, водночас керуючи персонажем та ігровими об’єктами.

Взаємодію користувачів з системою графічно відображено на діаграмі використання (рис. 2.3).

Рис. 2.3. Діаграма використання

Під час проходження рівня гравець збирає лайки та уникає доторкання до дизлайків. Після проходження рівня формується рейтинг рівня: від зібраних лайків віднімаються дизлайки та коефіцієнт часу (відсоток від затрачених хвилин, секунд і мілісекунд) та множиться на множник складності рівня.

Рейтинг кожного рівня додається до спільного рейтингу гравця, який виводиться в головному меню, причому він перераховується при оновлені, тобто якщо при повторному проходженні рівня враховується тільки один, найкращий результат. Рейтинг зберігається на комп’ютер та завантажується при заходженні в гру.

Процес формування рейтингу гравця відображено на рис. 2.4. Він проходить в два етапи:

– по завершенню рівня гра підраховує всі данні проходження (час, лайки, дизлайки та складність рівня) зводячи їх до одного числа, рахунку рівня, що зберігається у файл PlayerPrefs;

– в головному меню формується спільний рейтинг гравця, сумуванням рахунку кожного рівня, причому якщо цей рівень проходиться гравцем не в перший раз, то враховується один, найкращий результат.

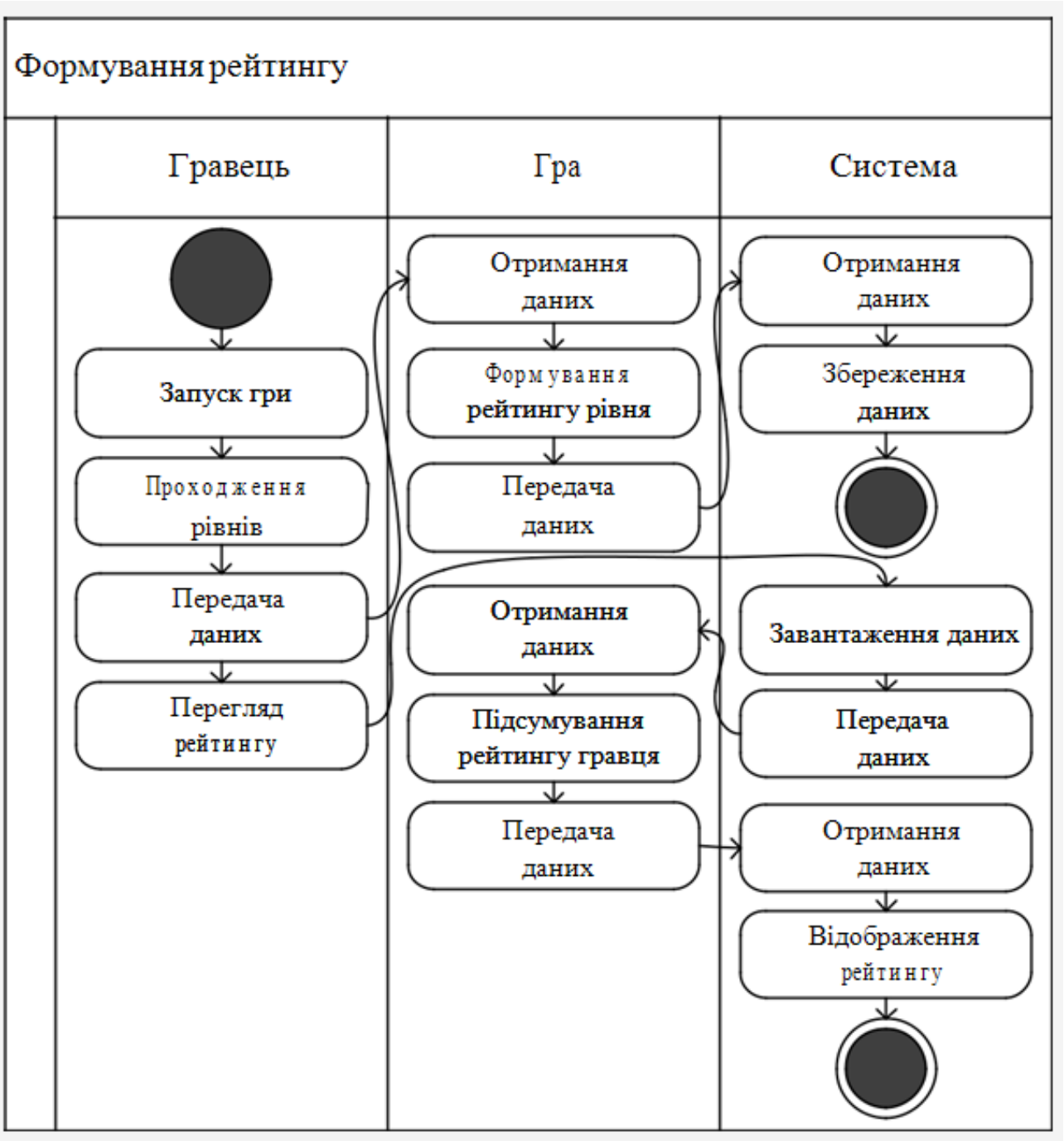

Рис. 2.4. Діаграма діяльності (формування рейтингу гравця)

Управління грою відбувається відповідними клавішами (рис. 2.5).

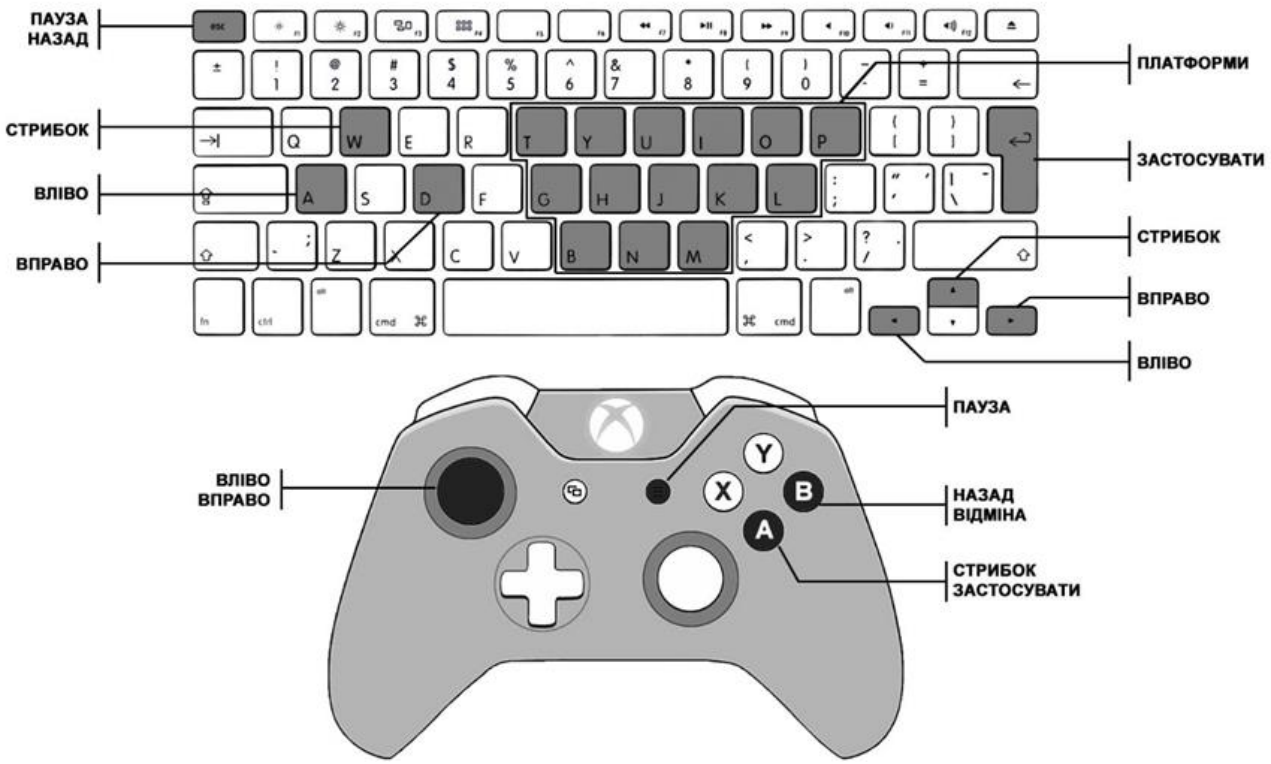

Рис. 2.5. Розкладка управління

2.5.2. Опис основних компонентів

В платформі Unity кожен об’єкт представляє собою GameObject, який зберігає атрибути користувацьких та влаштованих компонентів. Так, наприклад, стандартним компонентом GameObject, який не можливо видалити є «Transform» (рис. 2.6). Він зберігає параметри Position, Rotation, та Scale відповідно позиція, поворот і розмір за трьома осями (X,Y,Z) відносно початку координат.

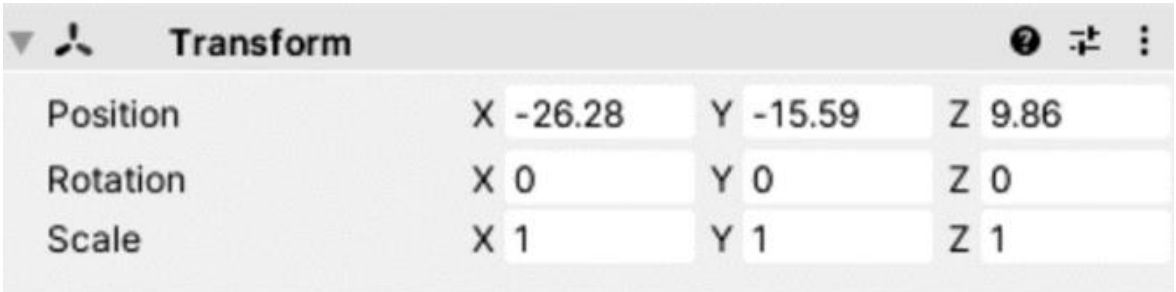

Рис. 2.6. Компонент «Transform»

Щоб відобразити двомірні зображення в тривимірному ігровому просторі використовується компонент «Sprite renderer» (рис. 2.7) Після вказівки у властивості Sprite посилання на імпортоване в проект гри зображення над нам можна проводити різні маніпуляції, такі як обертання, переміщення, зміна кольору тощо.

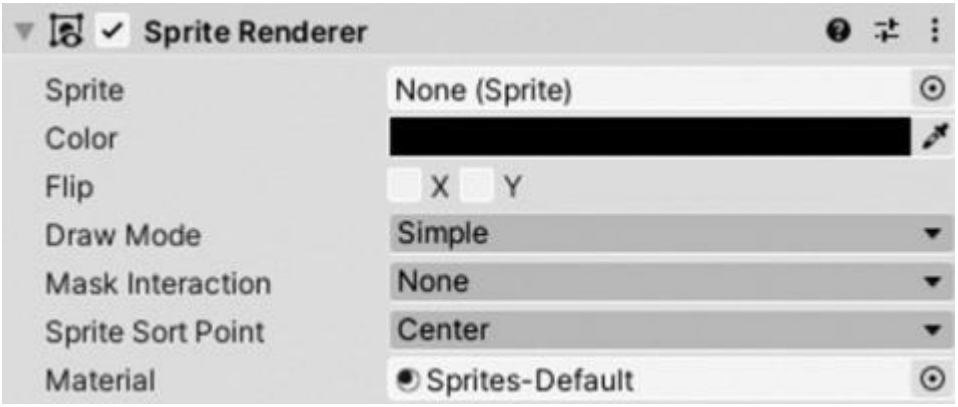

Рис. 2.7. Компонент «Sprite renderer»

Анімацію (ілюзію руху) розглянемо на прикладі персонажа, яким керує гравець, для цього застосовано компонент «Animator» (рис. 2.8), він зв’язує об’єкт з файлами анімацій (.anim), що створюються у вікні «Animation» (рис. 2.9), для анімації бігу послідовно перемикаються зображення (рис. 2.10).

Рис. 2.8. Компонент «Animator»

Рис. 2.9. Вікно «Animation»

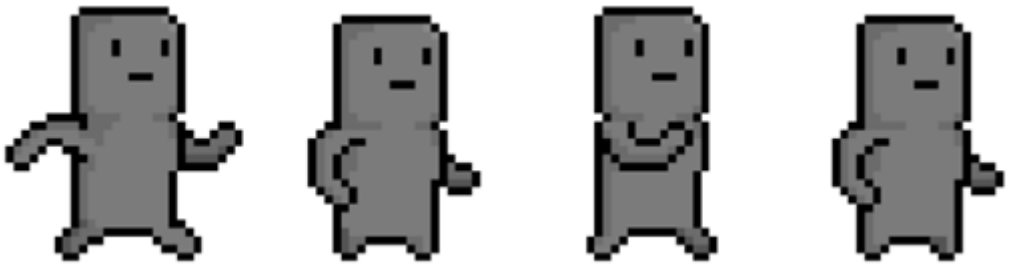

Рис. 2.10. Анімація персонажа «Player_Run»

Кожен файл .anim повинен містити одну дію, таким чином після створення всіх можливих дій об’єкта необхідно додати файли анімації до вікна «Animator» (рис. 2.11) і створити зв’язки між ними та налаштувати тригери переходу до наступної анімації для кожного зв’язка.

Для анімації персонажа створено параметри (рис. 2.12 та тригери для управління анімацією зі скрипта, так як анімація не впливає на переміщення персонажа по рівню.

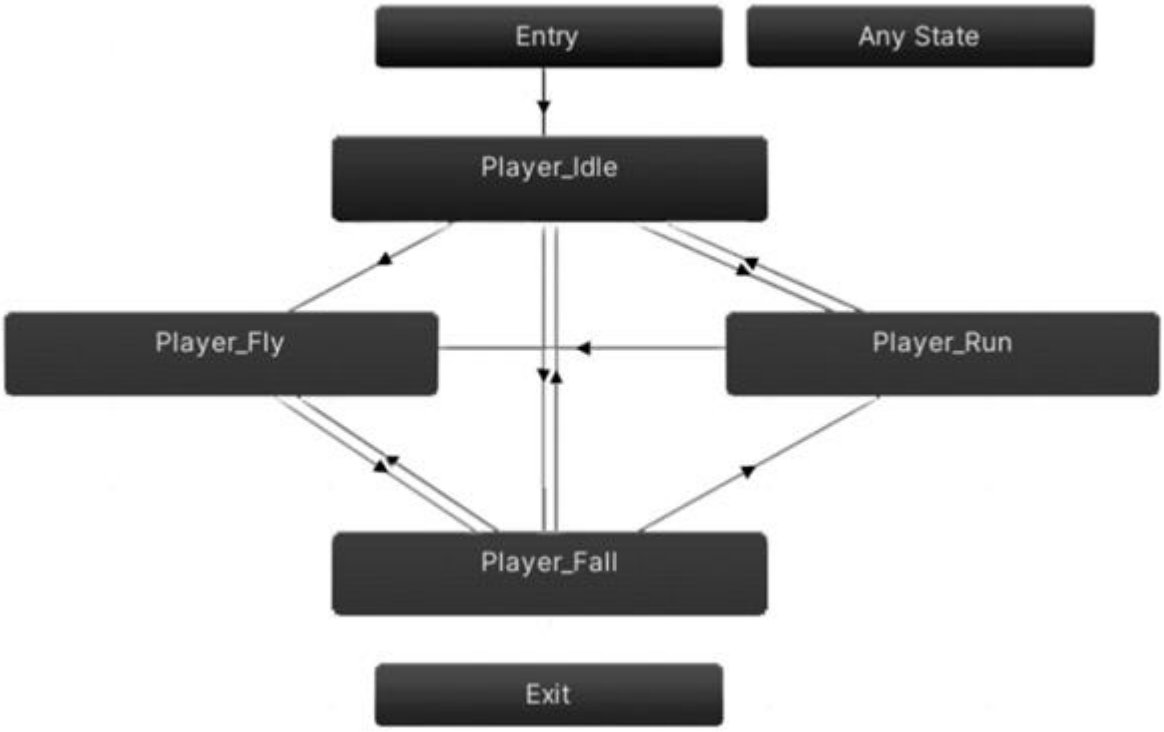

Рис. 2.11. Вікно «Animator»

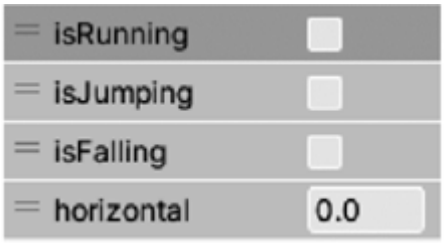

Рис. 2.12. Параметри анімації персонажа

Для того, щоб об’єкт підкорявся законам фізики існує компонент «Rigidbody 2D» (рис. 2.13) в ньому налаштовується маса, лінійне та кутове прискорення, гравітаційний множник та багато іншого. Але за рідкими випадками цей компонент сам по собі даремний, бо не може впливати на інші об’єкти.

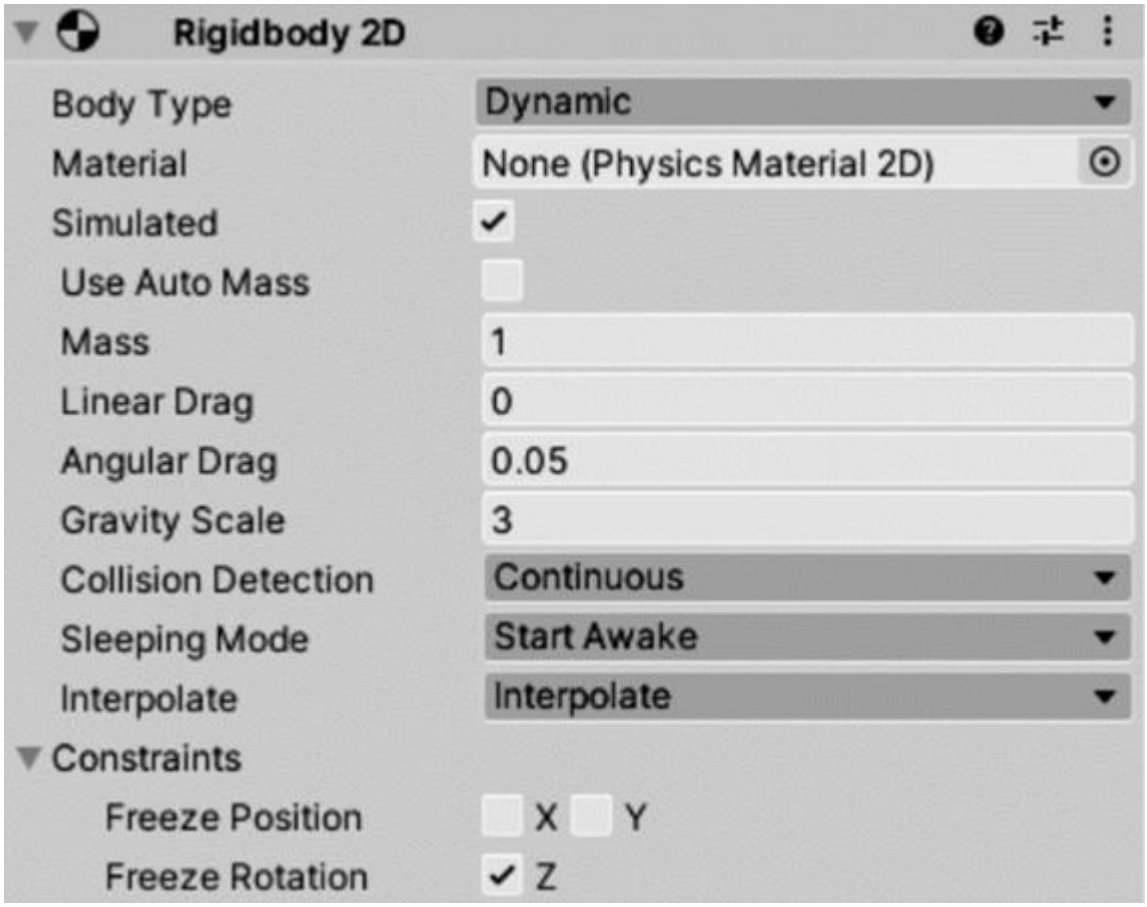

Рис. 2.13. Компонент «Rigidbody 2D»

Група компонентів «Collider 2D» додає колізію, тобто фізичну взаємодію з іншими об’єктами. Розглянемо «Box Collider 2D» (рис. 2.14) – він не дає 1 кубу провалитися через 2 куб (рис. 2.15).

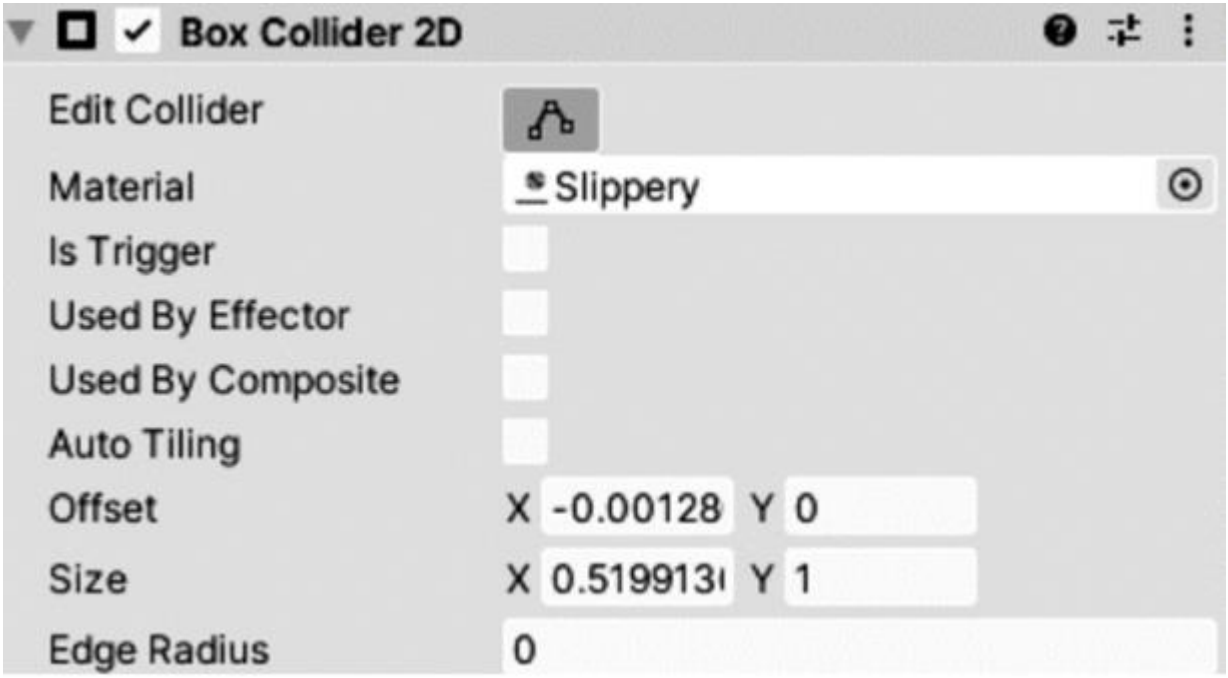

Рис. 2.14. Компонент «Rigidbody 2D»

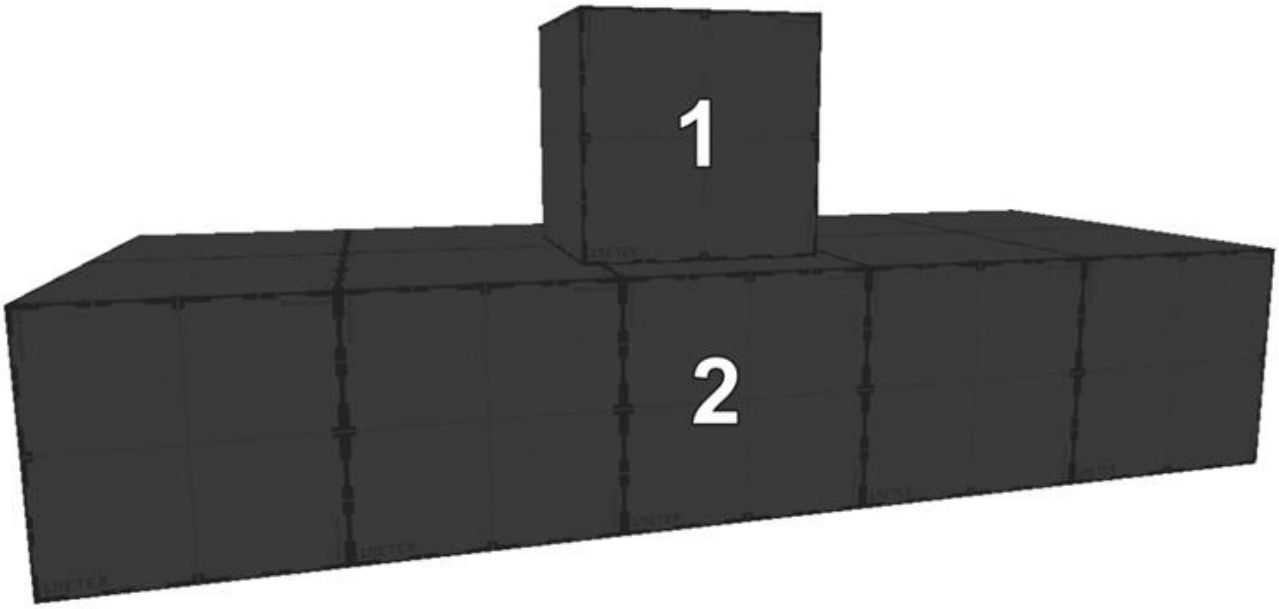

Рис. 2.15. Приклад дії Box Collider 2D

Створений користувачем компонент може керувати властивостями інших компонентів, додавати нові, змінювати ігрову сцену та використовувати всі можливості мови C#.

2.5.3 Реалізація ігрових елементів

2.5.3.1. Створення платформ

Для елемента обертаюча платформа створено наступні об’єкти і компоненти (табл. 2.1) та основний скрипт «pRotated» (табл. 2.2).

Таблиця 2.1

Компоненти обертаючої платформи

Об’єкт	Компонент	Призначення
pRotated	pRotated (Script)	Управління платформою
Pivot	Transform	Точка обертання платформи
Spikes	Mesh renderer	Візуальна модель шипів
	Box Collider 2D	Модель колізії шипів
	Kill (Script)	Вбиває персонажа та ворогів при дотоpканні
Platform	Mesh renderer	Модель колізії платформи
	Box Collider 2D	Фізична модель платформи
Key	TextMeshPro	Відображення кнопки платформи

Таблиця 2.2

Властивості скрипта «pRotated»

Властивість	Тип	Призначення
Smoothness	Float	Швидкість обертання
Angle	Float	Кут обертання
Button	KeyCode	Кнопка обертання
Key	TextMesh Pro	Відображення кнопки платформи

Pivot	Transform	Точка обертання платформи
Objects	Transform	Всі об’єкти платформи, крім Pivot

При натисканні кнопки, що задається у властивості «Button» об’єкти, що знаходяться в групі «Objects» обертається на кут «Angle» навколо точки «Pivot», якщо об’єкти «Spikes» активовані, то вони вбивають при доторканні персонажа, або ворога. Вигляд і дія платформи відображені на рис. 2.16.

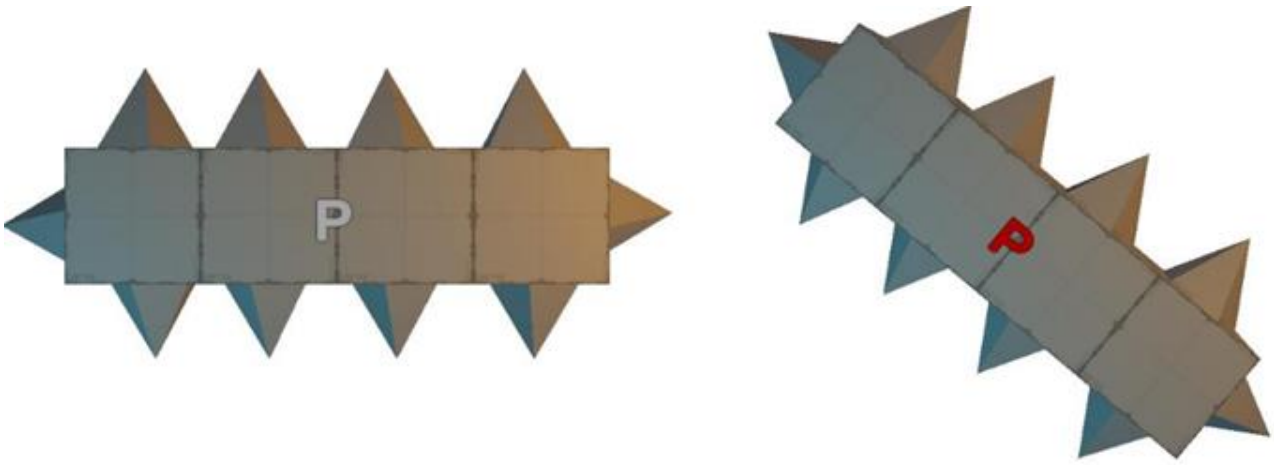

Рис. 2.16. Дія обертаючої платформи

Для механічної платформи створено наступні об’єкти і компоненти (табл. 2.3) та основний скрипт «pTosting» (табл. 2.4).

Таблиця 2.3

Компоненти механічної платформи

Об’єкт	Компонент	Призначення
pTosting	pTosting (Script)	Управління платформою
pTosting	Box Collider 2D	Тригер для об’єктів
Platform	Mesh renderer	Візуальна модель платформи
	Box Collider 2D	Модель колізії платформи
Key	TextMeshPro	Відображення кнопки платформи

Таблиця 2.4

Властивості скрипта «pTosting»

Властивість	Тип	Призначення
Force	Float	Сила підкидання
Button	KeyCode	Кнопка підкидання
Key	TextMesh Pro	Відображення кнопки платформи
Platform	Transform	Координати платформи

При затисканні кнопки, що задається у властивості «Button» об’єкту, що стоїть над платформою надається вектор прискорення вгору «Force». Якщо гравець одночасно натисне кнопку платформи і стрибка, то його підкине не вище визначеної висоти. Вигляд і дія платформи відображені на рис. 2.17.

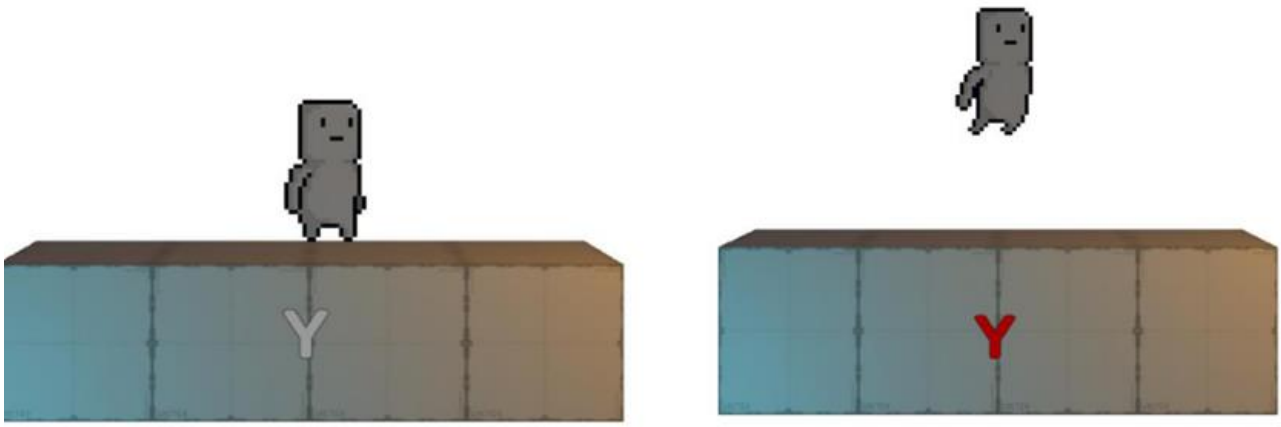

Рис. 2.17. Дія механічної платформи

Для лінійної платформи створено наступні об’єкти і компоненти (табл. 2.5) та основний скрипт «pSlide» (табл. 2.6):

Таблиця 2.5

Компоненти лінійної платформи

Об’єкт	Компонент	Призначення
pSlide	pSlide (Script)	Управління платформою
Pos1	Transform	Позиція 1
Pos2	Transform	Позиція 2
Pivot	Transform	Точка до якої переміщується платформа
Platform	Mesh renderer	Візуальна модель платформи
	Box Collider 2D	Модель колізії платформи
Key	TextMeshPro	Відображення кнопки платформи

Таблиця 2.6

Властивості скрипта «pSlide»

Властивість	Тип	Призначення
Speed	Float	Швидкість переміщення Pivot
Smoothness	Float	Швидкість переміщення платформи
Button	KeyCode	Кнопка підкидання
Key	TextMesh Pro	Відображення кнопки платформи
Platform	Transform	Координати платформи
Pos 1	Transform	Позиція 1
Pos 2	Transform	Позиція 2
Pivot	Transform	Точка до якої переміщується платформа

При затисканні кнопки, що задається у властивості «Button» точка «Pivot» переміщується між позиціями «Pos 1» та «Pos 2» зі швидкістю «Speed». Об’єкт «Platform» плавно переміщується до точки «Pivot». Вигляд і дія платформи відображені на рис. 2.18.

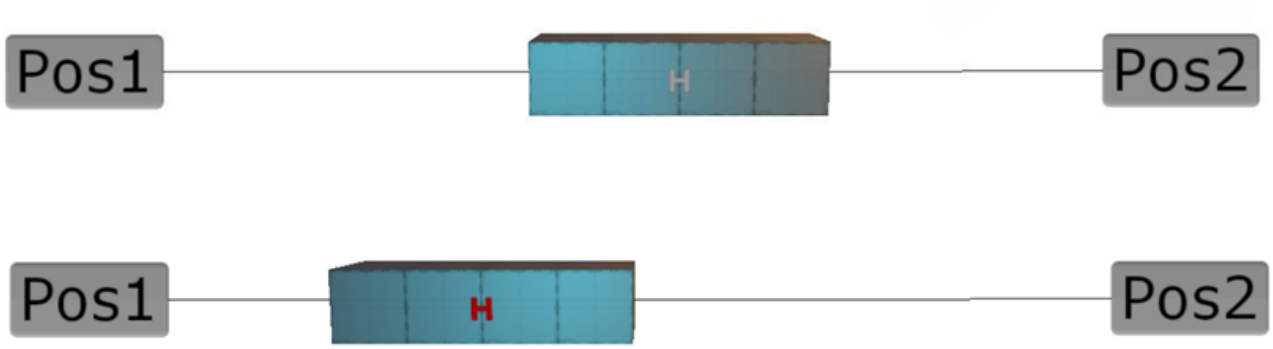

Рис. 2.18. Дія лінійної платформи

Для висувної платформи створено наступні об’єкти і компоненти (табл. 2.7) та основний скрипт «pHold» (табл. 2.8):

Таблиця 2.7

Компоненти висувної платформи

Об’єкт	Компонент	Призначення
pHold	pHold (Script)	Управління платформою
Platform	Mesh renderer	Візуальна модель платформи
	Box Collider 2D	Модель колізії платформи
Key	TextMeshPro	Відображення кнопки платформи

Таблиця 2.8

Властивості скрипта «pHold»

Властивість	Тип	Призначення
Offset	Float	Дальність висування платформи
Smoothness	Float	Швидкість переміщення платформи
Button	KeyCode	Кнопка висування
Key	TextMesh Pro	Відображення кнопки платформи
Platform	Transform	Координати платформи

При затисканні кнопки, що задається у властивості «Button» об’єкт «Platform» переміщується на «Offset» зі швидкістю «Smoothness». Вигляд і дія платформи відображені на рис. 2.19.

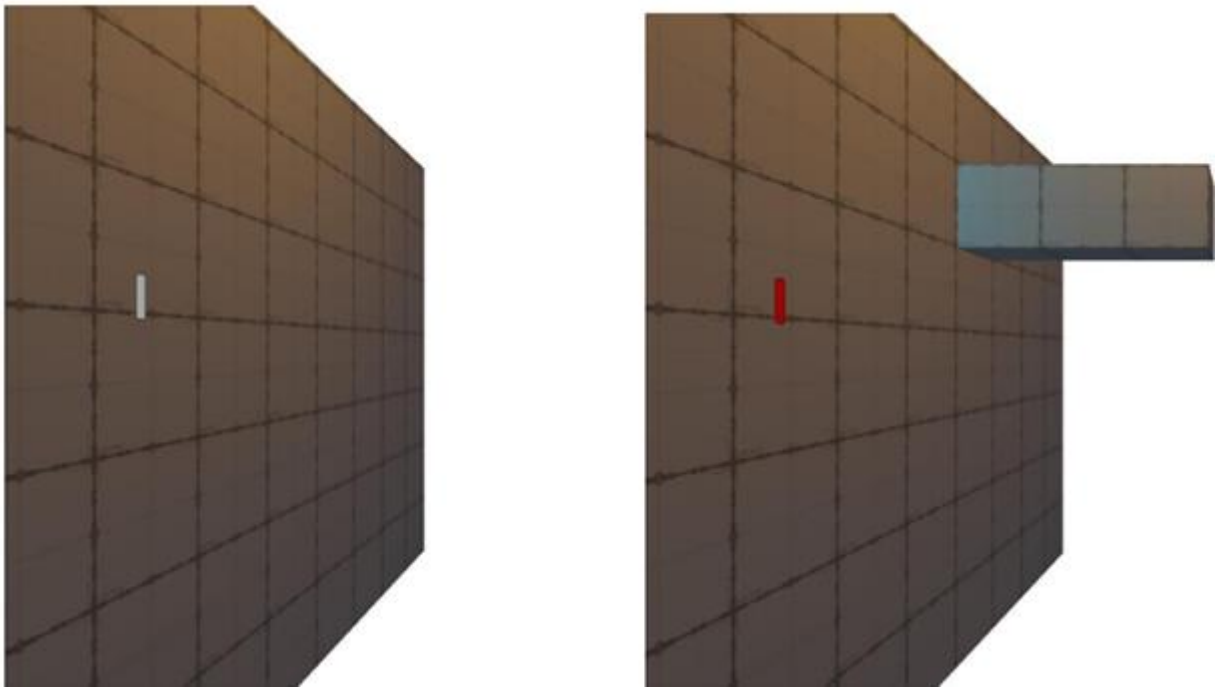

Рис. 2.19. Дія висувної платформи

Для статичної платформи створено наступні об’єкти і компоненти (табл. 2.9) та основний скрипт «pSadding» (табл. 2.10):

Таблиця 2.9

Компоненти висувної платформи

Об’єкт	Компонент	Призначення
pSadding	pSadding (Script)	Управління платформою
	Mesh renderer	Візуальна модель платформи
	Box Collider 2D	Модель колізії платформи
parTrig	parrentTrigger	Коректна прив'язка до платформи
	Box Collider 2D	Тригер прив’язки

Таблиця 2.10

Властивості скрипта «pSadding»

Властивість	Тип	Призначення
Sadding	Float	Сила просідання платформи
Smoothness	Float	Швидкість просідання платформи

Коли гравець стає на платформу або вдаряється об неї знизу вона просідає чи підстрибує на «Sadding» (рис. 2.20).

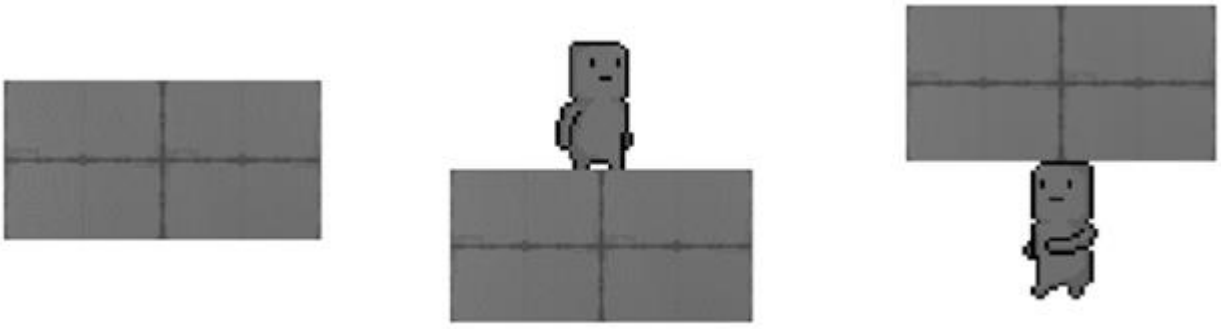

Рис. 2.20. Дія статичної платформи

2.5.3.2. Створення персонажу

Для персонажа було створено об’єкт «Player» і в нього вкладено об’єкт «Dust» для створення частинок пилу під ним. В «Player» вкладено наступні компоненти (табл. 2.11) та основний скрипт «Player» (табл. 2.12).

Таблиця 2.11

Компоненти персонажа

Об’єкт	Компонент	Призначення
Player	Player (Script)	Управління персонажем
	Sprite Renderer	Візуальна модель персонажа
	Box Collider 2D	Модель колізії персонажа
	Rigidbody 2D	Фізика персонажа
	Animator	Анімація персонажа
	Cast Shadow (Script)	Активація тіні для Sprite Renderer
Dust	Particle System	Система частинок для пилу від персонажа

Таблиця 2.12

Властивості скрипта «Player»

Властивість	Тип	Призначення
Move Speed	Float	Швидкість переміщення
Jump Force	Float	Висота стрибка
Jump Delay	Float	Затримка перед наступним стрибком
Max Speed	Float	Максимальна швидкість переміщення
Gravity	Float	Гравітація
FallMultiplayer	Float	Множник падіння
Collider Offset	Vector 3	Точка для початку Reycast
Ground Layer	Layer Mask	Маска шару для Reycast
Ground Length	Float	Довжина Reycast
Dust	Particle System	Система частинок для пилу від персонажа

Для переміщення персонажа використовуються клавіши «A» і «D» відповідно вліво і вправо. При натисканні однієї з клавіш переміщення компоненту «Rigidbody 2D» надається сила «Move Speed» і «Sprite Renderer» розвертається у відповідну сторону.

Якщо натиснути клавішу «W», то персонаж виконає стрибок шля-хом додавання «Jump Force» до «Rigidbody 2D», щоб він не міг стрибати постійно окрім «Jump Delay» (затримки стрибка) додано визначення знаходження персонажа на підлозі.

Щоб визначити чи стоїть персонаж на підлозі використовується стандартний метод Raycast, який створює так званий промінь, котрий повертає значення true при перетині з колізією. Початкова точка задається тримірним вектором «Collider Offset», довжина «Ground Length», а список шарів підлоги в «Ground Layer».

Кожна дія змінює параметри компонента «Animator» щоб програвалася відповідна анімація персонажа – біг, стрибок чи падіння.

Коли персонаж стрибає чи падає на підлогу його локальний розмір трохи витягується по координаті Y і на короткий час з’являються частинки пилу «Dust».

Персонаж має 3 одиниці здоров’я, які відображаються на екрані (рис. 2.21), коли він отримує пошкодження кількість зменшується, персонажа відкидає назад та на долю секунди його колір змінюється на червоний. Коли здоров’я закінчується персонаж помирає і з’являється на останній контрольній точці з відновленим здоров’ям, під час «переродження» екран темніє.

Рис. 2.21. Відображення здоров’я персонажа

2.5.3.3. Створення додаткових об’єктів

Для контрольної точки створено наступні об’єкти і компоненти (табл. 2.13) та основний скрипт «Checkpoint».

Таблиця 2.13

Компоненти висувної платформи

Об’єкт	Компонент	Призначення
Checkpoint	Checkpoint (Script)	Головний скрипт
	Mesh renderer	Візуальна модель контрольної точки
	Box Collider 2D	Тригер контрольної точки

Коли персонаж потрапляє в зону тригера точка, в якій з’являється персонаж після смерті переноситься до координат Checkpoint, а його колір змінюється на зелений. Вигляд і дія Checkpoint відображені на рис. 2.22 (зеленою лінією показаний тригер).

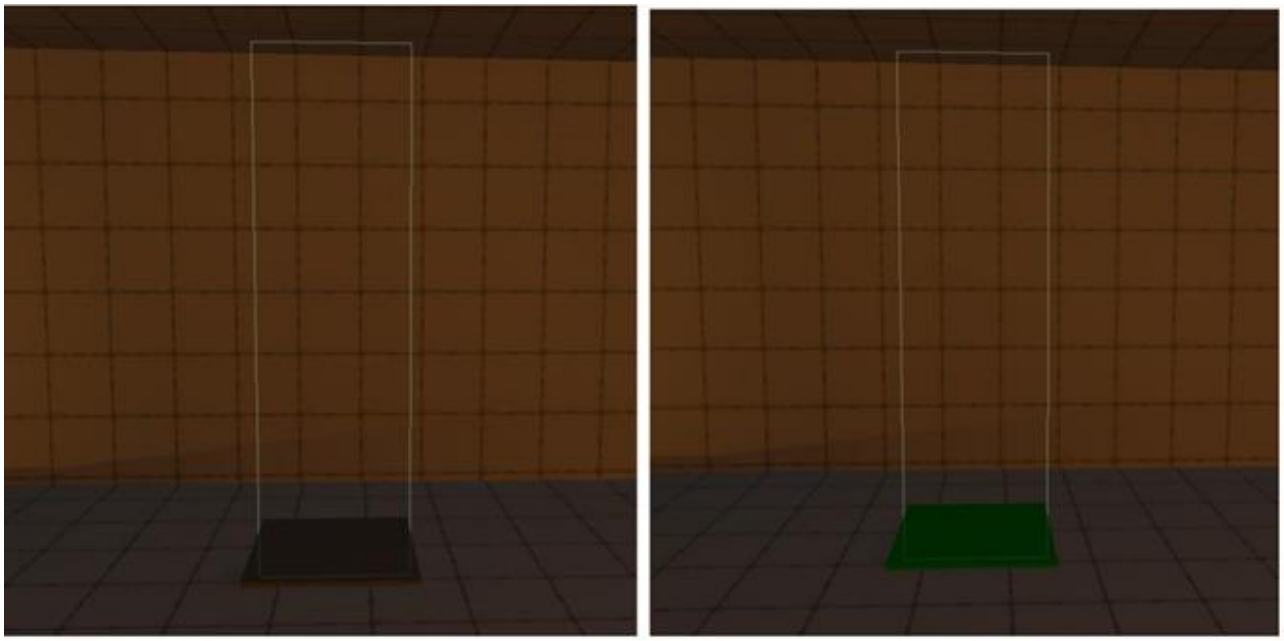

Рис. 2.22. Дія Checkpoint

Для лайка та дизлайка створено наступні об’єкти і компоненти (табл. 2.14) та основні скрипти «Like» та «Dislike».

Таблиця 2.14

Компоненти лайка та дизлайка

Об’єкт	Компонент	Призначення
Like	Like (Script)	Головний скрипт лайка
	Sprite Renderer	Візуальна модель лайка
	Box Collider 2D	Тригер лайка
	Cast Shadow (Script)	Активація тіні для Sprite Renderer
Dislike	Dislike (Script)	Головний скрипт дизлайка
	Sprite Renderer	Візуальна модель дизлайка
	Box Collider 2D	Тригер дизлайка
	Cast Shadow (Script)	Активація тіні для Sprite Renderer

Коли персонаж торкається до тригера лайка (рис. 2.23) чи дизлайка він додається до кількості лайків чи дизлайків гравця на рівні, що відображаються біля секундоміра (рис. 2.24). Секундомір починає відлік при запуску рівня і зупиняється тільки на фініші або в меню паузи.

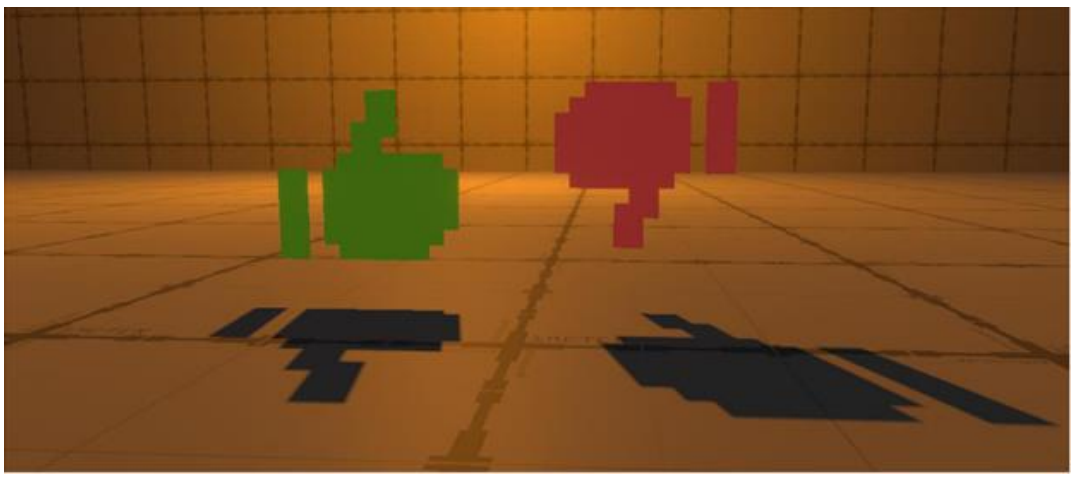

Рис. 2.23. Лайк та дизлайк на рівні

Рис. 2.24. Секундомір

Для патрульного ворога створено наступні об’єкти і компоненти (табл. 2.15) та основний скрипт «Enemy» (табл. 2.16).

Таблиця 2.15

Компоненти лінійної платформи

Об’єкт	Компонент	Призначення
Patrol	Sprite Renderer	Візуальна модель ворога
	Box Collider 2D	Модель колізії ворога
	Box Collider 2D	Тригер для знаходження персонажа
	Rigidbody 2D	Фізика ворога
	Animator	Анімація ворога
	Cast Shadow (Script)	Активація тіні для Sprite Renderer

Таблиця 2.16

Властивості скрипта «Enemy»

Властивість	Тип	Призначення
Move Speed	Float	Швидкість переміщення
Max Speed	Float	Максимальна швидкість
Direction	Int	Направлення переміщення
Health	Int	Кількість здоров’я
Stop	Bool	Перемикач зупинки ворога
Player	Player	Посилання на персонажа
Ground offset	Float	Положення Reycast для землі
Wall offset	Float	Положення Reycast для стінки
Ground Layer	Layer Mask	Маска шару для Reycast

Якщо патрульний ворог знаходиться на землі, то він переміщується в направленні «Direction» зі швидкістю «Move Speed».

За допомогою Reycast променя «Ground» (рис. 2.25) визначається чи є підлога під ворогом. Якщо її немає, то ворог повертається в протилежному напрямку. Промінь знаходиться далі і нижче ворога, щоб він встигнув повернутися, якщо попереду обрив. Для розвороту від стіни є промінь «Wall».

Коли гравець знаходиться в тригері «Box Collider 2D» швидкість ворога збільшується. Робот наносить шкоди в 1 одиницю персонажу при доторканні, але якщо гравець стрибає по ворогу пошкодження отримує ворог, три стрибки вбивають робота.

Рис. 2.25. Патрульний ворог

Основу рівня (рис. 2.26) спроектовано за допомогою пакета «Pro Builder», який додає до Unity основні можливості тривимірного моделювання. Чотирнадцять «Box Collider 2D» використовується для надання основі колізій.

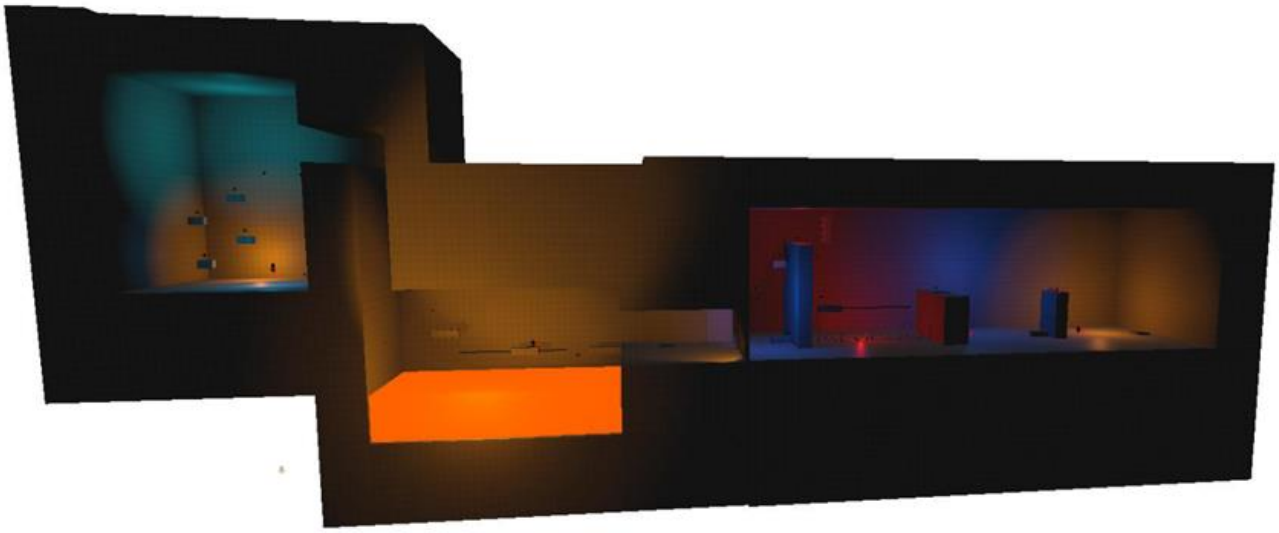

Рис. 2.26. Основа рівня

2.5.4. Опис файлової структури програми

Проект називається «Break your hands».

Основні налаштування гри зберігаються у файлах «browscap.ini» та «config.xml». Ігрові ресурси об’єднуються у файлі «game resources», а стандартні файли Unity у «data.unity3d». Всі бібліотеки, необхідні для запуску

гри знаходяться у папці «Managed». Графічно основні файли гри згруповані на діаграмі компонентів (рис. 2. 27).

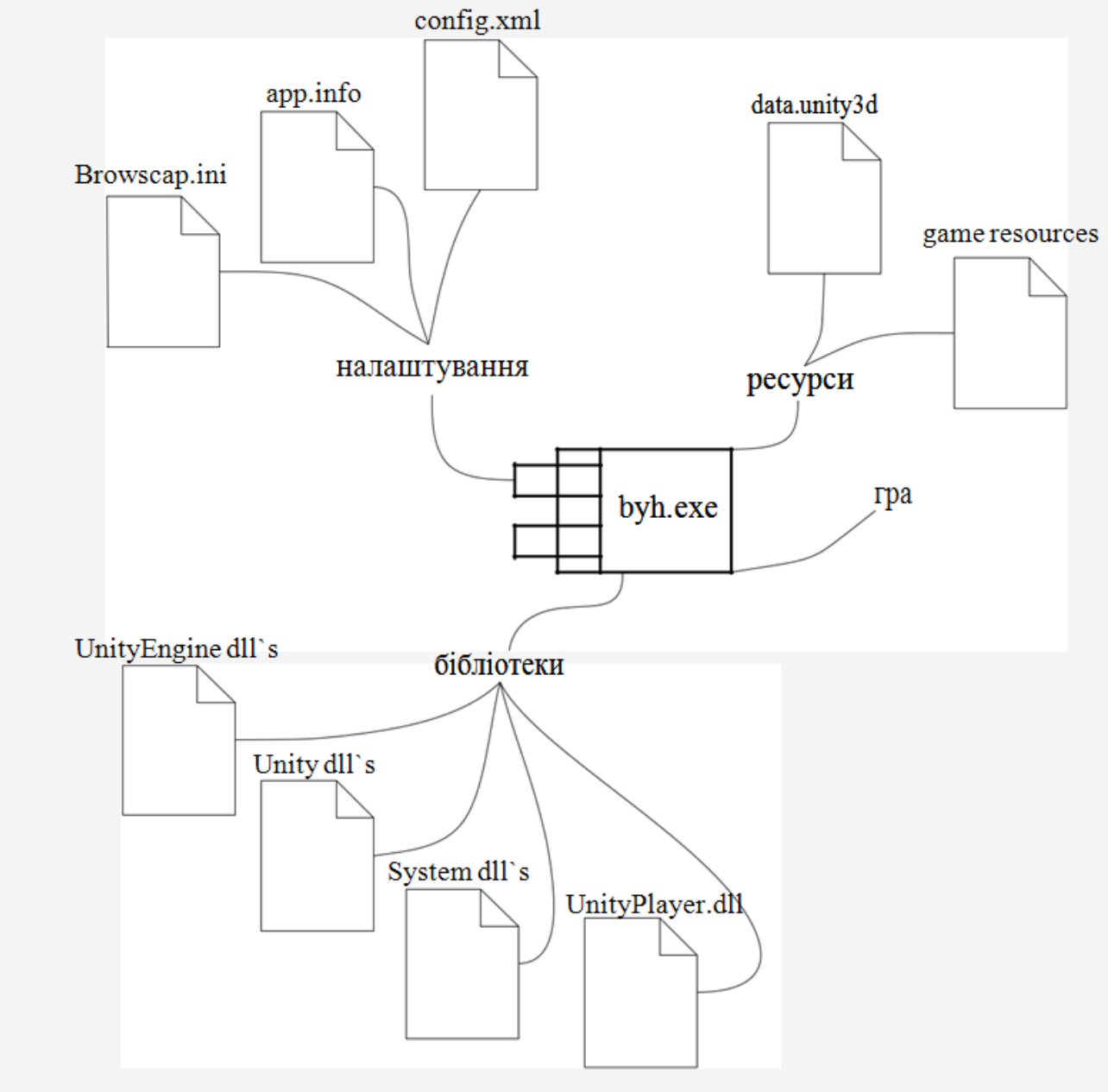

Рис. 2.27. Діаграма компонентів (основні файли додатку)

2.6. Обґрунтування та організація вхідних та вихідних даних програми

Аналізуючи поставлені перед додатком завдання, доцільно виділити наступні групи потоків інформації, що забезпечують правильну роботу програми:

– вхідний потік - дані, що поступають у додаток за допомогою введення їх користувачем.

– вихідний потік - дані, що генеруються в процесі роботи додатку на підставі обробки даних користувачів і передбачених відповідей на певні його дії.

Вхідними даними для роботи додатку є інформація, яка вводиться та відноситься до групи вхідного потоку:

– вибір варіанту гри;

– координати розміщення головного гравця на карті ігрового простру.

– команди для керування ходом ігрового процесу, введені за допомогою пристроїв вводу.

В результаті обробки цих даних, здійснюється виведення певної інформації або реакція додатку, яка відноситься до групи вихідного потоку, що забезпечує безпосередньо результат роботи на певному етапі функціонування додатку.

Вихідними даними є:

– зображення головного персонажу гри та його дій;

– зображення ворожих персонажів;

– зображення мапи ігрового поля;

– зменшення одиниць життя;

– інформація про поточний стан гри та про її закінчення в разі знищення героя.

2.7. Опис роботи розробленого програмного продукту

2.7.1. Використані технічні засоби

Гра не тестувалася на комп'ютерах різної конфігурації, тому наводяться характеристики того комп'ютера, на якому гра створювалася:

– процесор: Mobile DualCore Intel Core i3-350M, 2266 MHz ;
– відеокарта: ATI Mobility Radeon HD 5650;
– пам’ять: 700Mb;
– ОС: Windows 7.

2.7.2. Використані програмні засоби

В програмі використана мова програмування C#. Для створення об’єктів графіки використано інструмент розробки Blender 2.79 та інструмент для розробки двомірних та трьохмірних ігор Unity 3D.

2.7.3. Виклик та завантаження програми

Скомпільований проект гри займає 60 МБ на жорсткому диску і запускається з файлу «Break your hands.exe».

2.7.4. Опис інтерфейсу користувача

Після запуску гри завантажується головне меню (рис. 2.28), з якого можна перейти в налаштування, вийти з гри чи розпочати гру. Меню виконує свої функції вірно.

Рис. 2.28. Головне меню

Кожен рівень спроектовано таким чином, що користувач повинен минаючи перешкоди і ворогів, з найвищою кількістю балів дійти до кінця рівня, водночас керуючи персонажем та ігровими об’єктами.

Після завантаження рівня (рис. 2.29) гравцеві надається управління персонажем (рис. 2.30). Він переміщується коректно, камера не втрачає з виду персонажа і переміщується плавно.

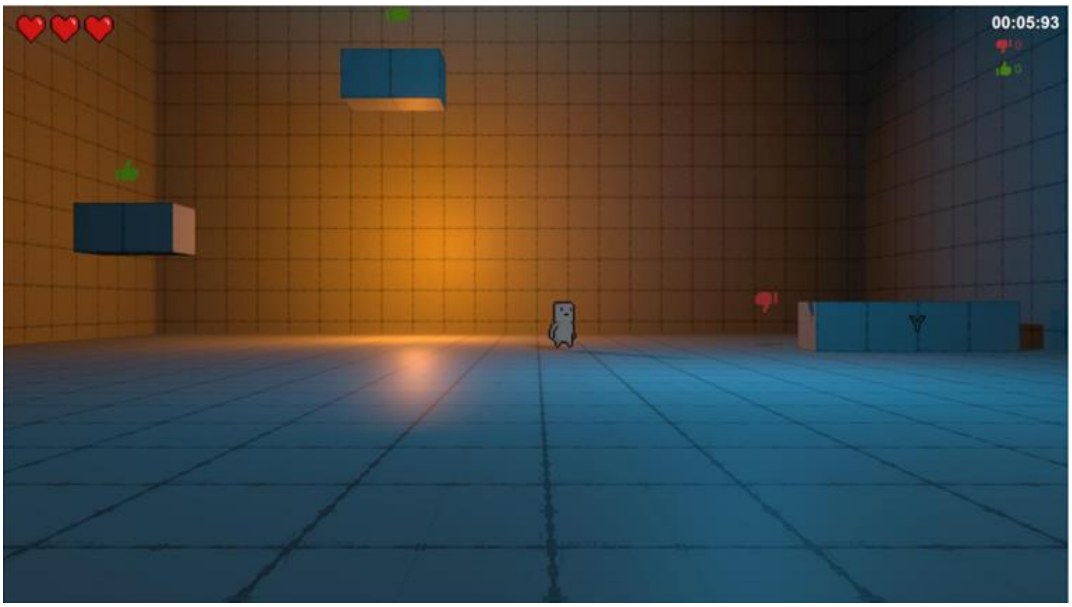

Рис. 2.29. Початок рівня

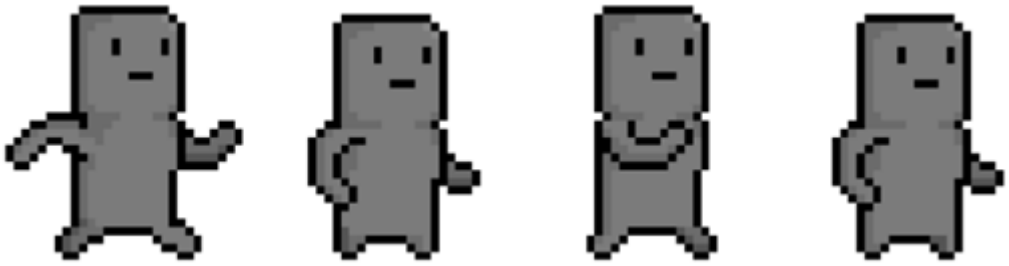

Рис. 2.30. Головний персонаж

На ігровому полі знаходяться лайки та дизлайки, які персонаж повинен збирати (лайки) або минати (дизлайки) (рис. 2.31).

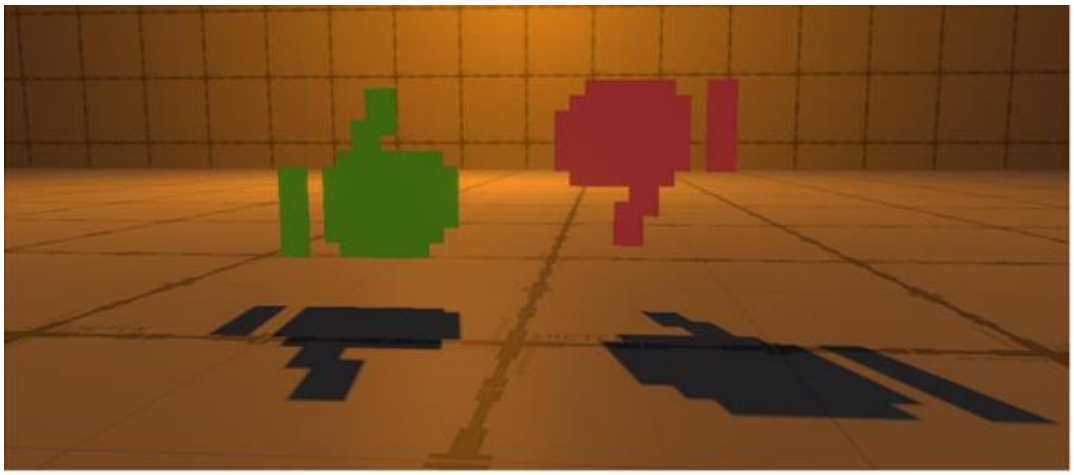

Рис. 2.31. Лайки та дизлайки

Платформи реагують на натиснуті гравцем клавіші (рис. 2.32), персонаж не застряє в них, лайки та дизлайки коректно рахуються.

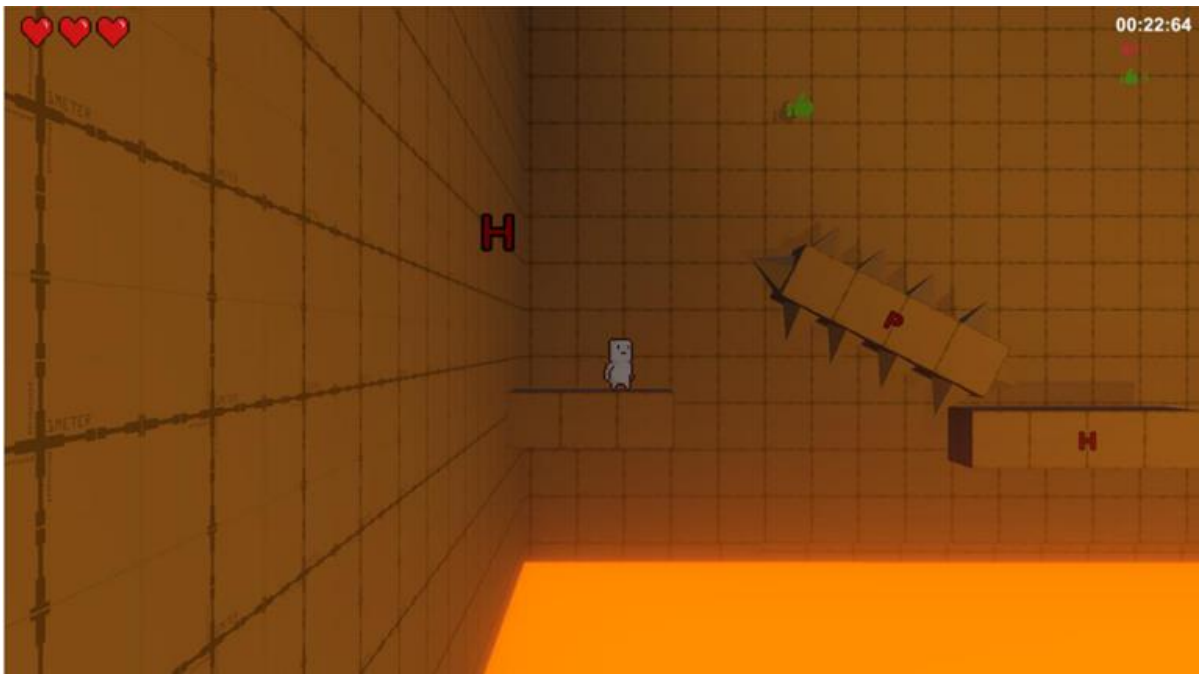

Рис. 2.32. Використання платформ

Меню паузи коректно зупиняє гру і гравець не може керувати персонажем (рис. 2.33).

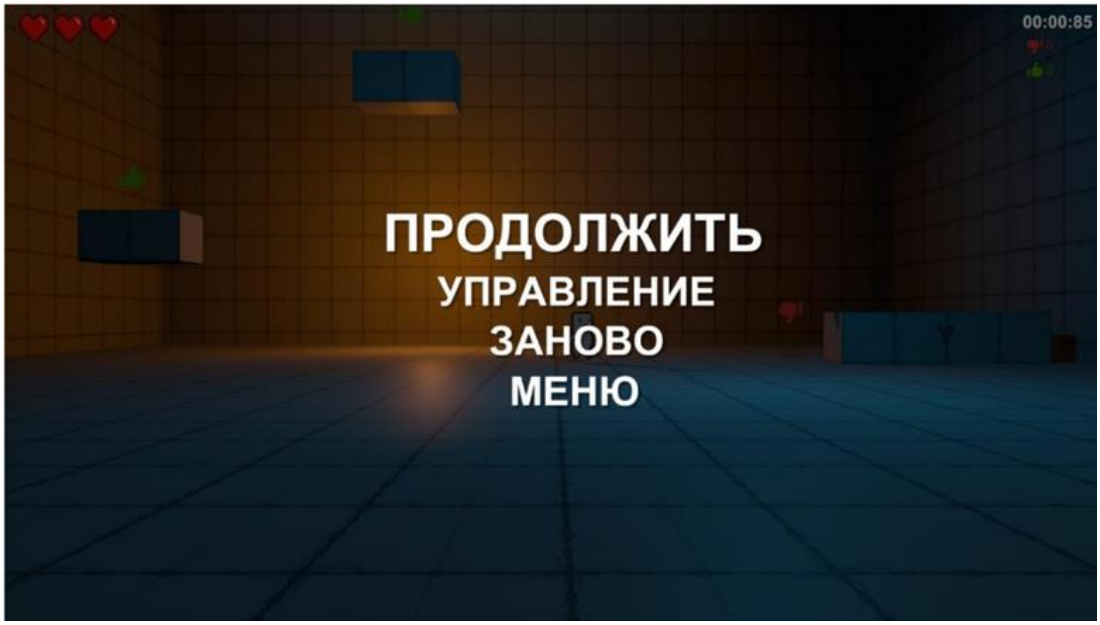

Рис. 2.33. Меню паузи

Робот (рис. 2.34) патрулює платформу, на котрій стоїть. Розроблений

штучний інтелект поводить себе вірно (рис. 2.35). Персонаж, як і патрульний отримують та надають шкоди, а коли здоров’я закінчується помирають.

Рис. 2.34. Робот (противник)

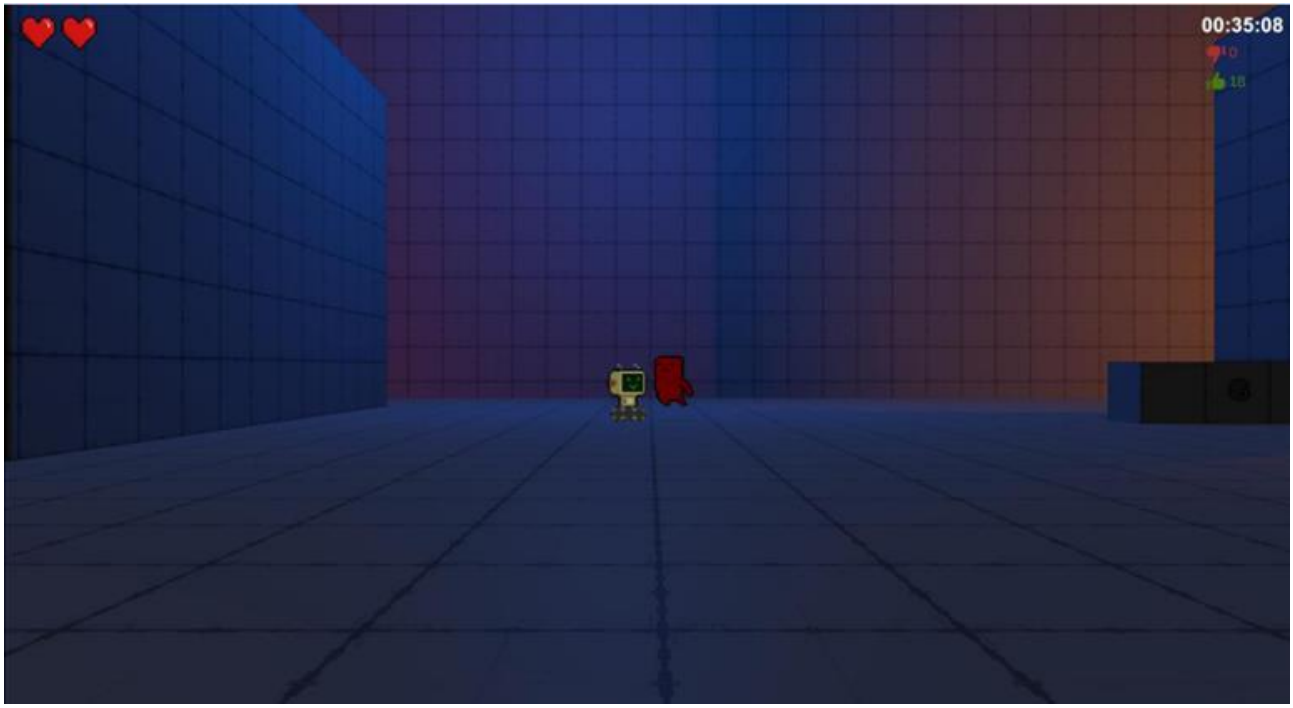

Рис. 2.35. Отримання шкоди

Після проходження рівня рейтинг підраховується та зберігається (рис. 2.36, 2.37).

Рис. 2.36. Завершення рівня

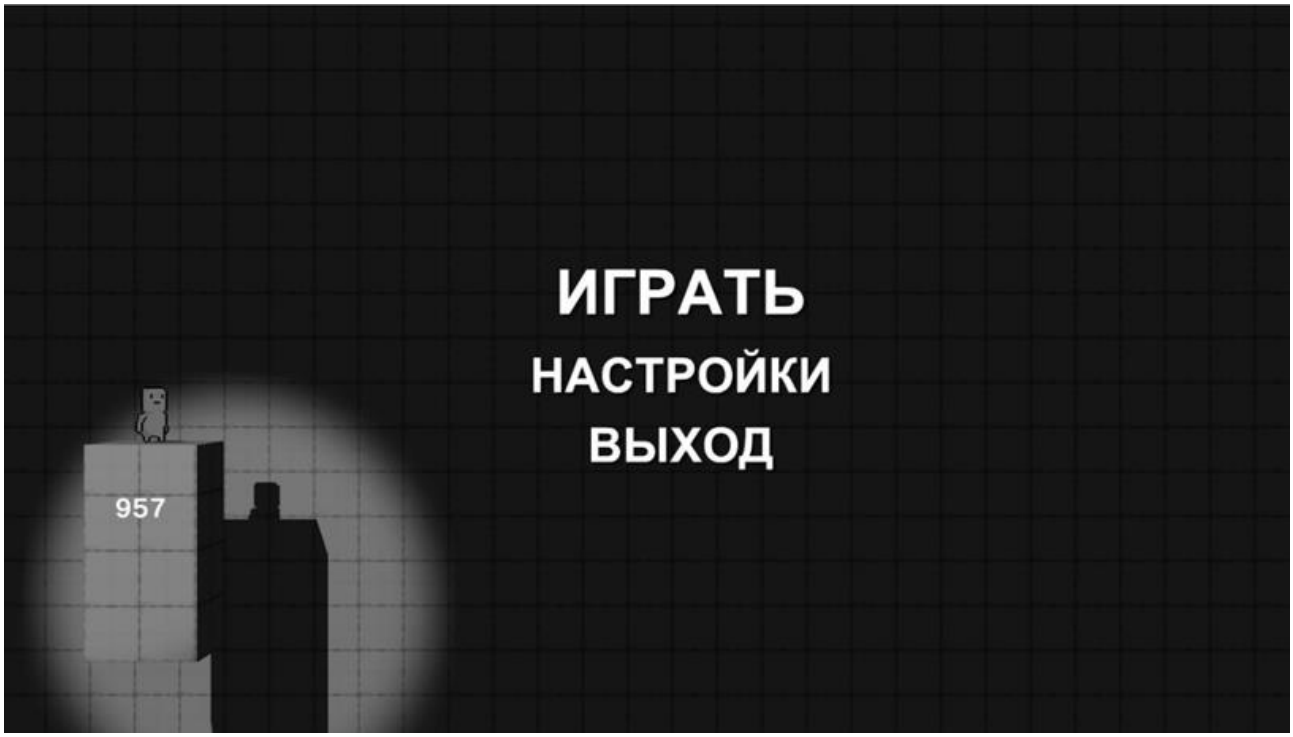

Рис. 2.37. Перевірка збереження рейтингу

Також в меню є можливість перейти до вікна налаштувань, в якому можна залишити налаштування програми за замовченням, або змінити їх на особисті (рис. 2.38).

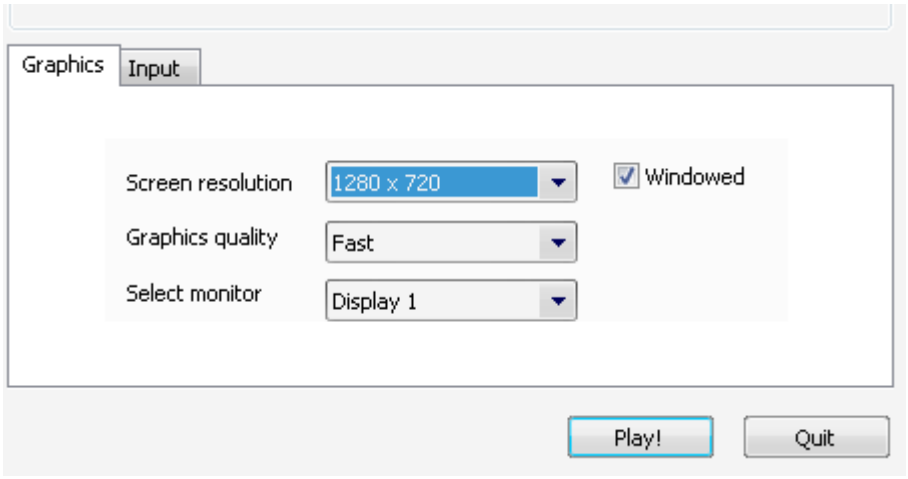

Рис. 2.38. Вікно налаштування параметрів конфігурації

На даному вікні можна змінити налаштування графіки: обрати роздільну здатність екрану, обрати якість графіки (рис. 2.39).

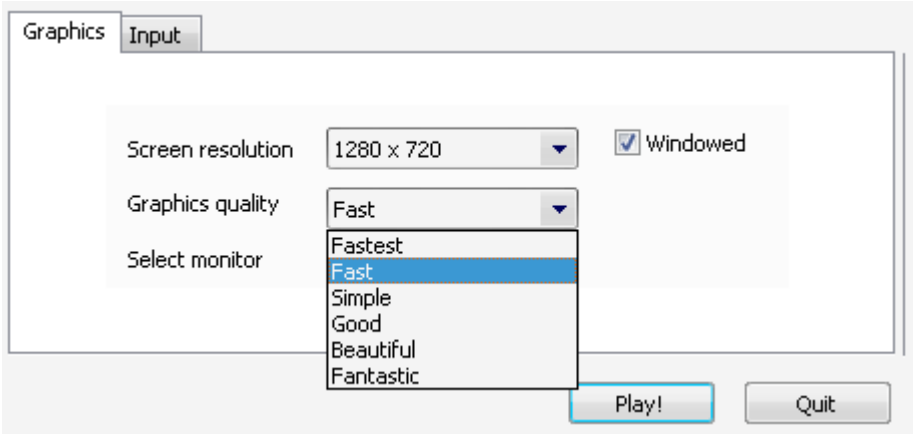

Рис. 2.39. Зміна налаштування графіки

Було проведено низку тестів та виправлень помилок гри. Кількість кадрів в секунду стабільна, провисань не спостерігається.

РОЗДІЛ 3
ЕКОНОМІЧНИЙ РОЗДІЛ

3.1. Розрахунок трудомісткості та вартості розробки програмного продукту

Вхідні дані:

- передбачуване число операторів – 800;
- коефіцієнт складності програми – 2;
- коефіцієнт корекції програми в ході її розробки – 0,08;
- годинна заробітна плата програміста, грн / год – 40.

Нормування праці в процесі створення ПЗ істотно ускладнено в силу творчого характеру праці програміста. Тому трудомісткість розробки ПЗ може бути розрахована на основі системи моделей з різною точністю оцінки.

Трудомісткість розробки ПЗ можна розрахувати за формулою:

$$t = t_u + t_a + t_n + t_{отл} + t_д, \text{ людино-годин,} \quad (3.1)$$

де to - витрати праці на підготовку й опис поставленої задачі (приймається 50),

ти - витрати праці на дослідження алгоритму рішення задачі,

та - витрати праці на розробку блок-схеми алгоритму,

тп - витрати праці на програмування по готовій блок-схемі,

тотл - витрати праці на налагодження програми на ЕОМ,

тд - витрати праці на підготовку документації.

Складові витрати праці визначаються через умовне число операторів у ПЗ, яке розробляється.

Умовне число операторів (підпрограм):

$$Q = q \times C \times (1+ p) \text{ , людино-годин,} \qquad (3.2)$$

де q - передбачуване число операторів,

С - коефіцієнт складності програми,

р - коефіцієнт кореляції програми в ході її розробки.

$$Q = 800 \cdot 2 \cdot (1 + 0{,}08) = 1728 \text{ людино-годин.}$$

Витрати праці на вивчення опису задачі ti визначається з урахуванням уточнення опису і кваліфікації програміста:

$$t_u = \frac{QB}{(75...85)K} \text{ , людино-годин,} \qquad (3.3)$$

де В - коефіцієнт збільшення витрат праці внаслідок недостатнього опису задачі; В=1.2 ... 1.5,

k - коефіцієнт кваліфікації програміста, обумовлений від стажу роботи з даної спеціальності.

Витрати праці на розробку алгоритму рішення задачі:

$$t_u = \frac{1728 \cdot 1{,}2}{80 \cdot 0{,}8} = 32 \text{ , людино-годин.}$$

Витрати на складання програми по готовій блок-схемі:

$$t\alpha = \frac{Q}{(20...25)K} \quad \text{людино-годин.} \qquad (3.4)$$

$$t_a = \frac{1728}{20 \cdot 0{,}8} = 108 \text{ людино-годин.}$$

Витрати на складання програми по готовій блок-схемі:

$$t_n = \frac{Q}{(20...25)K} \quad \text{людино-годин.} \tag{3.5}$$

$$t_n = \frac{1728}{25 \cdot 0{,}8} = 86 \text{ людино-годин.}$$

Витрати праці на налагодження програми на ЕОМ:

$$t_{отл} = \frac{Q}{(4...5)K} \quad \text{людино-годин.} \tag{3.6}$$

$$t_{отл} = \frac{1728}{4 \cdot 0{,}8} = 540 \text{ людино-годин.}$$

За умови комплексного налагодження завдання:

$$t^{к}_{отл} = 1{,}2 \cdot t_{отл}; \tag{3.7}$$

$$t_{отл} = 540 * 1{,}2 = 648$$

Витрати праці на підготовку документації:

$$t_д = t_{др} + t_{до}, \text{ людино-годин,} \tag{3..8}$$

де tдр - трудомісткість підготовки матеріалів і рукопису.

$$t_{др} = \frac{Q}{(15...20)K}, \text{ людино-годин.} \tag{3.9}$$

$$t_{др} = \frac{1728}{15 \cdot 0{,}8} = 144 \text{ людино-годин,}$$

tдо - трудомісткість редагування, печатки й оформлення документації

$$t_{до} = 0{,}75 \cdot t_{др}, \text{ людино-годин.} \quad (3.10)$$

$$tdo = 0{,}75 \cdot 144 = 108$$

$$t_д = 108 + 144 = 252 \text{ людино-годин.}$$

Отримуємо трудомісткість розробки програмного забезпечення:

$$t = 50 + 32 + 108 + 86 + 648 + 252 = 1176 \text{ людино-годин.}$$

3.2. Розрахунок витрат на створення програми

Витрати на створення ПЗ Кпо включають витрати на заробітну плату виконавця програми Зз/п і витрат машинного часу, необхідного на налагодження програми на ЕОМ.

$$К_{ПО} = З_{ЗП} + З_{МВ}, \text{ грн,} \quad (3.11)$$

Заробітна плата виконавців визначається за формулою:

$$З_{ЗП} = t \cdot С_{ПР}, \text{ грн,} \quad (3.12)$$

де t - загальна трудомісткість, людино-годин,

Спр - середня годинна заробітна плата програміста, грн/година.

$$З_{ЗП} = 1176 \cdot 40 = 47072 \text{ грн.}$$

Вартість машинного часу, необхідного для налагодження програми на ЕОМ:

$$З_{МВ} = t_{отл} \times С_М, \text{ грн,} \quad (3.13)$$

де tотл - трудомісткість налагодження програми на ЕОМ, год,

Смч - вартість машино-години ЕОМ, 7 грн/год.

Визначені в такий спосіб витрати на створення програмного забезпечення є частиною одноразових капітальних витрат на створення АСУП.

$$З_{МВ} = 648 \times 7 = 4536 \text{ грн.}$$

$$Кпо = 47072 + 4536 = 51608 \text{ грн.}$$

Очікуваний період створення ПЗ:

$$T = \frac{t}{B_k \cdot F_p} \text{ міс.} \qquad (3.14)$$

де Bk - число виконавців,

Fp - місячний фонд робочого часу (при 40 годинному робочому тижні Fp=176 годин).

$$T = \frac{1176}{1 \cdot 176} = 6{,}7 \text{ міс.}$$

Визначено трудомісткість розробленої інформаційної системи (1176 люд-год), проведений підрахунок вартості роботи по створенню програми (51608 грн.) та розраховано час на його створення (6,7 міс).

ВИСНОВКИ

Метою кваліфікаційної роботи є аналіз доступних технологій та методів розробки для створення ігрових програм та розробка власної комп'ютерні 3D ігри в середовищі UNITY 3D.

Комп'ютерна гра «Break your hands» розроблена для платформи Windows, жанр: платформер у піксельній стилістиці з тривимірним оточенням. Реалізація виконана на платформі Unity, мова програмування для написання скриптів та ігрової логіки – C#. Об'єктно-орієнтована концепція мови програмування C# найліпше підходить для описання ігрової логіки об'єктів за допомогою системи класів. Вибране програмне забезпечення Blender цілком підійшло для моделювання тривимірних об'єктів, що складають рівень.

Кожен рівень спроектовано таким чином, що користувач повинен минаючи перешкоди і ворогів, з найвищою кількістю балів дійти до кінця рівня, водночас керуючи персонажем та ігровими об'єктами. Ігровий процес візуально транслює користувачеві максимально чутливе і плавне управління персонажем та ігровими об'єктами.

Виконано тестування та відладку гри, всі знайдені помилки було виправлено. Гра коректно завантажує рівні, персонаж переміщується згідно команд периферійних пристроїв, вороги поводяться згідно передбачених алгоритмів. Розробка гри велась на основі поставленого завдання та спроектованих діаграм алгоритмів проекту

В результаті створено повноцінний працеспроможний додаток, що надає можливість реалізувати комп'ютерну гру, в якій використовується сучасний графічний дизайн, невибагливі правила гри, зміна налаштувань під конкретного гравця, та який призначений для розваги користувачів..

В економічному розділі визначено трудомісткість розробленого програмного продукту (1176 люд-год), проведений підрахунок вартості роботи по створенню програми (51608 грн.) та розраховано час на його створення (6,7 міс).

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Документація Unity URL: http://docs.unity3d.com/Manual/ UnityManual.html/. дата звернення: 15.04.2021.
2. Шилдт Г. C# 4.0: повне керівництво / Г. Шилдт., 2011. – 451 с.
3. Документація Visual Studio URL: docs.microsoft.com/ru-ru/visualstudio/get-started/visual-studio-ide?view=vs-2019. дата звернення: 15.04.2021.
4. Розендаль Т. Інструменти моделювання в Blender / Тон Розендаль., 2016. – 135 с.
5. Чекмарьов А. Операційні системи / А. Чекмарьов., 2014. – 93 с.
6. Поташників А. Особливості застосування методів DirectX / А. Поташників., 2016. – 48 с.
7. Техно-блог Engadget. “Ігрова археологія” стаття про гру Maze War URL: https://www.engadget.com/2012/06/12/the-game-archaeologist-maze-war// дата звернення: 15.04.2021.
8. 15 programming languages you need to know in 2015. URL: http://mashable.com/2015/01/18/programming-languages-015/#myt.sTr5omq1/ дата звернення: 15.04.2021.
9. Blender. Офіційний веб-портал компанії Blender Foundation URL: https://www.blender.org// дата звернення: 15.04.2021.
10. Unity. Офіційний веб-портал компанії Unity Technologies. URL: https://unity3d.com/. дата звернення: 15.04.2021.
11. Бібліографічний запис. Бібліографічний опис. Загальні вимоги та правила складання : (ГОСТ 7.1-2003, IDT) : ДСТУ ГОСТ 7.1:2006. – Чинний з 2007–07–01. – К. : Держспоживстандарт України, 2007. – 47 с. – (Система стандартів з інформації, бібліотечної та видавничої справи) (Національний стандарт України).
12. Бусигін Б.С., Коротенко Г.М., Коротенко Л.М. Прикладна інформатика. Підручник для студентів комп'ютерних спеціальностей. –

Дніпропетровськ: Видавництво НГУ, 2004. – 559 с. URL: http://www.programmer.dp.ua/book-ua-k01.php. дата звернення: 15.03.2019.

13. Бусыгин Б.С., Дивизинюк М.М., Коротенко Г.М., Коротенко Л.М. Введение в современную информатику. Учебник. – Севастополь: Издательство СНУЯЭиП, 2005. – 644 с. / URL: http://www.programmer.dp.ua/book-ru-k02.php. дата звернення: 15.01.2018.

14. Методичні вказівки з виконання економічного розділу в дипломних проектах студентів спеціальності “Комп’ютерні системи ” / Укладачі О.Г. Вагонова, Нікітіна О.Б. Н.Н. Романюк – Дніпропетровськ: Національний гірничий університет. – 2013. – 23с.

15. Методичні рекомендації до виконання кваліфікаційних робіт бакалаврів напряму підготовки 121 «Програмна інженерія» галузі знань 12 Інформаційні технології/, Л.М. Коротенко , О.С. Шевцова; Нац. гірн. ун-т. – Д : ДВНЗ НГУ, 2019. – 65 с.

16. Benjamin Pierce. Types and Programming Languages. Press, 2002.– 221с.

17. Analog Devices ADP176x CMOS Linear Regulators. URL: http://spectrum.ieee.org/computing/software/the-2016-top-programming-languages. дата звернення: 21.03.2021.

18. Color Contrast Analyzer URL: http://tools.cactusflower.org/analyzer/ Дата звернення: 21.02.2021.

19. Matt Weisfeld. The Object-Oriented Thought Process. - Fourth Edition. - Addison-Wesley Professional, 2013. - 336 с. - ISBN 978-0-321-86127-6.

20. MiniMax и NegaMax URL: http://src-code.net/minimax-i-negamax/ дата звернення: 21.03.2021.

ДОДАТОК А

КОД ПРОГРАМИ

Player.cs:

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
public class Player : MonoBehaviour
{

    [Header("Movement")]
    public float moveSpeed = 10f;
    private Vector2 direction;
    private bool facingRight = true;
    public float jumpForce = 15f;
    public float jumpDelay = 0.25f;
    private float jumpTimer;
    public bool canJump = true;
    public bool canParent = false;
    [Header("Physics")]
    public float maxSpeed = 7f;
    public float linearDrag = 4f;
    public float gravity = 1f;
    public float fallMultiplier = 5f;
    [Header("Collision")]
    public LayerMask groundLayer;
    public float groundLength = 0.6f;
    public bool onGround = false;
    public Vector3 colliderOffset;
    private bool changingDirections;
    private bool canDamage = true;
    [Header("Components")]
    public Rigidbody2D rb;
    public Animator animator;
    public SpriteRenderer sprite;
    public HUD hud;
    public Material mat;
    public ParticleSystem dust;
    public respowned res;
    public Vector3 startPos;
    private int health = 3;
    private Vector3 startCam;
    float fallBak;
    private void Start()
    {
        hud.setHP(health);
        mat.color = Color.white;
        startPos = transform.position;
```

```
        fallBak = fallMultiplier;
    }

    void Update()
    {
        if (!PauseMenu.GameIsPaused)
        {
            bool wasOnGround = onGround;
            onGround = Physics2D.Raycast(transform.position + colliderOffset, Vec-
tor2.down, groundLength, groundLayer)
                    ||  Physics2D.Raycast(transform.position - colliderOffset, Vec-
tor2.down, groundLength, groundLayer);
            if (!wasOnGround && onGround)
            {
                canJump = true;
                createDust();
                StartCoroutine(JumpSqueeze(1.25f, 0.8f, 0.05f));
            }
            if (Input.GetButtonDown("Jump"))
                jumpTimer = Time.time + jumpDelay;
direction = new Vector2(Input.GetAxisRaw("Horizontal"), Input.GetAxis-
Raw("Vertical"));
if (Mathf.Abs(Input.GetAxisRaw("Horizontal")) > 0 && (rb.velocity.y == 0 ||onGround))
            {
                animator.SetBool("isRunning", true);
            }
            else
                animator.SetBool("isRunning", false);
            if (!canParent)
                transform.SetParent(null);
            if (rb.velocity.y > 0 && !onGround)
            {
                animator.SetBool("isJumping", true);
                animator.SetBool("isFalling", false);
            }

            if (rb.velocity.y < 0 && !onGround)
            {
                animator.SetBool("isJumping", false);
                animator.SetBool("isFalling", true);
            }
            if (rb.velocity.y == 0 || onGround)
            {
                animator.SetBool("isJumping", false);
                animator.SetBool("isFalling", false);
            }
if (Input.GetKey(KeyCode.LeftAlt) && Input.GetKeyDown(KeyCode.Keypad1))
                takeDamage();
            if (Input.GetKey(KeyCode.LeftAlt) && Input.GetKeyDown(KeyCode.Keypad2))
                death();
            //hud.setVelocity(rb.velocity.x, rb.velocity.y, rb.drag);
        }
```

```
}
void FixedUpdate()
{
    moveCharacter(direction.x);
    if (jumpTimer > Time.time && onGround)
        Jump();

    modifyPhysics();
}
private void OnCollisionEnter2D(Collision2D collision)
{
    if (collision.gameObject.layer == 10
    && transform.position.y - (transform.localScale.y / 2)
    > collision.transform.position.y + (collision.transform.localScale.y / 3))
    {
        canJump = false;
        rb.AddForce(Vector2.up * 25, ForceMode2D.Impulse);
    }
    if (collision.gameObject.tag == "Lava")
        death();
    if (canParent && !Input.GetButtonDown("Jump") && direction.x == 0)
    {
        transform.SetParent(collision.transform);
    }
}
private void OnCollisionStay2D(Collision2D collision)
{
    if (canParent && !Input.GetButtonDown("Jump"))
    {
        transform.SetParent(collision.transform);
    }
}
private void OnCollisionExit2D(Collision2D collision)
{
    if (collision.gameObject.tag != "Sadding" || Input.GetButton("Jump"))
        transform.SetParent(null);
}
public void takeDamage()
{
    if (canDamage)
    {
        if (health > 1)
        {
            health--;
            canDamage = false;
            Invoke("reload", 0.8f);
            hud.setHP(health);
            sprite.material.color = new Color(1f, 0f, 0f);
            StartCoroutine(JumpSqueeze(1.2f, 0.8f, 0.1f));
            Invoke("clearColor", 0.2f); if (facingRight)
                rb.AddForce(new Vector2(-2, 1) * 8, ForceMode2D.Impulse);
            else
```

```
                    rb.AddForce(new Vector2(2, 1) * 8, ForceMode2D.Impulse);
            }
            else
            {
                death();
            }
        }
}
public void takeDamage(bool toRight)

{
    if (canDamage)
    {
        if (health > 1)
        {
            health--;
            canDamage = false;
            Invoke("reload", 0.8f);
            hud.setHP(health);
            sprite.material.color = new Color(1f, 0f, 0f);
            StartCoroutine(JumpSqueeze(1.2f, 0.8f, 0.1f));
            Invoke("clearColor", 0.2f); if (!toRight)
                rb.AddForce(new Vector2(-2, 1) * 8, ForceMode2D.Impulse);
            else
                rb.AddForce(new Vector2(2, 1) * 8, ForceMode2D.Impulse);
        }
        else
        {
            death();
        }
    }
}
void reload()
{
    canDamage = true;
}
void clearColor()
{
    sprite.material.color = Color.white;
}
public void death()
{
    health = 0;
    hud.setHP(0);
    hud.fadeOut();
    sprite.material.color = new Color(1f, 0f, 0f, 0f);
    rb.simulated = false;
    Invoke("respown", 1f);
    res.resp = true;
}
void respown()
{
```

```
            rb.simulated = true;
            fallMultiplier = 999999999;
            transform.position = startPos;
            Invoke("fallB", 0.1f);
            health = 3;
            hud.setHP(health);
            res.resp = false;
            Invoke("fadeIn", 0.5f);
            Invoke("clearColor", 0.2f);
        }
        void fallB()
        {
            fallMultiplier = fallBak;
            Jump();
        }
        void fadeIn()
        {
            hud.fadeIn();
        }
        void createDust()
        {
            dust.Play();
        }
        // Movement
        void moveCharacter(float horizontal)
        {
            rb.AddForce(Vector2.right * horizontal * moveSpeed);
            if ((horizontal > 0 && !facingRight) || (horizontal < 0 && facingRight)) Flip();
            if (Mathf.Abs(rb.velocity.x) > maxSpeed)
rb.velocity = new Vector2(Mathf.Sign(rb.velocity.x) * maxSpeed, rb.velocity.y);
                if (Mathf.Abs(horizontal) > 0)
                    transform.SetParent(null);
        }
        void Jump()
        {
            transform.SetParent(null);
                createDust();
                rb.velocity = new Vector2(rb.velocity.x, 0);
                rb.AddForce(Vector2.up * jumpForce, ForceMode2D.Impulse);
                jumpTimer = 0;
                StartCoroutine(JumpSqueeze(0.5f, 1.2f, 0.1f));
        }
        void modifyPhysics()
        {
            changingDirections = (direction.x > 0 && rb.velocity.x < 0) ||
                                 (direction.x < 0 && rb.velocity.x > 0);
            if (onGround)
            {
                rb.drag = 10f;
                rb.gravityScale = 0;
            }
            else
```

```
        {
            rb.gravityScale = gravity;
            rb.drag = linearDrag * 0.20f;
            if(canJump)
            {
                if (rb.velocity.y < 4)
                    rb.gravityScale = gravity * fallMultiplier;
                else if (rb.velocity.y > 0 && !Input.GetButton("Jump"))
                    rb.gravityScale = gravity * (fallMultiplier);
            }
            else
            {
                if (rb.velocity.y < 4)
                    rb.gravityScale = gravity * fallMultiplier;
                else if (rb.velocity.y > 0)
                    rb.gravityScale = gravity * (fallMultiplier);
            }
        }
    }
    void Flip()
    {
        facingRight = !facingRight;
        sprite.flipX = !sprite.flipX;
        if(onGround && Mathf.Abs(rb.velocity.x) > 4)
            createDust();
    }
    IEnumerator JumpSqueeze(float xSqueeze, float ySqueeze, float seconds)
    {
        Vector3 originalSize = Vector3.one;
        Vector3 newSize = new Vector3(xSqueeze, ySqueeze, originalSize.z);
        float t = 0f;
        while (t <= 1.0)
        {
            t += Time.deltaTime / seconds;
            transform.localScale = Vector3.Lerp(originalSize, newSize, t); yield
            return null;
        }
        t = 0f;
        while (t <= 1.0)
        {
            t += Time.deltaTime / seconds;
            transform.localScale = Vector3.Lerp(newSize, originalSize, t); yield
            return null;
        }
    }
    private void OnDrawGizmos()
    {
        Gizmos.color = Color.red;
        Gizmos.DrawLine(transform.position + colliderOffset, transform.position + col-
liderOffset + Vector3.down * groundLength);
        Gizmos.DrawLine(transform.position - colliderOffset, transform.position - col-
liderOffset + Vector3.down * groundLength); }}
```

Enemy.cs:

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
public class Enemy : MonoBehaviour
{
    public float speed = 20f;
    public float direction = 1;
    public float maxSpeed = 30;
    [Header("Colliders")]
    public float groundLength = 2f;
    public float bgrLength = 2f;
    public float wallLength = 2f;
    public float backLength = 2f;
    public Vector3 grouOffset;
    public Vector3 bgrOffset;
    public Vector3 wallOffset;

    public Vector3 backOffset;
    public LayerMask groundLayer;
    [Header("Components")]
    public Player player;
    public SpriteRenderer sprite;
    public Rigidbody2D rb;
    public BoxCollider2D box;
    bool contact = false;
    bool facingRight = true;
    public bool isStand = false;
    public int health = 3;
    public bool stop = false;
    private void OnCollisionEnter2D(Collision2D collision)
    {
        isStand = true;
        if (collision.gameObject.layer == 8)
if ((collision.gameObject.transform.position.y - collision.transform.lo-calScale.y / 2 + 0.2f) <
                (transform.position.y + transform.localScale.y / 2))
            {
                if (collision.transform.position.x > transform.position.x)
                    player.takeDamage(true);
                else
                    player.takeDamage(false);
            }
            else
            {
                takeDamage();
            }
        if (collision.gameObject.layer == 9)
        {
            transform.SetParent(collision.transform);
        }
        if (collision.gameObject.tag == "Lava")
```

```
            {
                death();
            }
        }
        private void OnCollisionStay2D(Collision2D collision)
        {
            isStand = true;
            if (collision.gameObject.layer == 8)
if ((collision.gameObject.transform.position.y - collision.transform.lo-calScale.y / 2 + 0.2f) <
                    (transform.position.y + transform.localScale.y / 2))
                {
                    if(collision.transform.position.x > transform.position.x)
                        player.takeDamage(true);
                    else
                        player.takeDamage(false);
                }
        }
        void takeDamage()
        {
            if (health > 1)
            {
                health--;
                sprite.material.color = new Color(1f, 0f, 0f);
                Invoke("clearColor", 0.2f);
            }
            else
            {
                sprite.material.color = Color.black;
                Invoke("death", 0.2f);
                clearColor();
            }
        }
        public void death()
        {
            Destroy(gameObject);
        }
        private void OnCollisionExit2D(Collision2D collision)
        {
            isStand = false;
            transform.SetParent(null);
        }
        void clearColor()
        {
            sprite.material.color = Color.white;
        }
        private void OnTriggerExit2D(Collider2D collision)
        {
            if (collision.gameObject.layer == 8)
            {
                contact = false;
                box.size = new Vector2(5, 0.8f);
            }
```

```
    }
    private void OnTriggerEnter2D(Collider2D collision)
    {
        if (collision.gameObject.layer == 8)
        {
            contact = true;
            box.size = new Vector2(8, 0.8f);
        }
    }
    private void FixedUpdate()
    {
        RaycastHit2D grouRay = Physics2D.Raycast(transform.position + grouOffset, Vec-
tor2.down, groundLength, groundLayer);
        RaycastHit2D wallRay = Physics2D.Raycast(transform.position + wallOffset, Vec-
tor2.down, wallLength, groundLayer);
        RaycastHit2D backRay = Physics2D.Raycast(transform.position + backOffset, Vec-
tor2.down, backLength, groundLayer);
        RaycastHit2D bgrRay = Physics2D.Raycast(transform.position + bgrOffset, Vec-
tor2.down, bgrLength, groundLayer);
            if (contact && !stop)
            {
                moveTo(player.transform, speed);
            }
if (!grouRay.collider && isStand || (wallRay.collider && rb.velocity.y ==0))
            {
                if (facingRight) turnTo("L");
                else turnTo("R");
            }
            if (grouRay.collider && isStand && !stop)
            {
                if (Mathf.Abs(rb.velocity.x) < maxSpeed)
                    rb.AddForce(Vector2.right * speed * direction);
            }
    }
    void moveTo(Transform pos, float sp)
    {
        float pX = pos.position.x, pY = pos.position.y;
        float eX = transform.position.x, eY = transform.position.y;
        if (pX < eX)
        {
            turnTo("L");
        }
        else if (pX > eX)
        {
            turnTo("R");
        }
        rb.AddForce(Vector2.right * speed * direction);
    }
    void turnTo(string direc)
    {
        if (direc == "R" && !facingRight)
        {
```

```
                facingRight = true;
                sprite.flipX = false;
                direction = 1;
grouOffset = new Vector3(Mathf.Abs(grouOffset.x), grouOffset.y, grouOffset.z);
                wallOffset = new Vector3(Mathf.Abs(wallOffset.x), wallOffset.y, wallOff-
set.z);
                backOffset = new Vector3(Mathf.Abs(backOffset.x) * -1, backOffset.y,
backOffset.z);
                bgrOffset = new Vector3(Mathf.Abs(bgrOffset.x) * -1, bgrOffset.y, bgrOff-
set.z);
                    rb.velocity = new Vector2(speed / 2 * direction, rb.velocity.y);
            }
            else if (direc == "L" && facingRight)
            {
                facingRight = false;
                sprite.flipX = true;
                direction = -1;
                grouOffset = new Vector3(Mathf.Abs(grouOffset.x) * -1, grouOffset.y,
grouOffset.z);
                wallOffset = new Vector3(Mathf.Abs(wallOffset.x) * -1, wallOffset.y,
wallOffset.z);
                backOffset = new Vector3(Mathf.Abs(backOffset.x), backOffset.y, backOff-
set.z);
                bgrOffset = new Vector3(Mathf.Abs(bgrOffset.x), bgrOffset.y, bgrOffset.z);
                rb.velocity = new Vector2(speed / 2 * direction, rb.velocity.y);
            }
        }

        private void OnDrawGizmos()
        {
            Gizmos.color = Color.red;
            Gizmos.DrawLine(transform.position + grouOffset, transform.position + grouOffset +
Vector3.down * groundLength);
            Gizmos.DrawLine(transform.position + wallOffset, transform.position + wallOffset +
Vector3.down * wallLength);
            //Gizmos.DrawLine(transform.position + backOffset, transform.position + backOff-set +
Vector3.down * wallLength);
            //Gizmos.DrawLine(transform.position + bgrOffset, transform.position + bgrOffset
+ Vector3.down * bgrLength);
        }
}

Checkpoint.cs:
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
public class Checkpoint : MonoBehaviour
{
    bool used = false;
    private void OnTriggerEnter2D(Collider2D collision)
    {
        if (collision.gameObject.layer == 8 && used == false)
```

```
            {
                gameObject.GetComponent<Renderer>().material.color = Color.green;
                collision.gameObject.GetComponent<Player>().startPos =
                    new Vector3(transform.position.x, transform.position.y + 3, trans-
form.position.z);
                used = true;
            }
        }
}
```

Finish.cs:

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine.UI;
using UnityEngine.EventSystems;
using UnityEngine;
public class Finish : MonoBehaviour
{
    public static bool levelFinished = false;
    public int difficultyMultiplayer;

    public int level;
    public GameObject FinishMenu, startButton;
    public Text TextScore;
    private void OnTriggerEnter2D(Collider2D collision)
    {
        if (collision.gameObject.layer == 8)
        {
            Time.timeScale = 0f;
            PauseMenu.GameIsPaused = true;
            levelFinished = true;
            PlayerPrefs.SetInt("aLevel" + (level + 1), 1);
            PlayerPrefs.Save();
            HUD hud = HUD.Instance;
float time = (hud.minutes * 0.4f) + (hud.seconds * 0.25f) + (hud.miliseconds * 0.05f);
            int score = (int.Parse(hud.likes.text) - int.Parse(hud.dislikes.text))
*           (difficultyMultiplayer) - (int) time; if (score < 0) score = 0;
            TextScore.text = "СЧЁТ: " + score;
            FinishMenu.SetActive(true);
            EventSystem.current.SetSelectedGameObject(null);
            EventSystem.current.SetSelectedGameObject(startButton);
            if (PlayerPrefs.HasKey("Score" + level) && PlayerPrefs.GetInt("Score" +
level) < score)
            {
                PlayerPrefs.SetInt("Score" + level, score);
                PlayerPrefs.Save();
            }
            else if (!PlayerPrefs.HasKey("Score" + level))
            {
                PlayerPrefs.SetInt("Score" + level, score);
                PlayerPrefs.Save();
            }
```

```
            }
        }
        private void Update()
        {
            if (PauseMenu.GameIsPaused)
            {
                if (Input.GetAxis("Mouse X") != 0 || Input.GetAxis("Mouse Y") != 0)
                {
                    EventSystem.current.SetSelectedGameObject(null);
                }
                if (Input.GetAxis("Vertical") > 0 && EventSystem.current.currentSelected-
GameObject == null)
                {
                    EventSystem.current.SetSelectedGameObject(null);
                    EventSystem.current.SetSelectedGameObject(startButton);
                }
            }
        }
}

pHold.cs:
using System.Collections;
using System.Collections.Generic;
using TMPro;
using UnityEngine;
public class pHold : MonoBehaviour
{
    [Header("Values")]
    public float offset = 2f;
    public float smoothness = 0.125f;
    public KeyCode button;
    [Header("Components")]
    public TextMeshPro key;
    public Transform platform;
    private Vector3 pStart, pEnd;
    private void Start()
    {
        pStart = platform.position;
        pEnd = pStart + new Vector3(offset, 0f, 0f);
        key.text = button.ToString();
    }
    void Update()
    {
        if (Input.GetKey(button) && !PauseMenu.GameIsPaused)
        {
            key.color = Color.red;
            if (platform.position.x < pEnd.x)
                platform.position = Vector3.Lerp(platform.position, platform.position + new
Vector3(2f, 0f, 0f), smoothness);
        }
        else if (!Input.GetKey(button) && !PauseMenu.GameIsPaused)
        {
```

```
            platform.position = Vector3.Lerp(platform.position, pStart, smoothness);
        }
        if (Input.GetKeyUp(button))
        {
            key.color = new Color(217, 217, 217);

        }
    }
}
```

pRotated.cs:

```
using System.Collections;
using System.Collections.Generic;
using TMPro;
using UnityEngine;
public class pRoteted : MonoBehaviour
{
    [Header("Values")]
    public float smoothness = 0.125f;
    public float angle;
    public KeyCode button;
    [Header("Components")]
    public TextMeshPro key;
    public Transform pivot;
    public Transform objects;
    private void Start()
    {
        key.text = button.ToString();
        objects.SetParent(pivot);
    }
    void Update()
    {
        if (Input.GetKey(button) && !PauseMenu.GameIsPaused)
        {
            key.color = Color.red;
            if (pivot.rotation.z < angle)

                pivot.rotation = Quaternion.Lerp(pivot.rotation, Quaternion.Euler(0, 0, angle),
smoothness);
        }
        else if (!Input.GetKey(button) && !PauseMenu.GameIsPaused)
        {
            pivot.rotation = Quaternion.Lerp(pivot.rotation, Quaternion.Euler(0, 0, 0),
smoothness);
        }
        if (Input.GetKeyUp(button))
        {
            key.color = new Color(217, 217, 217);
        }
    }
}
```

pSadding.cs:

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
public class pSadding : MonoBehaviour
{
    public parentTrigger pairTrig;
    public float smoothness = 0.3f;
    public float sagging = 0.2f;
    public bool isStay = false;
    public bool under = false;
    Vector3 pStart;
    Vector3 newPos;
    void Start()
    {
        pStart = transform.position;
    }
    private void OnCollisionEnter2D(Collision2D collision)
    {
        if (collision.transform.position.y > transform.position.y + 1)
        {
            isStay = true;
        }
        else if (collision.transform.position.y < transform.position.y - 1.1f)
            under = true;
    }
    private void OnCollisionStay2D(Collision2D collision)
    {
        if (collision.transform.position.y > transform.position.y + 1)
        {
            isStay = true;
        }
        else if (collision.transform.position.y < transform.position.y - 1.1f)
            under = true;
    }
    private void OnCollisionExit2D(Collision2D collision)
    {
        Invoke("ret", 0.25f);
    }

    void Update()
    {
        if (pairTrig.stay)
        {
            transform.position = Vector3.Lerp(transform.position, new Vector3(pStart.x,
pStart.y - sagging, pStart.z), smoothness);
        }
        else
        {
            transform.position = Vector3.Lerp(transform.position, pStart, smoothness);
        }
        if (under)
```

```
            {
                transform.position = Vector3.Lerp(transform.position, new Vector3(pStart.x,
pStart.y + sagging + 0.2f, pStart.z), smoothness/3);
            }
        }
        void ret()
        {
            isStay = false;
            under = false;
        }
}

pSlide.cs:
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using TMPro;
public class pSlide : MonoBehaviour
{
        [Header("Values")]
        public float speed;
        public float smoothness = 0.125f;
        public KeyCode button;
        [Header("Components")]
        public Transform pos1;
        public Transform pos2;
        public Transform pivot;
        public Transform platform;
        public TextMeshPro key;
        Vector3 nextPos;
        Vector3 startPos;
        void Start()
        {
            startPos = pivot.transform.position;
            nextPos = pos1.position;
            key.text = button.ToString();
        }
        void Update()
        {
            if(GameObject.Find("Respowned").GetComponent<respowned>().resp == true)
            {
                pivot.position = startPos;
            }

            if (pivot.position == pos1.position)
            {
                nextPos = pos2.position;
            }
            if (pivot.position == pos2.position)
            {
                nextPos = pos1.position;
            }
```

```
            if (Input.GetKey(button) && !PauseMenu.GameIsPaused)
            {
                key.color = Color.red;
                pivot.position = Vector3.MoveTowards(pivot.position, nextPos, speed *
Time.deltaTime);
            }
            else if (Input.GetKeyUp(button))
            {
                key.color = new Color(217, 217, 217);
            }
            platform.position = Vector3.Lerp(platform.position, pivot.position, smoothness);
        }
        private void OnDrawGizmos()
        {
            Gizmos.color = Color.black;
            Gizmos.DrawLine(pos1.position, pos2.position);
        }
}
```

pTossing.cs:

```
using System.Collections;
using System.Collections.Generic;
using TMPro;
using UnityEngine;
public class pTossing : MonoBehaviour
{
    [Header("Values")]
    public float force;
    public KeyCode button;
    [Header("Components")]
    public TextMeshPro key;
    public Transform platform;
    bool isStay = false;
    Collider2D body;
    private void Start()
    {
        key.text = button.ToString();
    }
    private void OnTriggerEnter2D(Collider2D collision)
    {
        body = collision;
        isStay = true;
        if(collision.gameObject.layer == 8)
            collision.gameObject.GetComponent<Player>().canJump = true;
    }
    private void OnTriggerExit2D(Collider2D collision)
    {
        isStay = false;
    }
    void addforce(Collider2D collision)
    {
```

```
            if (collision.gameObject.layer == 8)
                collision.gameObject.GetComponent<Player>().canJump = false;
            collision.gameObject.GetComponent<Rigidbody2D>().AddForce(new
Vector2(0f, force));
        }
        private void Update()
        {
            if (isStay && Input.GetKeyDown(button) && !Input.GetButtonDown("Jump") &&
!In-put.GetButtonUp("Jump"))
            {
                addforce(body);
            }
            if (Input.GetKey(button) && !PauseMenu.GameIsPaused)
            {
                platform.localScale = new Vector3(1, 1.2f, 1);
                key.color = Color.red;
            }
            else if (Input.GetKeyUp(button))
            {
                platform.localScale = new Vector3(1, 1, 1);
                key.color = new Color(217, 217, 217);
            }
        }
}
```

CastShadow.cs:

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
public class CastShadow : MonoBehaviour
{
        void Start()
        {
            Renderer renderer =
            GetComponent<Renderer>(); if (renderer ==
            null)
                Debug.Log("Renderer is empty");
            GetComponent<Renderer>().shadowCastingMode = UnityEngine.Rendering.ShadowCast-
ingMode.TwoSided;
            GetComponent<Renderer>().receiveShadows = true;
        }
}
```

Dislike.cs:

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
public class Dislike : MonoBehaviour
{

        private void OnTriggerEnter2D(Collider2D collision)
```

```
    {
        if (collision.gameObject.layer == 8)
        {
            HUD.Instance.dislike();
            Destroy(gameObject);}}}
```

Fullscreen.cs:

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
public class Fullscreen : MonoBehaviour
{
    public bool full;
    private void Start()
    {
        Screen.fullScreen = full;
    }
}
```

KeyFollow.cs:

```
using System.Collections;
using System.Collections.Generic;
using TMPro;
using UnityEngine;
public class KeyFollow : MonoBehaviour
{
    public Transform maxKeyY, minKeyY, player;
    public TextMeshPro key;
    bool contact = false;
    public respowned res;
    Vector3 startPos;
    private void Start()
    {
        startPos = key.transform.position;
    }
    private void OnTriggerEnter2D(Collider2D collision)
    {
        contact = true;
    }
    private void OnTriggerExit2D(Collider2D collision)
    {
        contact = false;
    }
    void Update()
    {
        if(contact)
        {
            if (key.transform.position.y <= maxKeyY.position.y && key.transform.posi-
tion.y >= minKeyY.position.y)
            {
                key.transform.position = new Vector3(key.transform.position.x,
player.position.y + 2, key.transform.position.z);
```

```
                }
                if (key.transform.position.y > maxKeyY.position.y)
                {
                    key.transform.position = new Vector3(key.transform.position.x,
maxKeyY.position.y, key.transform.position.z);
                }
                if (key.transform.position.y < minKeyY.position.y)
                {
                    key.transform.position = new Vector3(key.transform.position.x,
minKeyY.position.y, key.transform.position.z);
                }

            }
            else if (res.resp)
            {
                key.transform.position = startPos;
            }
        }
}
```

kill.cs:

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
public class kill : MonoBehaviour
{
    private void OnCollisionEnter2D(Collision2D collision)
    {
        if (collision.gameObject.layer == 8)
            collision.gameObject.GetComponent<Player>().death();
        if (collision.gameObject.layer == 10)
            collision.gameObject.GetComponent<Enemy>().death()
            ;
    }
}
```

Like.cs:

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
public class Like : MonoBehaviour
{
    private void OnTriggerEnter2D(Collider2D collision)
    {
        if (collision.gameObject.layer == 8)
        {
            HUD.Instance.like();
            Destroy(gameObject);
        }
    }
}
```

parentTrigger.cs:

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
public class parentTrigger : MonoBehaviour
{
    public Player player;
    public bool stay;
    private void OnTriggerEnter2D(Collider2D collision)
    {
        if (collision.gameObject.layer == 8)
        {
            stay = true;
            player.canParent = true;
        }
    }
    private void OnTriggerStay2D(Collider2D collision)
    {
        if (collision.gameObject.layer == 8)
        {
            stay = true;
            player.canParent = true;
        }
    }
    private void OnTriggerExit2D(Collider2D collision)
    {
        if (collision.gameObject.layer == 8)
        {
            stay = false;
            player.canParent = false;
        }
    }
}
```

Respowned.cs:

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
public class respowned : MonoBehaviour
{
    public bool resp = false;
}
```

Score.cs:

```
using System.Collections;
using System.Collections.Generic;
using TMPro;
using UnityEngine;
public class Score : MonoBehaviour
{
```

```
    public TextMeshPro menuScore;
    public int levelsCount;
    public int score;
    void Start()
    {
        UpdateScore();
    }
    public    void UpdateScore()
    {
        score = 0;
        for (int i = 1; i <= levelsCount; i++)
        {
            if (PlayerPrefs.HasKey("Score" + i))
            {
                score += PlayerPrefs.GetInt("Score" + i);
            }
        }
        menuScore.text = score.ToString();
    }
}
```

SingleButton.cs:

```
using System.Collections;
using System.Collections.Generic;
using TMPro;
using UnityEngine;
public class singleButton : MonoBehaviour
{
    public KeyCode button;
    TextMeshPro key;
    void Start()
    {
        key = this.GetComponent<TextMeshPro>();
    }
    void Update()
    {
        if (Input.GetKey(button) && !PauseMenu.GameIsPaused)
        {
            key.color = Color.red;
        }
        else if (Input.GetKeyUp(button))
        {
            key.color = new Color(217, 217, 217);
        }
    }
}
```

HUD.cs:

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

```
public class HUD : MonoBehaviour
{
    public static HUD Instance { get; private set; }
    public Text healthText;
    public Image[] hearts;
    public Sprite fullHeart;
    public Animator animator;
    public Text velocity;
    public Text stopwatch;
    public Text likes;
    public Text dislikes;
    public float seconds, miliseconds, minutes;
    public float speed;
    private void Awake()
    {
        if (Instance == null)
        {
            Instance = this;
        }
        else
        {
            Destroy(gameObject);
        }
    }
    public void setHP(int hp)
    {
        healthText.text = "HEALTH     : " + hp.ToString();
        for (int i = 0; i < hearts.Length; i++)
        {
            if (i < hp)
                hearts[i].enabled = true;
            else
                hearts[i].enabled = false;
        }
    }
    public void like()
    {
        likes.text = (int.Parse(likes.text) + 1).ToString();
    }
    public void dislike()
    {
        dislikes.text = (int.Parse(dislikes.text) + 1).ToString();
    }
    public void fadeOut()
    {
        animator.Play("UI_FadeOut");
    }
    public void fadeIn()
    {
        animator.Play("UI_FadeIn");
    }
    public void setVelocity(float x, float y, float drag)
```

```
    {
        velocity.text = "X= " + x.ToString() + "\nY= " + y.ToString() + "\nD: " +
drag.ToString();
    }
    private void Update()
    {
        stopwatch.text = minutes.ToString("00") + ':' + seconds.ToString("00") + ':' +
miliseconds.ToString().Substring(2);
    }
    private void FixedUpdate()
    {
        miliseconds += 0.02f;
        if(miliseconds >= 1)
        {
            seconds++;
            miliseconds = 0.0000001f;
        }
        if(seconds >= 60)
        {
            minutes++;
            seconds = 0;
        }
    }
}
```

buttonTip.cs:

```
using System.Collections;
using System.Collections.Generic;
using TMPro;
using UnityEngine.EventSystems;
using UnityEngine;
public class buttonTip : MonoBehaviour, IPointerEnterHandler, IPointerExitHandler {
    public TextMeshPro score;
    public int level;
    public string highscore;
    public void OnPointerEnter(PointerEventData eventData)
    {
        if (PlayerPrefs.HasKey("Score" + level))
        {
            score.text = PlayerPrefs.GetInt("Score" + level).ToString();
            score.color = Color.green;
        }
        else
        {
            score.text = 0.ToString();
            score.color = Color.green;
        }
    }
    public void OnPointerExit(PointerEventData eventData)
    {
        score.text = highscore;
        score.color = Color.white;
```

```
    }
    private void Start()
    {
        Invoke("high", 0.0001f);
    }
    void high()
    {
        highscore = score.text;
    }
}
```

LoadMenu.cs:

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine.SceneManagement;
using UnityEngine;
public class loadMenu : MonoBehaviour
{

    public GameObject mainMenu;
    public GameObject levelsMenu;
    public void showLevels()
    {
        mainMenu.SetActive(false);
        levelsMenu.SetActive(true);
    }
    public void showMenu()
    {
        mainMenu.SetActive(true);
        levelsMenu.SetActive(false);
    }
    public void startLevel(int index)
    {
        SceneManager.LoadScene(index);
    }
}
```

MainMenu.cs:

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine.SceneManagement;
using UnityEngine;
using UnityEngine.UI;
using UnityEngine.EventSystems;
public class MainMenu : MonoBehaviour
{
    public GameObject menuFirstButton, levelsFirstButton;
    public GameObject mainMenu, levelsMenu; public
    Animator anim;
    public Button[] bLevels;
    public Score scr;
    public buttonTip[] bTip;
```

```
    int maxLevels = 3;
    private void Start()
    {
        EventSystem.current.SetSelectedGameObject(null);
        EventSystem.current.SetSelectedGameObject(menuFirstButton); for
        (int i = 0; i <= bLevels.Length && i < maxLevels - 1; i++) {
            if (PlayerPrefs.GetInt("aLevel" + (i + 2)) == 1)
            {
                bLevels[i].interactable = true;
            }
        }
    }
    public void resetAll()
    {
        PlayerPrefs.DeleteAll();
        scr.UpdateScore();
             for (int i = 0; i < bTip.Length; i++)
               bTip[i].highscore = 0.ToString();
             for (int i = 0; i < bLevels.Length; i++)
                    bLevels[i].interactable = false;
    }
    public void showLevels()
    {
        EventSystem.current.SetSelectedGameObject(null);
        EventSystem.current.SetSelectedGameObject(levelsFirstButton);
        //EventSystem.current.currentSelectedGameObject.gameObject.GetComponent<Anim
        a-
tor>().Play("Selected");
        mainMenu.SetActive(false);
        levelsMenu.SetActive(true);
    }
    public void showMenu()
    {
        EventSystem.current.SetSelectedGameObject(null);
        EventSystem.current.SetSelectedGameObject(menuFirstButton);
        mainMenu.SetActive(true);
        levelsMenu.SetActive(false);
    }
    public void startLevel(int index)
    {
        SceneManager.LoadScene(index);
    }
    public void Quit()
    {
        Application.Quit();
    }
    private void Update()
    {
        if (Input.GetAxis("Mouse X") != 0 || Input.GetAxis("Mouse Y") != 0)
        {
            EventSystem.current.SetSelectedGameObject(null);
        }
```

```
        if(Input.GetAxis("Vertical") > 0 && EventSystem.current.currentSelectedGameOb-
ject == null)
        {
            if (mainMenu.active)
            {
                EventSystem.current.SetSelectedGameObject(null);
                EventSystem.current.SetSelectedGameObject(menuFirstButton);
            }
            if (levelsMenu.active)
            {
                EventSystem.current.SetSelectedGameObject(null);
                EventSystem.current.SetSelectedGameObject(levelsFirstButton);
            }
        }
        if(Input.GetButton("Cancel") && levelsMenu.active)
        {
            showMenu();
        }
    }
}
```

PauseMenu.cs:

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.EventSystems;
using UnityEngine.SceneManagement;
public class PauseMenu : MonoBehaviour
{
    public static bool GameIsPaused = false;
    public GameObject pauseMenuUI, finishMenuUI,
    controlsMenuUI; public GameObject firstButton, controlsResume;
    void Update()
    {
        if (Input.GetButtonDown("Pause"))
        {
            if (GameIsPaused && !finishMenuUI.active)
            {
                Resume();
            }
            else if (!finishMenuUI.active)
            {
                Pause();
            }
        }
        if (GameIsPaused && Input.GetButton("Cancel") && !Finish.levelFinished)
        {
            Resume();
        }
        if (Input.GetAxis("Mouse X") != 0 || Input.GetAxis("Mouse Y") != 0)
        {
            EventSystem.current.SetSelectedGameObject(null);
```

```
        }
        if (Input.GetAxis("Vertical") > 0 && EventSystem.current.currentSelectedGameOb-ject
== null)
        {
            if (pauseMenuUI.active)
            {
                EventSystem.current.SetSelectedGameObject(null);
                EventSystem.current.SetSelectedGameObject(firstButton);
            }
            if (controlsMenuUI.active)
            {
                EventSystem.current.SetSelectedGameObject(null);
                EventSystem.current.SetSelectedGameObject(controlsResume); } } }
    public void Resume()
    {
        pauseMenuUI.SetActive(false);
        controlsMenuUI.SetActive(false);
        Time.timeScale = 1f;
        Invoke("res", 0.1f);
    }
    void res()
    {
        GameIsPaused = false;
    }
    public void Pause()
    {
        pauseMenuUI.SetActive(true);
        Time.timeScale = 0f;
        GameIsPaused = true;
        EventSystem.current.SetSelectedGameObject(null);
        EventSystem.current.SetSelectedGameObject(firstButton);
    }
    public void Menu()
    {
        SceneManager.LoadScene("Menu");
        Time.timeScale = 1f;
        GameIsPaused = false;
    }
    public void Reset() {
        SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex);
        Time.timeScale = 1f;
        pauseMenuUI.SetActive(false);
        GameIsPaused = false; }
    public void Controls()
    {
        pauseMenuUI.SetActive(false);
        controlsMenuUI.SetActive(true);
        EventSystem.current.SetSelectedGameObject(null);
        EventSystem.current.SetSelectedGameObject(controlsResume);
    }
    public void QuitGame() {
        Application.Quit(); }
```

```
public void Next()
{
    SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex + 1);
    Time.timeScale = 1f;
    GameIsPaused = false; } }
```

ДОДАТОК Б

ВІДГУК КЕРІВНИКА ЕКОНОМІЧНОГО РОЗДІЛУ

ДОДАТОК В

ПЕРЕЛІК ФАЙЛІВ НА ДИСКУ

Ім’я файлу	**Опис**
Пояснювальні документи	
Диплом_ doc	Пояснювальна записка до кваліфікаційної роботи. Документ Word.
Диплом_.pdf	Пояснювальна записка до кваліфікаційної роботи в форматі PDF
Програма	
Program.rar	Архів. Містить коди програми і откомпільовану програму
Презентація	
Презентація_.ppt	Презентація кваліфікаційної роботи