Міністерство освіти і науки України
Національний технічний університет
«Дніпровська політехніка»

Інститут електроенергетики
(інститут)

Факультет інформаційних технологій
(факультет)

Кафедра Програмного забезпечення комп'ютерних систем
(повна назва)

ПОЯСНЮВАЛЬНА ЗАПИСКА
кваліфікаційної роботи ступеня

бакалавра
(назва освітньо-кваліфікаційного рівня)

студента *Бруя Владислава Валерійовича*
(ПІБ)

академічної групи *122-20-3*
(шифр)

спеціальності *122 Комп'ютерні науки*
(код і назва спеціальності)

освітньої програми *Комп'ютерні науки*
(назва освітньої програми)

на тему: *Розробка кросплатформеної гри Jumping cube в жанрі платформера на базі GodotEngine та мови програмування GDScript*

Керівники	**Прізвище, ініціали**	**Оцінка за шкалою**		**Підпис**
		рейтинговою	**інституційною**	
кваліфікаційної роботи	*проф. Бердник М.Г*			
розділів:				
спеціальний	*проф. Бердник М.Г*			
економічний	*доц. Касьяненко Л.В.*			
Рецензент				
Нормоконтролер	*доц. Гуліна І.Г.*			

Дніпро
2024

Міністерство освіти і науки України
НТУ «Дніпровська політехніка»

ЗАТВЕРДЖЕНО:
завідувач кафедри
програмного забезпечення комп'ютерних систем
(повна назва)

М.О. Алексєєв
(підпис) (прізвище, ініціали)

« » 2024 року

ЗАВДАННЯ
на кваліфікаційну роботу

бакалавра
(назва освітньо-кваліфікаційного рівня)

студента *122-20-3* (група) *Бруй Владислав Валерійович* (прізвище та ініціали)

тема кваліфікаційної роботи *Розробка кросплатформеної гри Jumping cube в жанрі платформера на базі GodotEngine та мови програмування GDScript*

затверджена наказом ректора НТУ «ДП» від *23.05.2024* р. № *469-с*

Розділ	Зміст виконання	Термін виконання
Спеціальний	***На основі матеріалів виробничої практики та інших науково-технічних джерел провести аналіз стану рішення проблеми та постановку задачі. Обґрунтувати вибір та здійснити реалізацію методів вирішення проблеми***	***01.06.2024 р.***
Економічний	***Провести розрахунок трудомісткості розробки програмного забезпечення, витрат на створення ПЗ й тривалості його розробки***	***06.06.2024 р.***

Завдання видав _____ *проф. Бердник М.Г.*
(підпис) (посада, прізвище, ініціали)

Завдання прийняв до виконання _____ *Бруй В.В.*
(підпис) (прізвище, ініціали)

Дата видачі завдання: *14.01.2024 р.*

Термін подання кваліфікаційної роботи до ЕК: *10.06.2024 р.*

РЕФЕРАТ

Пояснювальна записка: 93 с., 30 рис., 6 табл., 3 дод., 25 джерел.

Об'єктом розробки є кроссплатформенний програмний додаток гри Jumping cube в жанрі платформера на базі GodotEngine та мови програмування GDScript.

Мета кваліфікаційної роботи розробити кросплатформенну гру «Jumping cube» в жанрі аркада на мові програмування GDScript.

У введенні представлений аналіз поточного стану проблеми, визначені мета кваліфікаційної роботи та область її застосування, обґрунтована актуальність теми та сформульовані завдання.

У першому розділі проведено дослідження предметної галузі, визначено актуальність завдання та призначення розробки, сформульовано постановку завдання та встановлено вимоги до програмної реалізації, технологій та програмних засобів.

Другий розділ присвячено аналізу існуючих рішень, вибору платформи для розробки, проектуванню та розробці програми. Описано алгоритм та структуру функціонування програми, визначено вхідні та вихідні дані, наведено характеристики технічних засобів та описано процес виклику та завантаження програми.

Економічний розділ визначає трудомісткість розробленої інформаційної системи, проведено розрахунок вартості роботи зі створення програми та визначено час, необхідний для її розробки.

Практичне завдання полягає в створені кроссплатформенної гри, яка буде відповідати сучасним стандартам і буде чудовою розвагою під час зачекання, а також допомогатиме у розвитку реакції.

Актуальність даної розробки полягає в можливості розвивати моторику рук, а також зорову і рухову реакції. Скорочення часу під час очікування дозволяє залишатися в гарному настрої та тонусі.

Список ключових слів: КРОССПЛАТФОРМЕР, АРКАДА, 2D, GDSCRIPT, GODOT

ABSTRACT

Explanatory note: 93 p., 30 figures, 6 tables, 3 appendixes, 9 sources.

Object of development: a cross-platform game based on GodotEngine and the GDScript programming language.

The purpose of the qualification work is to develop a cross-platform game "Jumping cube" in the arcade genre using the GDScript programming language.

In the introduction, an analysis of the current state of the problem is provided, the purpose of the qualification work and its application area are defined, the relevance of the topic is justified, and the task is set.

In the first section, the research of the subject area is conducted, the relevance of the task and the purpose of the development are determined, the task is formulated, and the requirements for software implementation, technologies, and software tools are established.

The second section is devoted to the analysis of existing solutions, the choice of platform for development, design, and program development. The algorithm and structure of the program's functioning are described, the input and output data are defined, the characteristics of technical means are provided, and the process of calling and loading the program is described.

In the economic section, the labor intensity of the developed information system is determined, the cost of creating the program is calculated, and the time required for its development is determined.

The practical task is to create a cross-platform game that will meet modern standards and be an excellent entertainment during waiting, as well as help in developing reaction.

The relevance of this development lies in the ability to improve hand motor skills, as well as visual and motor reactions. Reducing waiting time helps to stay in a good mood and tone.

List of keywords: CROSSPLATFORMER, ARCADE, 2D, GDSCRIPT, GODOT

ЗМІСТ

ВСТУП

На сьогоднішній день комп’ютерні ігри на «Android» та «IOS» вже набрали великої популярності. Близько 40% населення планети грають в відеоігри, а якщо брати до уваги ще тих, хто коли небудь хоч раз пробував грати, то цифра стане ще більшою. З цього можна зробити висновок що тема відеоігор є достатньо актуальною сьогодні. Велика кількість компаній проводить співбесіди на посаду розробника ігор. Серед них такі відомі компанії як «Ubisoft», «CD Project Red», «Valve», «Sony Computer Entertainment» та деякі українські компанії: «Absolutist», «4A Games» та інші[25].

Всі ігри поділяються на жанри: аркада, пригоди, стратегії, логічні, казуальні, RPG, MMORPG, MOBA тощо. Одним з найпопулярніших і найактуальніших є жанр аркада. Ігри в цьому жанрі існували в будь-яку епоху історії створення відеоігор. Аркада один із найдавніших жанрів, хоча по сьогоднішній день велика кількість людей обирає ігри саме цього виду. Річ у тому, що такі відеоігри не потребують багато вільного часу, в них можна грати коли потрібно дочекатися громадський транспорт або чергу в магазині. Ігри в жанрі аркада розробляють як на платформи «Windows», «Linux» так і на «Android», «IOS».

Окрім цього відеоігри можна розділити за типом графіки, а саме 2D і 3D. 3D ігри набрали більше попиту ніж 2D, однак для створення 3D графіки, механіки та можливостей потрібно набагато більше навичок та ресурсів, обсяг роботи в цьому випадку в рази більший. 2D ігри легше створювати, їх може розробити навіть одна людина. Проте їх функціонал та зацікавленість користувачів в них грати не менше.

Аркада - поширений в індустрії відеоігор термін, що позначає ігри з навмисно примітивним ігровим процесом. Деякі ресурси про відеоігри виділяють їх як окремий жанр і зараховують до них платформери[1]. Платформер - жанр комп'ютерних ігор, у яких ігровий процес становить стрибки по платформах, лазіння сходами, збирання предметів, необхідні

перемоги над ворогами чи завершення рівня. Платформери з'явилися на початку 1980-х і стали тривимірними ближче до кінця 1990-х. Через деякий час після утворення жанру у нього з'явилася назва. Коли ігрові консолі не були достатньо потужними, щоб відображати тривимірну графіку або відео, вони були обмежені статичними ігровими світами, які містилися на один екран. Персонаж лазив вгору і вниз сходами або стрибав з платформи на платформу, часто борючись з противниками і збираючи предмети, що покращують характеристики[2].

Метою дипломної роботи є створення проєкту 2D гри в жанрі аркада під назвою «Jumping cube». Основною ідеєю гри є проходження рівнів перестрибуючи через перешкоди, збір монет за які можна купувати нових персонажів та задній фон, а також покращення навичок швидкості сприйняття ситуації та вміння прийняти миттєве рішення.

2D гра створюється за допомогою ігрового двигуна з відкритим кодом – «Godot Engine» з основною мовою програмування «GD Script». Проте існує інша версія цього двигуна, що підтримує мову «C#» та декілька інших мов.

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1. Загальні відомості з предметної галузі

Для створення гри було використано кросплатформений двигун «Godot Engine» (рис. 1.1). Переваги двигуна полягають в тому що, по перше він простий та зрозумілий, по друге він не потребує установки, тобто потрібно лише завантажити файл з розширенням «.exe» та запустити його, по третє архів з файлами на оффіціальному сайті займає лише 65 Мб на диску.

Окрім всього цього двигун абсолютно безкоштовний. Може бути використаний на таких ОС як «Linux», «Windows», «OS X» та інші. Не взираючи на його ніби-то прості умови використання, його функціонал та можливості майже не поступаються таким двигунам-гігантам як «Unity 3D» та «Unreal Engine 4». Може експортувати готові проєкти на ПК, будь-які мобільні ОС, консолі та веб-платформи.

Двигун «Godot» має відкритий код написаний мовою «C++». Завдяки цьому можна подивитися ієрахію та працездатність всіх класів, а також всіх фукцій. Можна з легкістю визначити будь-яке виключення, опис якого присутній в компіляторі. Двигун призначений як для розробки 2D проєктів так і 3D, навіть має властний конструктор для створення користувацького інтерфейсу.

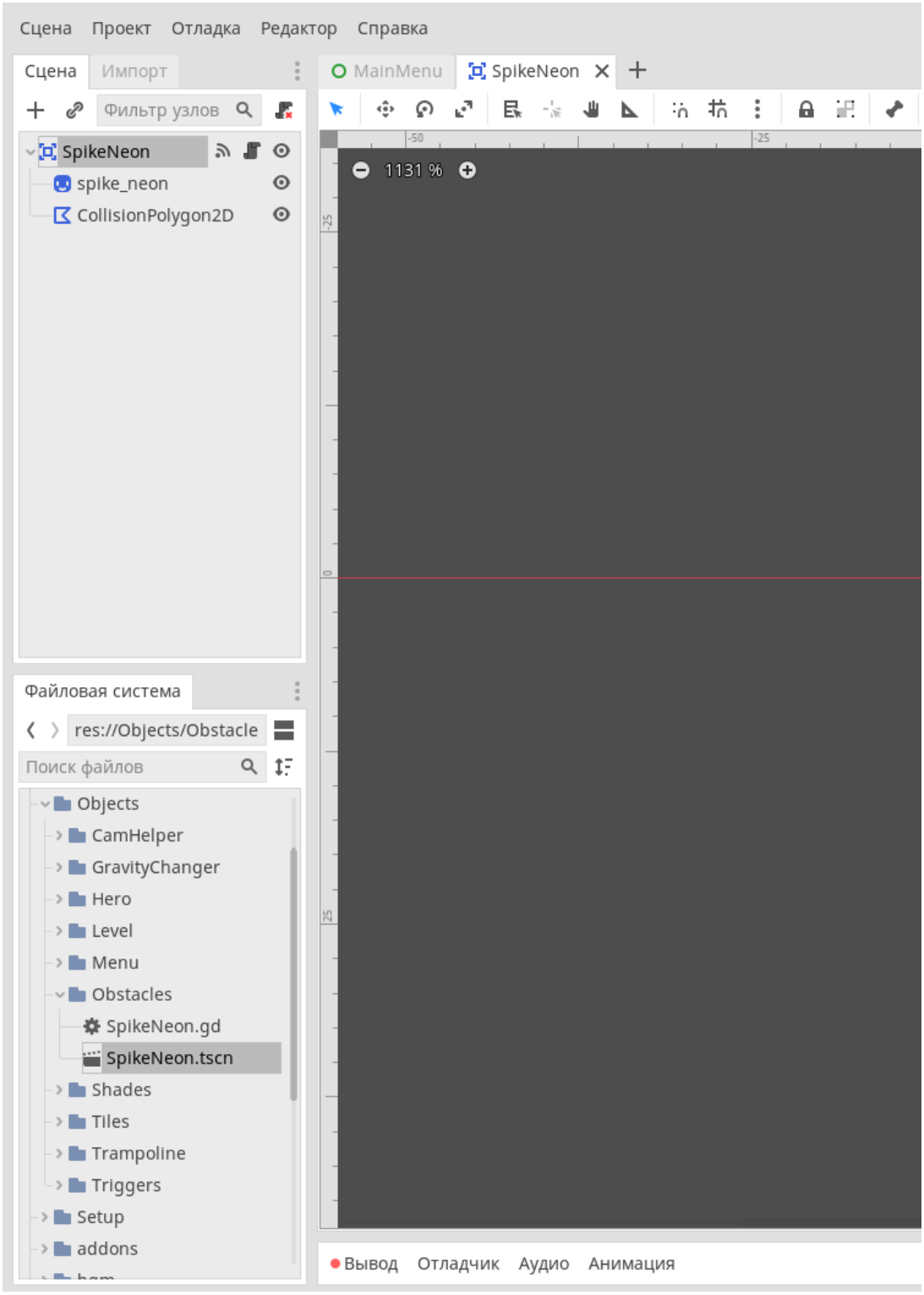

Рис. 1.1. Зовнішній вигляд двигуна «Godot Engine» (новий проєкт)

Основні елементи для створення гри це сцени і вузли. Сцени бувають трьох типів: 2D, 3D і користувацький інтерфейс. А вузлів існує велика кількість. Основними 2D і 3D вузлами є ті, за допомогою яких створюється головний персонаж, платформа або приміщення, яка має власну колізію,

додаткові елементи наприклад монети або бустери, додаткові персонажі які мають примітивний штучний інтелект та інше (рис. 1.2, 1.3). Основними вузлами в користувацькому інтерфейсі являються вузли текстур, контейнери та кнопки. Це базові вузли без яких не може створюватися будь яка гра. Такі вузли як програвач музики та відео, вузол котролю (управління), вузол навігації, вузол для створення світла та тіні, а також сторонні вузли які не можна віднести до вище перелічених трьох тимів, наприклад таймер, програвач анімації, вузол завантаження ресурсів та інші (рис. 1.4).

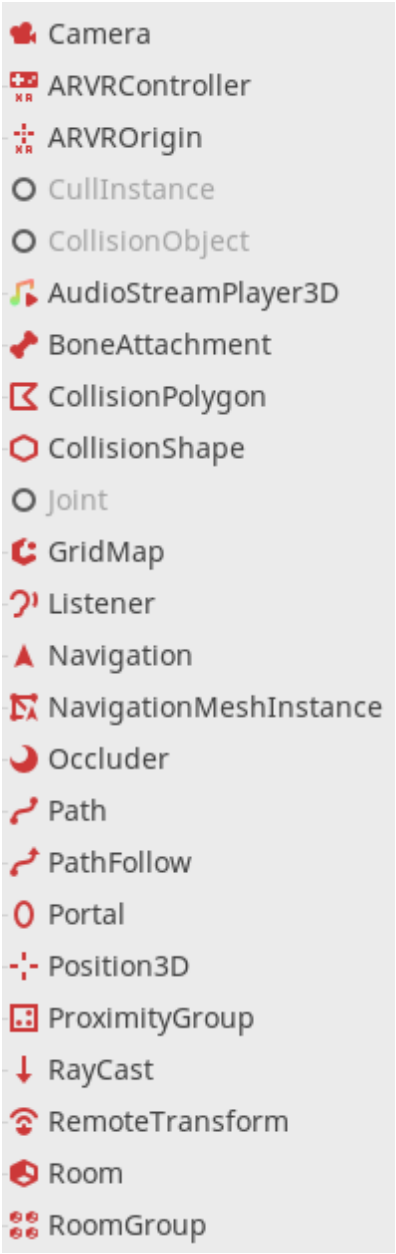

Рис. 1.2. Вузли типу 3D

Рис. 1.3. Вузли типу 2D

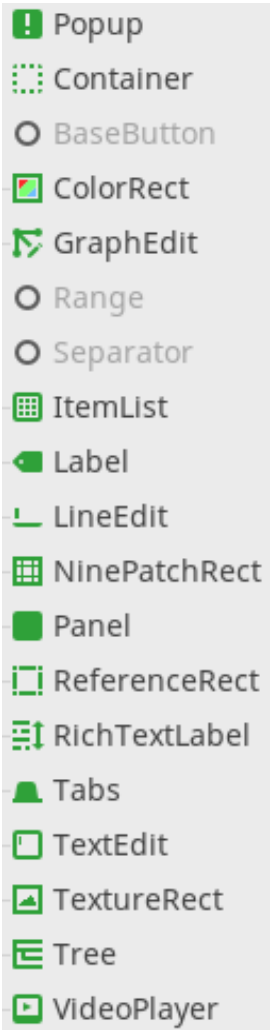

Рис. 1.4. Вузли користувацького інтерфейсу

Основна мова програмуваня двигуна «Godot Engine» це – «GDScript». Однак існує інша версія цього двигуна яка підтримує мову «C#» та декілька інших мов. Двигун має можливість інтегрувати скриптинг таких мов як «C++», «Python», «Lua» та інші. Різниця полягає лише в зручності самого розробника, тобто якщо розробник добре розуміє мову «C#», то без проблем може використовавати саме цю мову, функціональність та можливості двигуна ніяк не зміняться.

«GDScript» – високорівнева мова програмування, яка за своїм синтаксимом та стилем кодування схожа на мову «Python». Була створенами спеціально для двигуна «Godot Engine». Досить легка в вивченні та кодуванні. Також саме для цієї мови були створенні документації та посібники,

найбільша кількість відео та матеріалу для вивчення створено саме для мови «GDScript». Розробники «Godot Engine» рекомендують використати саме її томущо двигун повністтю адаптовано і сбалансовано для цієї мови. Среда розробки працює стабільно і без виключень[3].

До кожної сцени, а також вузла можна приєднати скрипт (рис. 1.5).

```
1  extends Control
2
3  const ACCOUNT_LOGIN = "account_login_"
4  const ACCOUNT_PASSWORD = "account_password_"
5
6  var nextScene
7
8  func _process(delta):
9      $AboutAccounts/CurrentAccount.text = "Current account: " + globals.login
10     $AboutAccounts/AllTime.text = "Your game time: " + String(int(globals.timer))
11
12 func _ready():
13     $BackgroundVideoPlayer.modulate = globals.color
14     $Textures/NinePatchRect.texture = globals.bg
15     $Textures/NinePatchRect2.texture = globals.bg
16     $Textures/NinePatchRect3.texture = globals.bg
17     $Textures/NinePatchRect4.texture = globals.bg
18     $backgroundMusic.volume_db = globals.volumeMenu
19
20 func _on_ButtonPlay_pressed():
21     if !globals.login:
22         $Warnning.show()
23     else:
24         nextScene = load("res://Objects/Menu/LevelsMenu.tscn").instance()
25         get_tree().current_scene.add_child(nextScene)
26
```

Рис. 1.5. Приклад скрипта на мові «GDScript»

Скрипт – це внутрішній (прікріпленний за вузлом і не має власного місця в файловій системі) або зовнішній (може існувати сам по собі, може бути не прікрипленний до жодної сцени чи вузла, або може бути прікріпленний до декількох сцен або вузлів) файл, який містить в собі опис дій або логіки певних сцен(и) або вузлів.

Може бути написанний на будь якій мові, при створенні нового скрипта існує можливість обрати мову программування (рис. 1.6). Має функцію внутрішього скрипта (вбудованого). Має власний шлях у файловій системі (якщо обраний прапорець в пункті «вбудований скрипт», то поле з шляхом стає не активним). При ствоерннні скрипта він повинен наслідуватися від сцени або вузла, можна використати базовий (шаблонний класс який існує в двигуні), або, як це зазвичай відбувається, можна наслідувати від сцени або вузла яка була створена. В цьому випадку компоненти і властивості будуть саме тієї сцени до якої прікріпленний скрипт. Також має параметр шаблону – це параметр який відповідає за регенерацію коду одразу після створреня скрипта. Їх існує 3 вида: за замовчуванням, без коментарів і пустий. За замовчування створює наслідування, базові функції і коментарі[21].

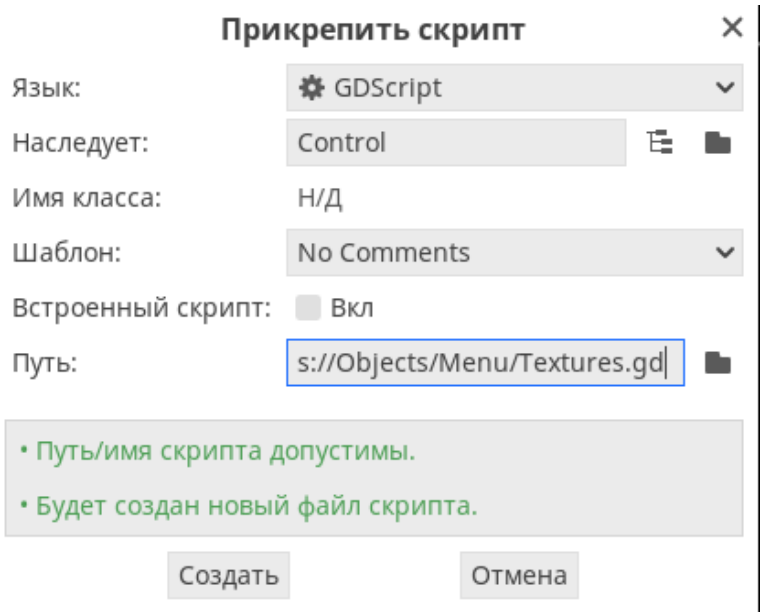

Рис. 1.6. Створення скрипта

Після створення скрипта його можна змінювати. Можна використовувати передбачені двигуном класси або функції, а також змінювати властивості тих об’єктів які існують в сцені до якої прікрипленний скрипт або звертатися напряму до об’єктів. Також в проєкті існують глобальні скрипти. Це скрипти які запускаються одразу після запуску програми лише 1 раз.

Завдяки їм з’являється можливість передавати дані між об’єктами або викликати певну функцію в будь-якому місті в програмі[3].

1.2. Призначення розробки та галузь застосування

Розробка гри має на меті розвиток реакції, а також розважальний характер. Цей додаток буде забезпечувити зручний та інтуітивно-зрозумілий інтерфейс для надання доступу короистувачеві.

1.3. Підстава для розробки

Підставами для розробки та виконання кваліфікаційної роботи є:

- освітня програма «Комп’ютерні науки»;
- навчальний план та графік навчального процесу;
- наказ ректора Національного технічного університету «Дніпровська політехніка» № 469-с від 23.05.2024 р;
- завдання на кваліфікаційну роботу на тему «Розробка кросплатформеної гри Jumping cube в жанрі платформера на базі GodotEngine та мови програмування GDScript».

1.4. Постановка завдання

Розробити 2D гру в жанрі аркада під назвою «Jumping cube». Необхідно змоделювати меню та функціонал гри в стилі 2D в жанрі аркада під різні платформи.

Вхідними даними є: ім’я і пароль користувача для реєстрації або авторизації.

Вихідними даними є: вивід монет користувача та рейтингу (час гри, час проходження рівня, кількість спроб).

1.5. Вимоги до програми або програмного виробу

1.5.1. Вимоги до функціональних характеристик

Гра створюється для проходження в одиночному режимі декількома гравцями під різними аккаунтами. Метою гри є подолання перешкод кожного рівня за найменшу кількість спроб та досягнення їх кінця, збір уламків кристалу водночас керуючи персонажем, а також придбанням нових досягнень в меню магазину[15].

Персонаж - ігровий персонаж представляє собою куб певних розмірів з кольоровою текстурою, яку можна змінювати.

Керування головним героєм відбувається дотиком до екрана якщо це мобільний пристрій та натиском на клавішу «Пробіл» якщо пристрій має фізичну клавіатуру або ліва кнопка миші якщо є фізична миша (рис. 1.7). Переміщується персонаж у двомірному просторі тільки вперед (з ліва на право), при натисканні на клавішу «Пробіл» або ліва кнопка миші, або дотик до екрана персонаж ніби підстрибує (відображається анімація перевертання, персонаж переміщається в гору, паралельно переміщаючись уперед, а після досягнення певної висоти спускається в низ до тих пір поки не торкнеться поверхні). Якщо персонаж доторкнеться до перешкоди, то починає відображатися анімація розпаду на частини і він повертається на старт. Нараховується одна спроба. Таймер рівня прирівнюється до нуля[19].

Рис. 1.7. Засоби управлніння

Платформа - статичний об’єкт, який нагадує є образ поверхні по якому можна переміщуватися. Має виступи та впадини. Якщо персонаж знаходиться на ній він просто переміщується вперед.

Перешкода - статичний або динамічний об’єкт, який нагадує образ шипа або лазера. Якщо персонаж доторкнеться до такого об’єкта, то головний герой повертається на старт і рівень потрібно проходити з початку.

Тригери - об’єкти які ускладнюють та урізноманітнюють процес гри. Мають різні образи та функціонал:

- трамплін це об’єкт який має образ платформи з анімацією розпаду на частинки, працює як звичайний стрибок, тільки з більшою висотою коли персонаж доторкається до нього;

- зміна кольору це об’єкт який не має образу, але якщо персонаж пройде через нього, то певна частина платформи змінить колір;

- зміна гравітації це об’єкт який не має образу, але якщо персонаж пройде через нього, то починає переміщуватися по верхній частині платформи;

- фініш це об’єкт який має текстуру фінішу (чорно-білі квадрати),

Уламок кристалу - об’єкт який має образ дорогоцінного каменю з анімацією обертання. Потрібен для того щоб купувати нові образи, наприклад образ головного героя або платформи. Збирається при проходженні рівнів протягом усієї гри.

Інтерфейс – об’єкт, який складається з декількох вікон, перемикання між якими відбувається наступними кнопками:

- «Play» дозволяє перейти до додаткового меню, за потреби обрати рівень, також містить ще 2 кнопки, одна з них починає саму гру, інша повертає до головного меню;

- «Shop» дозволяє перейти в додаткове меню, де можна купити за внутрішньоігрову валюту нові образи для головного персонажа та для інших об’єктів, також має 2 кнопки для покупки обраного образу та для повернення до головного меню;

- «Options» дозволяє перейти до меню з налаштуваннями для гри такими як: гучність ефектів музики;

- «Exit» повністю завершує роботу гри.

В грі наявний задній фон для покращення візуального сприйняття.

1.5.2. Вимоги до інформаційної безпеки

Вимоги до інформаційної безпеки гри є важливою складовою його розробки та експлуатації. Для забезпечення безпеки додатку і захисту важливих даних від несанкціонованого доступу та зловживань, необхідно враховувати наступні вимоги:

- використання лише ліцензійного софту;
- реалізована протидія несанкціонованому доступу;
- сервер захищений від аварійного вимикання та перепадів напруги.

1.5.3. Вимоги до складу та параметрів технічних засобів

Створена гра працює самостійно без додаткових програм та додатків. Розглянемо основні вимоги до ПК та смартфона на яких відбувається гра.

Мінімальні вимоги до ПК:

- ОС Windows 10;
- процесор: Intel Pentium III 1200 / AMD Athlon XP 1400 з мінімальною частотою 1.2GHz;
- оперативна пам’ять 1 Гб;
- відеокарта Nvidia Geforce 440 / AMD Radeon HD 5670 на 1 Гб видео-пам’яті;
- близько 100 Мб фізичної пам’яті;
- роздільна здатність екрану 600 на 800 пікселів.

Рекомендовані вимоги до ПК:

- ОС Windows 10;
- процесор: Intel Pentium g3000/ AMD A4 2000 з мінімальною частотою 2.7GHz[6];
- оперативна пам’ять 2 Гб;
- відеокарта Nvidia Geforce 550 / AMD Radeon HD 4550 на 2 Гб відео-памяті[7];
- близько 100 Мб фізичної пам’яті;

- роздільна здатність екрану 600 на 800 пікселів.

Мінімальні вимоги до мобільного пристрою:

- ОС Android 8.0;
- процесор: Qualcomm Snapdragon 845;
- оперативна пам’ять 1 Гб;
- близько 100 Мб фізичної пам’яті;
- роздільна здатність екрану 480 на 800 пікселів.

Рекомендовані вимоги до мобільного пристрою:

- Android 8.0;
- процесор: Qualcomm Snapdragon 630;
- оперативна пам’ять 2 Гб;
- близько 100 Мб фізичної пам’яті;
- роздільна здатність екрану 720 на 1280 або більше пікселів[5].

1.5.4. Вимоги до інформаційної та програмної сумісності

Розроблена гра не містить в собі потужну графіку, тим більш графіка має 2D стиль, тому потребує ще менш затрат на обробку графічних елементів. Не містить в собі дуже гнучкого і різноманітного функціоналу, також обробляє досить не велику кількість процесів. Проєктування рівнів, головного героя та перешкод буде оптимізовано для зменшення об’єму гри, тому для гри вистачить приблизно 200 Мб пам’яті як на ПК на і на смартфони.

Для нормального функціонування гри необхідно графічний (відеокарта або процесор з відеопам’яттю) та звуковий (динамік або навушники для мобільного пристрою та стереосистема, навушники, динамік монітору та інше для ПК) адаптер. Такі адаптери не повинні бути професіональними. Достатньо простих пристроїв які здатні виводити графіку на екран та відтворювати звук.

РОЗДІЛ 2
ПРОЄКТУВАННЯ ТА РОЗРОБКА ПРОДУКТУ

2.1. Функціональне призначення програми

Результатом кваліфікаційної роботи має бути 2D гра, яка надає користувачу зручний доступ до усіх її функцій.

Після запуску гри відкривається вікно головного меню (рис. 2.1, 2.2). У верхній чатсині головного вікна знаходяться назва та логотип гри. З правої частини вібодражается статистика гравця.

Головна сторінка містить кнопки для переходу на ішні вікна, також кнопка реєстрації та виходу з гри:

- інформація про розробника;

- кнопку «Авторизації», яка має рисунок та дозволяє перейти до вікна регістрації/авторизації;

- кнопку «Play», яка надає можливість перейти до вікна меню рівнів, обрати рівень і розпочати гру;

- кнопку «Shop», яка надає можливість перейти до вікна меню магазиу, де можна змінити значок головного персонажа або тему заднього фону всієї гри;

- кнопку «Options», яка надає можливість перейти до вікна меню налаштувань гри, в якому є можливість вимкнути або змінити гучність у всьому меню чи в самій грі;

- кнопку «Quit», яка дає змогу повністю вийти з гри.

В нижній частині головного меню розташована версія гри та контактна інформація, а саме: контактний номер телефону, електрона пошта та посилання на соціальні мережі.

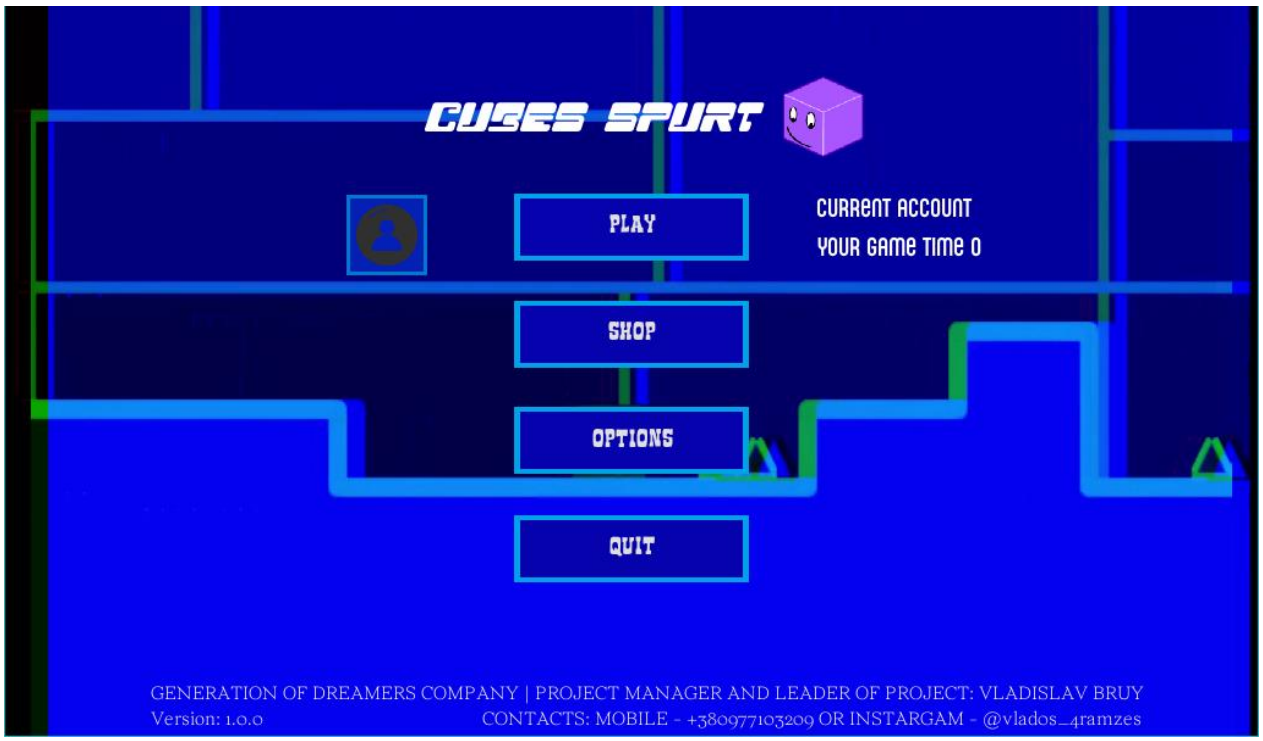

Рис. 2.1. Головне меню на ПК

Рис. 2.2. Головне меню на телефоні

2.2. Опис застосованих математичних методів

Для створення 2D гри можуть використовуватися різні математичні методи і алгоритми, які допомагають в розробці ігрової логіки, анімацій, фізики та інших аспектів:

1. Вектори і векторна алгебра: вектори використовуються для представлення позицій, напрямків і швидкостей об'єктів. Векторна алгебра допомагає в обчисленнях, пов'язаних з рухом і взаємодією об'єктів в просторі[16].

2. Тригонометрія: тригонометричні функції використовуються для обчислення кутів, поворотів і переміщень по колу. Вони допомагають в створенні плавних рухів і обертань об'єктів[17].

3. Перевірка зіткнень: методи перевірки зіткнень дозволяють визначити, чи перетинаються об'єкти в ігровому просторі. Це важливо для реалізації взаємодій між об'єктами, таких як зіткнення персонажів з перешкодами або противниками[18].

4. Інтерполяція і сплайни: інтерполяція і сплайни використовуються для створення плавних переходів і кривих траєкторій руху об'єктів. Лінійна інтерполяція (LERP) дозволяє плавно переходити між двома значеннями, а кубічні сплайни і криві Безьє забезпечують гладкі переходи і природні вигини[23].

Ці методи є основоположними для розробки 2D ігор і допомагають вирішувати різні завдання, пов'язані з рухом, взаємодією об'єктів і їх поведінкою.

2.3. Опис використаних технологій та мов програмування

Для розробки 2D гри були використані такі технології та інструменти, що забезпечують ефективну роботу та відповідність вимогам проекту:

1. Godot Engine - це потужне і гнучке середовище розробки ігор з відкритим вихідним кодом, яке надає всі необхідні інструменти для створення

2D та 3D ігор. Godot підтримує візуальне редагування, а також має широкий набір вбудованих інструментів для анімації, фізики та управління ресурсами. Godot Engine дозволяє швидко створювати прототипи та тестувати ігрові механіки, що суттєво прискорює процес розробки.

2. GDScript - це високорівнева динамічно типізована мова програмування, спеціально розроблена для використання в Godot Engine. GDScript має просту та зрозумілу синтаксичну структуру, що робить його ідеальним вибором для початківців розробників і тих, хто хоче швидко опанувати програмування ігор. GDScript використовується для написання ігрових скриптів, управління ігровими об'єктами, створення логіки гри та взаємодії з користувачем.

Для роботи над 2D грою використовувався наступний софт:

1. Операційна система Windows 10 - використання Windows 10 надає зручне та широко розповсюджене середовище для розробки ігор. Вона забезпечує доступ до різних інструментів та ресурсів, необхідних для розробки та тестування ігор[4].

2. Godot Editor - це вбудоване в Godot Engine середовище розробки, яке дозволяє створювати та редагувати ігрові сцени, скрипти, анімації та інші ресурси. Вона забезпечує зручний інтерфейс для роботи з ігровими проектами та підтримує різні плагіни та розширення для підвищення продуктивності.

3. Visual Studio Code - це популярний текстовий редактор, що підтримує багато мов програмування та надає розширені можливості для редагування коду. Він використовується для написання та редагування скриптів на GDScript, а також для роботи з іншими файлами проекту.

4. Git - це система контролю версій, що використовується для управління змінами в проекті. Вона дозволяє відстежувати історію змін, працювати в команді та керувати різними версіями проекту, що робить процес розробки більш організованим і контрольованим[24].

Ці технології та програмні засоби були використані для розробки 2D гри з метою забезпечення зручного та ефективного процесу розробки, тестування та впровадження проекту.

2.4. Опис структури програми та алгоритмів її функціонування

Проєкт складається з дерева файлів (рис 2.3). Коренева папка має назву «res://», після цього можна потрапити у будь-яку директорію або файл. Можна створювати, додавати та видаляти будь-які файли або папки. В кожний файл в будь-якій директорії можна дістатися програмно, потім завантажити дані файлу як об’єкт або зберегти повний шлях до файлу.

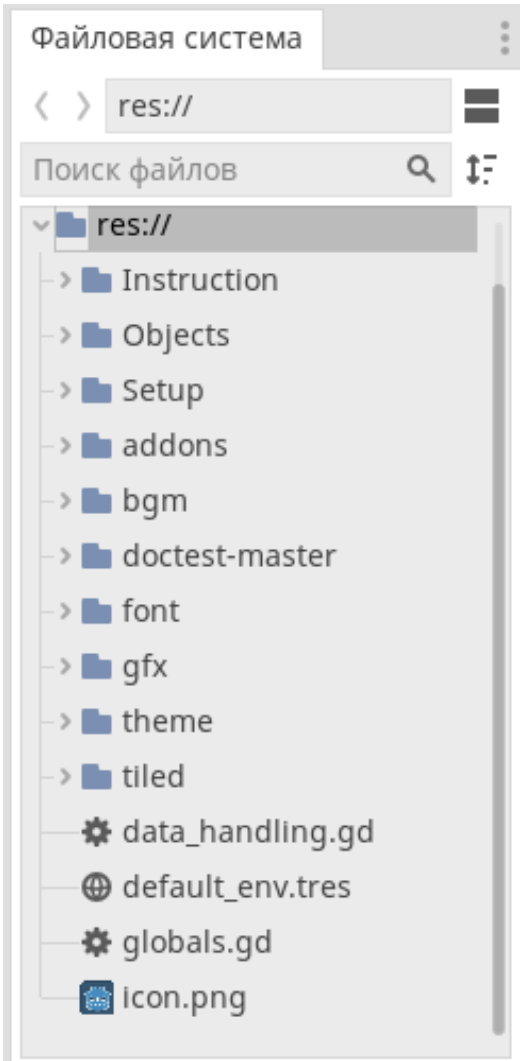

Рис. 2.3. Файлова структура проєкту

Опис основних файлів і директорій:

- «Objects», зберігає в собі всі готові та реалізовані об’єкти гри та скрипти до цих об’єктів (камера для персонажу, персонаж, рівні, перешкоди, тригери, основу рівня і вікна меню);

- «Setup», зберігає в собі файли для запуску гри без двигуна (окремо для ОС Windows і Android);

- «addons», зберігає скрипти пов’язані з основою рівня (зовнішній вигляд, функціонал, рендерінг та інше);

- «bgm», зберігає файли формату .mp3, тобто файли з музикою;

- «font», зберігає базові шрифти, на основі них створювався спеціальний шрифт для гри; зберігаються в цій же директорії;

- «gfx», зберігає ресурси і текстури всієї гри (монет, заднього фону, логотип, значків кнопок і персонажів, перешкод тощо);

- «theme», зберігає файли з розширенням .tres (файл створенний за допомогою Godot Engine), потрібен для зміни зовнішнього вигляду багатьох компонентів (в основному для Label і Button) користувацького інтерфейсу.

- «tiled», зберігає файл з основою рівня.

У двигуні Godot кожен об’єкт являється Node (нодою), який має певний набір атрибутів, в залежності від типу, кожна нода має свій набір. Проте є певні атрибути які є в кожній ноді, наприклад: Rect – зберігає параметри Position, Size, Min Size, Scale по вісям X, Y, а також Rotation у градусах (рис. 2.4)[11].

Рис. 2.4. Компонент «Rect»

Для відображення двомірних зображень використовується нода Sprite (рис. 2.5). Вона має атрибут Texture в якому вказується міце знаходження зображення, воно обов’язково повинне знаходитися в директорії з проєктом.

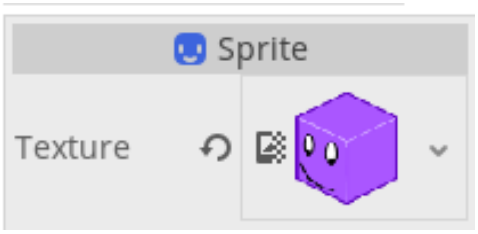

Рис. 2.5. Нода «Sprite»

Для того щоб будь-який об’єкт підкорювався фізиці, існує нода KinematicBody. Найчастиіше це головний герой або персонаж яким потрібно грати. Має базову властивість Transform в якій зміються параметри розташування по осям X, Y.

Анімація перевертання головного персонажа буде виконана за допомогою атрибуту Rotation та функції написаному на GDScript який змінює параметри Rotation.

Для того щоб задати розмір який може взаємодіяти з іншими колізіями необхідно вихначити границі які обмежуються так звану зону дії об’єкта, для

того щоб задати колізію об’єкта використовується CollosionShape. Він має основний параметр Shape, який дозволяє обрати форму колізії, наприклад овал, коло, прямокутник та інші (рис. 2.6).

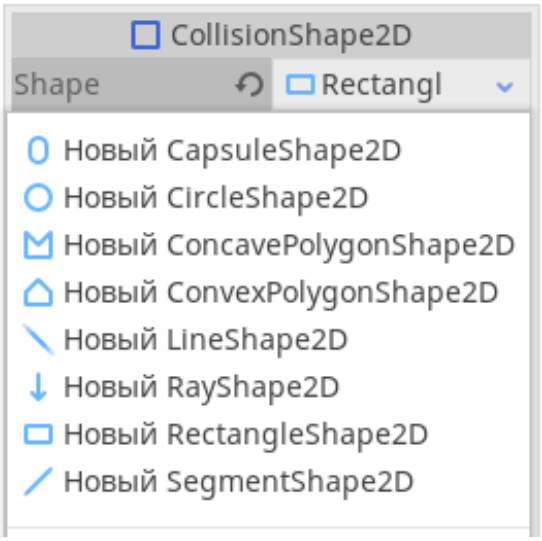

Рис. 2.6. Нода «CollosionShape»

Для створення повноцінних робочих об’єктів використовується декілька нод які утворють ієрархію. Тобто для отримання наприклад ігрового персонажа в ноду KinematicBody поміщаємо ноди Sprite для відображення картинки персонажа та CollosionShape для задання меж колізії (рис. 2.7)[20].

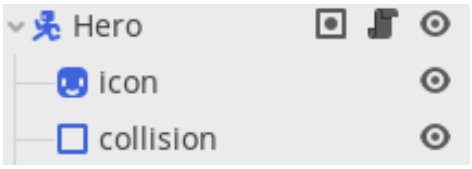

Рис. 2.7. Дерево ноди «Hero»

Об’єкти які повинні взаємодіяти з персонажем мають тип ноди Area. Це нода яка має колізію і логіку, але не підкорюється фізиці. Її основні атрибути це:

- «Monitoring» – дозволяє доторкатися до KinematicBody і інших Area та виходити з них;

- «Monitoriable» – дозволяє взаємодіяти (доторкатися і виходити) з іншими Area (рис. 2.8).

Як і KinematicBody нода повинна мати в ієрархічному дереві CollisionShape (рис. 2.9).

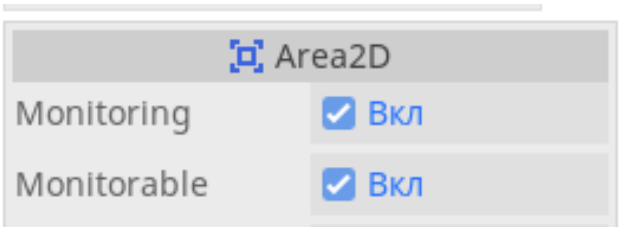

Рис. 2.8. Нода «Area»

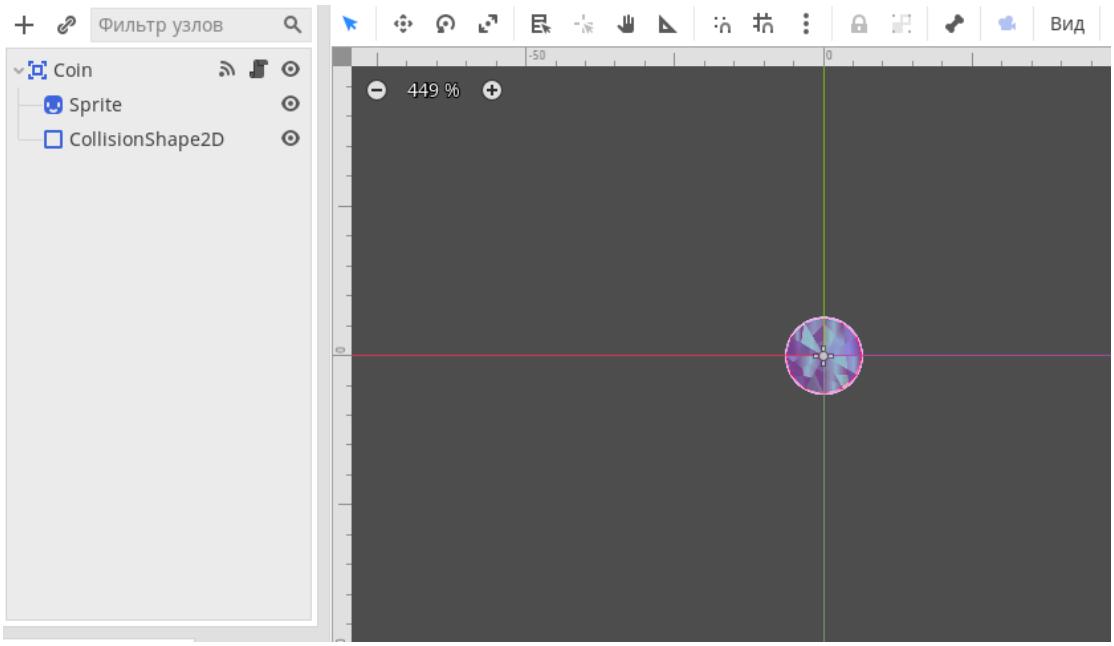

Рис. 2.9. Приклад створення монети з застосуванням ноди «Area»

Далі представлена ралізація ігрових елементів.

Неоновий шип - об’єкт, створенний за допомогою ноди «Area», має компоненти описані в таблиці 2.1, виконує роль перешкоди та виглядає як шип (рис. 2.10). При доторканні буде викликати функцію яка змушуватиме головного персонажа помирати (відображається анімація розпаду на частинки) та перезапускати рівень з початку.

Таблиця 2.1

Компоненти неонового шипа

Нода	Атрибут	Значення	Призначення
Sprite	Texture	Шлях до файлу: «res://gfx/obstacles»	Зображення
CollisionPoligon	Polygon	Triangle	Тип полігону
	Size(X, Y)	-16, 16	Розміри меж полігону

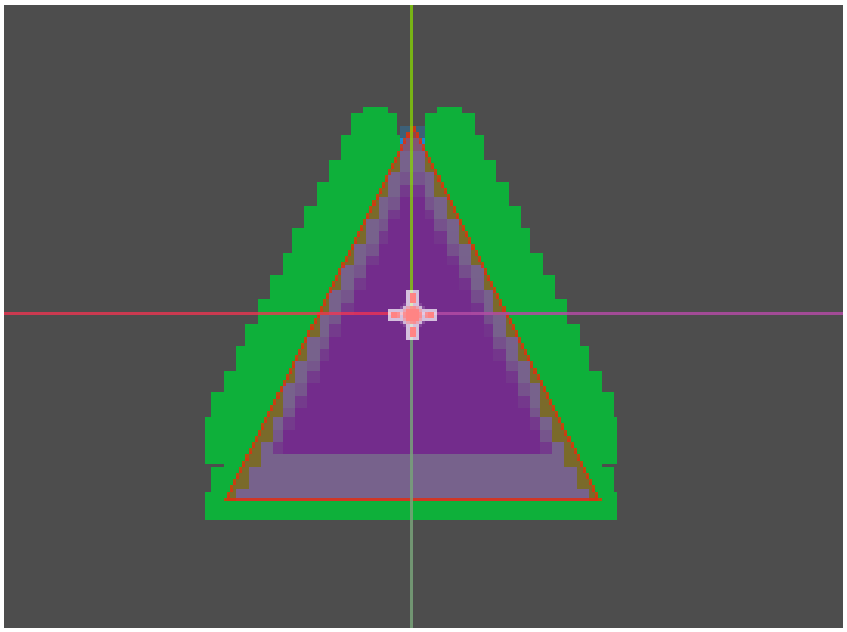

Рис. 2.10. Об’єкт «SpikeNeon»

Тригер – об’єкт, створенний за допомогою ноди «Area». Має компонети описані в таблиці 2.2. Візьмемо тригер фінішу (рис. 2.11), після дотику до нього буде з’являтися меню фінішу з можливістю вибору наступного рівня, повернення до головного меню або повного виходу з гри.

Таблиця 2.2

Компоненти тригера зміни кольору

Нода	Атрибут	Значення	Призначення
CollisionShape	Shape	Rectangle	Межі колізії
Lable	Text	Finish	Текст

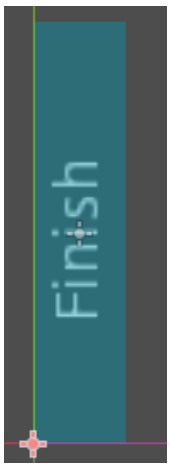

Рис. 2.11. Тригер «Finish»

Всі інші тригери такі як: швидкість, зближення, задній фон мають аналогічні компоненти до тригеру фінішу.

Трамплін та змінювач гравітації схожі на звичайні тригери але дещо ускладнені. Мають компоненти описані в таблиці 2.3. Одні з цих компонентів це: «Sprite» – для того щоб гравцю було видно зображення та «Particles» – анімація частинок, створюється уявлення що із зображення виходять частинки (рис. 2.12). Трамплін змушує персонаж підстрибувати, а змінювач гравітації змінює гравітацію для персонажа, тобто герой буде пересуватися по стелі.

Таблиця 2.3

Компоненти трампліна

Нода	Атрибут	Значення	Призначення
Sprite	Texture	res://gfx/trampoline/trampoline.svg	Зображення
CollisionShape	Shape	Rectangle	Тип полігону
Particles	Amount	10	Кількість
	LifeTime	0.6	Час існування
Нода	Атрибут	Значення	Призначення
Particles	Speed Scale	2	Швидкість переміщення
	Randomes	0.5	Рандомізація появлення
	Texture	res://gfx/bullet.png	Зображення

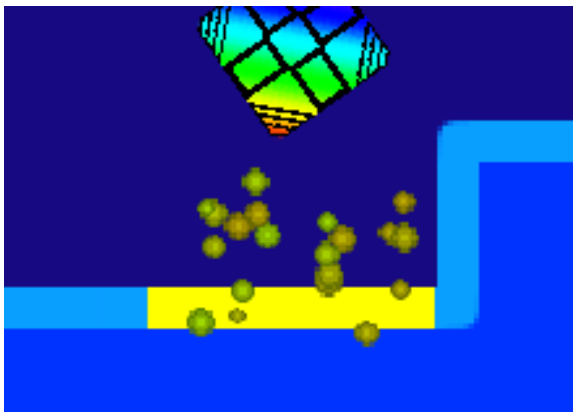

Рис. 2.12. Трамплін

Компоненти змінювача гравітації аналогічні до трампліна

Персонаж – об’єкт, створенний за допомогою ноди KynematicBody (рис. 2.13). Компоненти об’єкта «Hero» описані в таблиці 2.4.

Таблиця 2.4

Компоненти «Hero»

Нода	Атрибут	Значення	Призначення
Sprite	Texture	res://gfx/icons/icon_1.png	Зображення
ColisionShape	Shape	Rectangle	Межі колізії
RayCast	CastTo(X, Y)	8, 0	Направлення руху
Timer	Wait Time	1	Затримка перед перезапуском рівня
Target	-	-	Ціль за якою слідкує камера

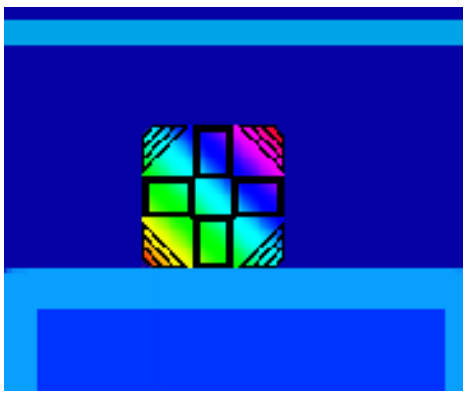

Рис. 2.13. Головний персонаж

Переміщення персонажа (нода KynematicBody) відбувається автоматично. Його керування здійснюється кнопками «Пробіл» або лівою кнопкою миші на ПК, а якщо це мобільний пристрій, то дотик до екрану (двигун самостійно визначає це в залежності від типу експорту). Після натискання герой підстрибує і перевертається, починає діяти функція «Jump» з силою в 300 одиниць. Вона приймає поточне значення персонажа по осі Y і помножує його на значення стрибка.

«motion.y = -jump_strength * host.gravity_vector».

Де «motion.y» - швидкість переміщення по осі Y, «-jump_strength» - сила стрибка, «host.gravity_vector» - поточне направлення персонажу.

Кожний стрибок змушує обертатися ноду «KynematicBody». Для цього потрібно відслідковувати положення за допомогою ноди «RayCast» який створює вектор, щоб повертати дійсне значення у разі зіткнення з колізією.

Камера – нода, яка буде автоматично слідкувати за персонажем та відображати всі його дії. Має наступні основні атрибути: «AnchorMode» - розташування камери (по лівому краю або по центру), «Rotation» - дозвіл на обертання, «Zoom – значення наближення»

Основа рівня – об’єкт, який спроєктовано за допомогою ноди TileMap (рис. 2.14), яка складається з текстури кубу. Має основні компоненти описані в таблиці 2.5.

Таблиця 2.5

Компоненти рінвя

Нода	Атрибут	Значення	Призначення
TileMap	Mode	Square	Тип відображення
	TileSet	res://Objects/Tiles/Neos.tres	Текстура
	Size(X, Y)	32, 32	Розмір

Рис. 2.14. Основа рівня (куб)

Вся платформа в грі складається з великої кількості кубів, які з’єднуються один з одним, утворюючи єдину платформу. Але основа може існувати сама по собі, утворюючи ніби літаючий острів.

Рейтинг кожного гравця спроєктовано за допомогою ноди Label, а саме час проходження рівня, кількість спроб, загальний час проведенний у грі і загальна кількість спроб по кожному рівні. Вся інформація виводиться в меню фініша. Зареєстрований корстувач відображается у правій частині головного вікна (рис. 2.15)

Рис. 2.15. Інфомація по авторизації і рейтингу

Рейтнгова система гри доступна тільки після авторизації користувача в грі. Для нових гравців спочатку необхідно пройти реєстрацію, заповнити відповідні поля в вікні «Реєстрація». Зареєстровані корстувачі через заповення полів відповідного вікна «Registration and Login» (рис. 2.16), усі основні компоненти якого описані в таблиці 2.6.

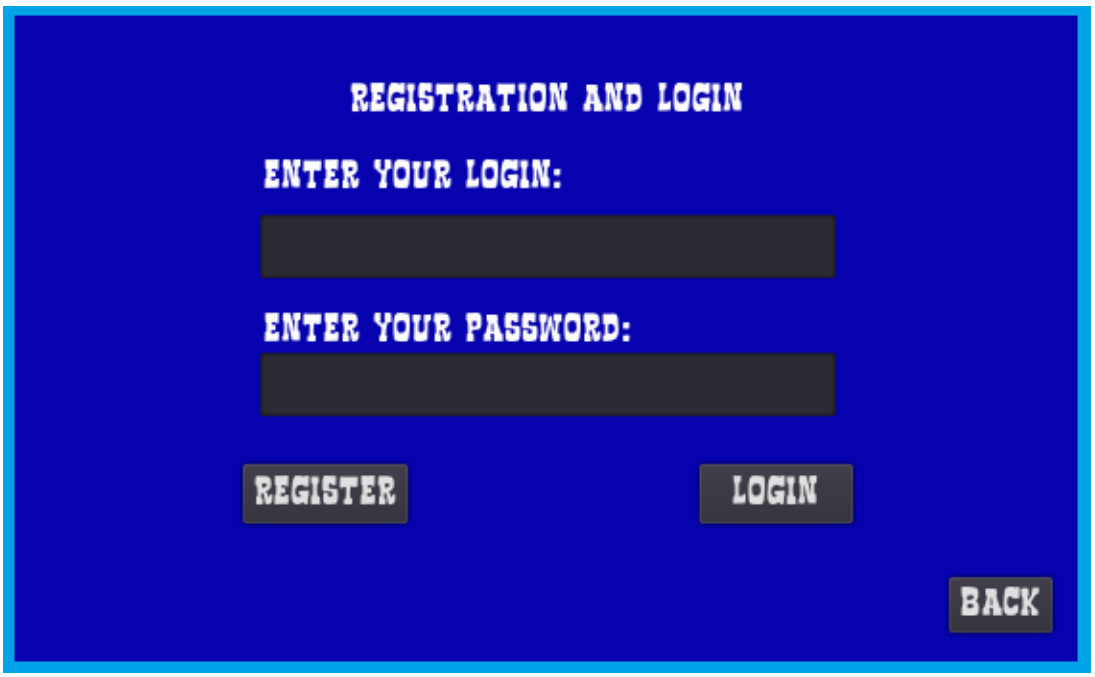

Рис. 2.16. Вікно авторизації/регістрації

Таблиця 2.6

Компоненти вікна «Реєстрації»

Нода	Атрибут	Значення	Призначення
Label	Text	Your game time	Текст поля
	Align	Left	Вирівнювання по горизонталі
	Valign	Center	Вирівнювання по вертикалі
LineEdit	Text	-	Текст
	Align	Left	Вирівнювання по горизонту
	Editable	True	Можливість змінювати текст
	Secret	False	Приховування тексту
	SecretCharecter	*	Символ приховування
Button	Text	Back	Текст
	Icon	-	Значок
	Align	Center	Вирівнювання по горизонту
	Disbled	False	Вимкнення за замовчуванням
	Pressed	False	Натиснута за замовчуванням

Рейтинг гравця відображаєтсья кожного разу при завершенні рівня в вікні «Фінішу» та представляє собою поле у вигляді одного компоненту «Label» (з наступними властивостями: «Text» який зберігає текст, «Aling» визначає ирівнювання по горизонталі, «Valign» визначає вирівнювання по вертикалі) та оновленою інформацією по результатам гри. За замовчуванням встановлено нульовий рейтинг кожному новому гравцю в системі.

В грі передбачено автозбереження. Після зміни будь-яких даних (збір монети, нараховування спроб і часу та інше) вони будуть зберігатися автоматично в файл. Всі дані (кількість монет, поточний персонаж, поточний задній фон, налаштування гучності та доступні або не доступні компоненти) автоматично завантажуються після запуску гри і авторизації гравця[22].

Ліситинг програмного коду описаний в додатку А.

2.5. Обґрунтування та організація вхідних та вихідних даних програми

Організація вхідних та вихідних даних в грі є вирішальним кроком у забезпеченні її функціональності та взаємодії з гравцями. Вона дозволяє створити зручний та ефективний інтерфейс, що допомагає гравцям легко взаємодіяти з грою та отримувати потрібну інформацію.

У 2D грі, розробленій за допомогою Godot Engine та GDScript, надходження вхідних даних реалізовано через інтерактивні елементи управління. Гравець може використовувати клавіатуру та мишу для керування персонажем, взаємодії з об'єктами та навігації в меню. Це дозволяє гравцю повністю зануритися в ігровий процес і точно контролювати дії свого персонажа.

Елементи управління підтримують валідацію та перевірку правильності введених даних, що допомагає уникнути помилок та забезпечити коректну реакцію гри на дії гравця. Наприклад, гра обробляє натискання клавіш для переміщення персонажа, атак, взаємодії з предметами та виконання інших дій.

Вихідні дані гри представлені у вигляді графіки, звуків та тексту, що відображають стан гри та надають гравцю необхідну інформацію. Результати дій гравця виводяться у зрозумілому форматі - анімації руху персонажів, звукові ефекти взаємодії з об'єктами та текстові повідомлення про події. Гравець може бачити рівень здоров'я персонажа, кількість зібраних предметів, поточні завдання та іншу важливу інформацію[10].

Ці елементи забезпечують зручну та ефективну взаємодію гравця з грою, що сприяє поліпшенню ігрового досвіду та задоволенню від гри.

2.6. Опис розробленого програмного продукту

2.6.1. Використані технічні засоби

Для розгортання гри необхідно мати обчислювальну машину з мінімальними характеристиками, щоб забезпечити стабільну та ефективну роботу.

Для розгортання використовувалася наступна обчислювальна машина з наступними характеристиками:

- процесор: AMD Ryzen 5 3600 3.20GHz;
- відеокарта GeForce GTX 1660 TI 3600GHz
- оперативна пам'ять: 16,0 ГБ;
- жорсткий диск: 220 ГБ вільного місця;
- неперервний доступ до мережі інтернет;
- операційна система: Windows 10;

2.6.2. Виклик та завантаження програми

Для використання цієї гри необхідно завантижити її через офіційну платформу на компьтер (використати платформу Steam) або телефон (використати платформу Google Play), встановити та зареєструватися (1 раз). Після чого користувачеві автоматично буде надано повний доступ до всього інтерфейсу.

2.6.3. Опис інтерфейсу користувача

Було проведено певну кількість тестів для перевірки працездатності та виявлення всіх помилок і недоліків. Кількість кадрів в секундку повністтю стабільна, зависань та багів не має.

Всі вікна гри та її функціонал коректно зберігається не залежно від ОС в якій було запушено.

Розглянемо тестування на ПК.

Після входу до гри необхідно зареєструватися або авторизуватися якщо дані користувача є в системі. Після натискання кнопки «Авторизації» з’являється меню реєстрації/авторизації, в якому знаходиться поля для вводу прізвища та імені (рис. 2.17). Після вводу даних необхідно натиснути на кнопку реєстрації або авторизації (в залежності від статусу даних гравця, якщо гравець зареєстрований, він натискає кнопку «Входу», якщо гравець не зареєстрований, він повиненнатиснути кнопку «Реєстрації»).

Рис. 2.17. Процес авторизації/реєстрації

Для того щоб розпочати гру необхідно скористатись відповідною кнопкою «Play», яка переносить гравця в меню рівнів (рис. 2.18). За логікою гри рівні стають доступні поступово, тобто не можна перейти на більш складний рівень не пройшовши попередній. Вибір рівня представлено за допомогою набору кнопок з відповідними їх номерами: «Level 1», «Level 2», «Level 3».

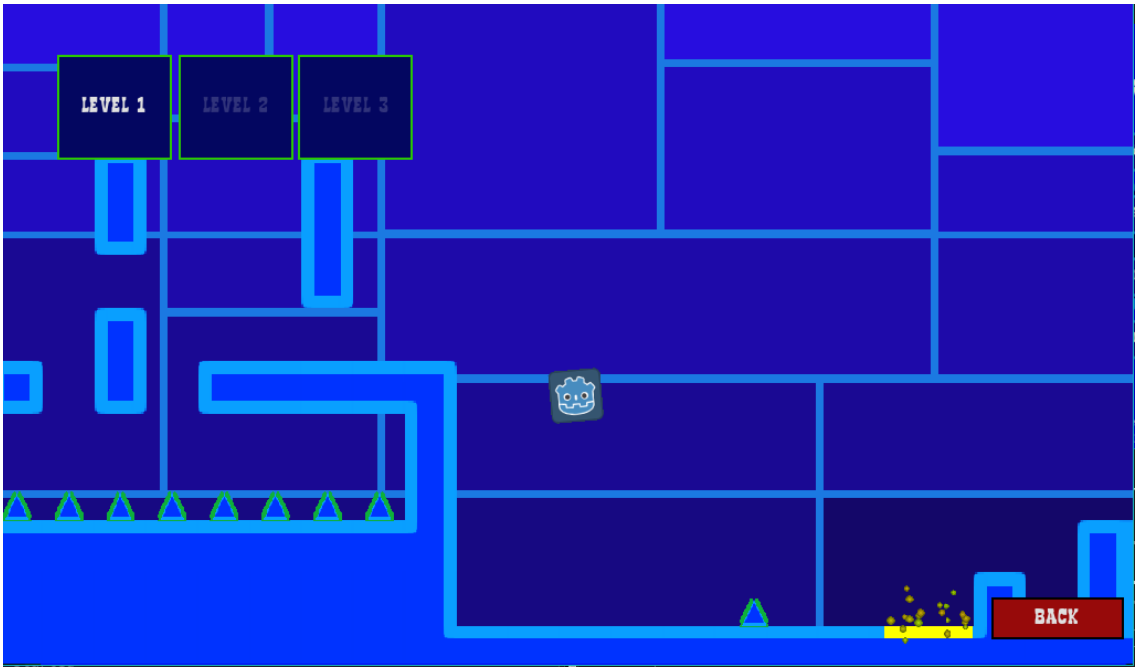

Рис. 2.18. Меню рівнів

Після вибору рівня починається гра (рис. 2.19). Персонаж самостійно, без натискання кнопок або виконання інших дій, з певною швидкістю пересувається вперед, при цьому камера постійно слідкує за ним. На початку рівня відображається кількість спроб - «Attempt». Лічильник рахує з одиниці тому що при запуску рівня вже почалась перша спроба, а надпис інформує гравця котра спроба в нього на рахунку.

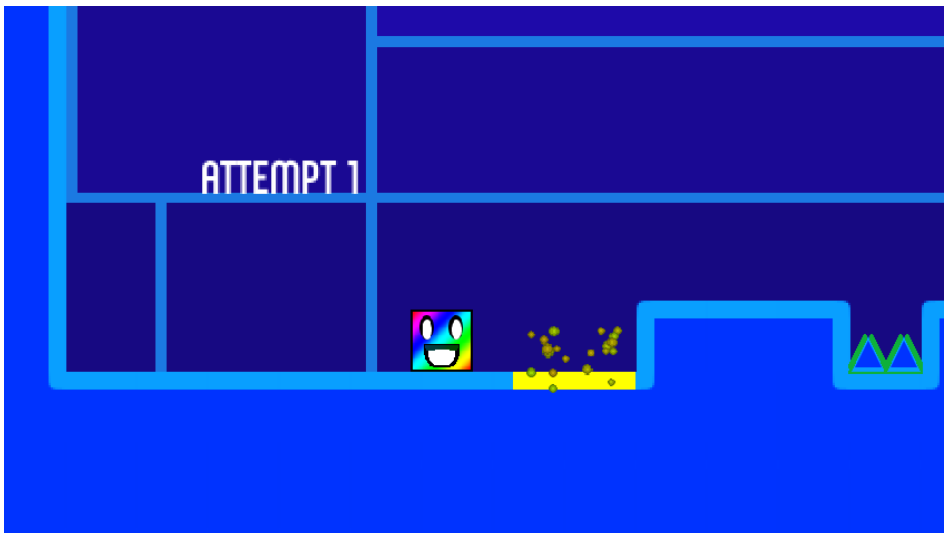

Рис. 2.19. Початок рівня

Просуваючись по рівню персонаж зустрічає різні перешкоди при дотику до яких відбувається певна реакція. Наприклад, після дотику до шипів він зникає і відображається анімація зникнення (рис. 2.20). Гравцю нараховується одна спроба і персонаж повертається на початок рівня (рис. 2.21). Така логіка працює з кожним шипом який присутній в рівні.

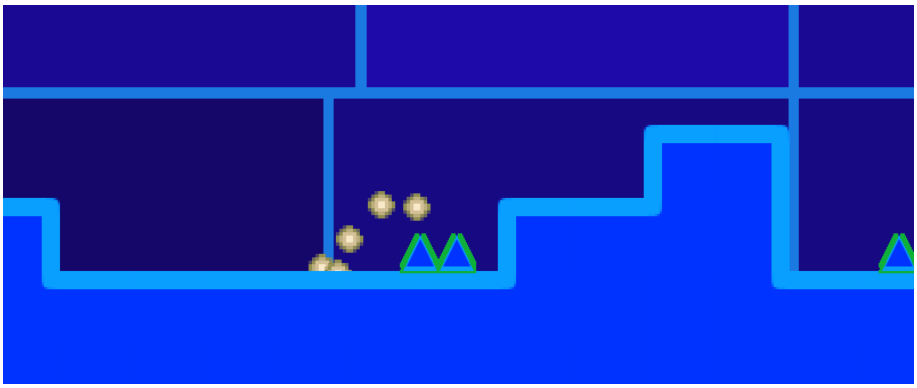

Рис. 2.20. Зникнення персонажа

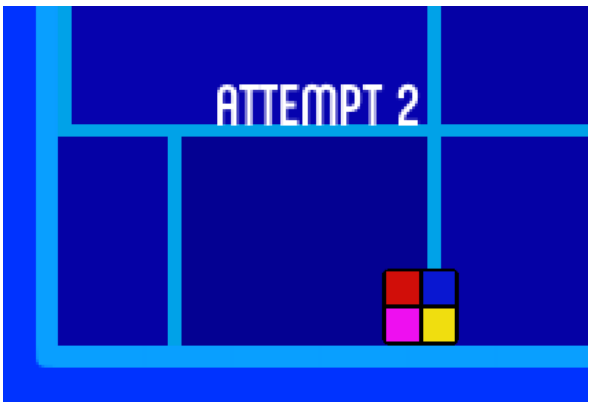

Рис. 2.21. Додання спроби

При проходженні через тригери відбуваються певні зміни в грі:

- тригер швидкості, змінює швидкість персонажу;

- тригер кольору, змінює колір заднього фону, платформи, самого персонажа, шипів і трамплінів;

- тригер гравітації, змінює гравітацію персонажа.

На рисунку 2.22 зображено коректну одночасну роботу одразу двох тригерів: зміна кольору (тригер кольору не видимий для гравця) та гравітації (представлений у вигляді кольорової платформи з якої вилітають частинки). Після повторного проходження через аналогічні тригери, які зустрічаються при подальшому проходженні рівня, процес гри повернеться до норми (стан який був з початку гри).

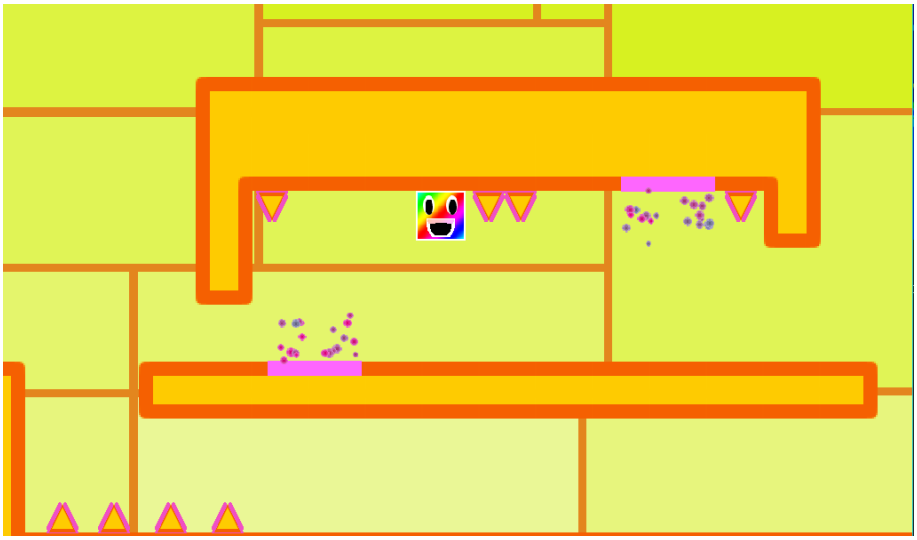

Рис. 2.22. Зміна кольору та гравітації

Для зручності користувача в грі передбачено паузу. Вона спрацьовує при натисканні кнопки «ESC» на фізичній клавіатурі або подвійне натискання по екрану мобільного пристрою (рис. 2.23). Процес гри повністю зупиняється, стає неможливим натискання на будь-які місця вікна окрім кнопок в меню паузи.

Меню паузи складається з таких кнопок:

- кнопка «Resume» (продовжити), дозволяє продовжити гру (меню паузи зникає і гра відновлюється з місця де була зупинена);

- кнопка «Restart» (перезавантажити), дозволяє почати рівень з початку, при чому гравцю додається одна спроба;

- кнопка «Main Menu» (головне меню), дозволяє повернутися до головного меню;

- кнопка «Quite the game» (вийти з гри), дозволяє вийти з гри до робочого столу.

Рис. 2.23. Меню паузи

Кінець рівня представлений полосою чорно-білих квадратів і тригером фінішу який встановлений на полосі. Після того як персонаж перетнув фінішну полосу з’являється вікно фініша (рис. 2.24). В цьому вікні розташована повна інформація про рейтинг гравця: кількість спроб за один

сеанс гри, поточний час в секундах витрачений на проходження рівня, загальна кількість спроб відповідним і загальний час витрачений на процес гри.

Рис. 2.24. Меню фініша

Окрім цього в вікні фініша є декілька додаткових кнопок, а саме:

- «Restart Level» (перезавантажити рівенб) - дозволяє розпочати відкритий рівень з початку;

- «Main Menu» (головне меню) - повернення до головного меню;

- «Next Level» (наступний рівень) - перехід на новий рівень;

- «Qite the Game» (вихід з гри) – завершення.

Після того як користувач обрав і натиснув кнопку меню фініша зникає і відбувається дія відповідно до натиснутої кнопки.

Якщо гравець завершує сеанс гри всі його досягнення автоматично зберігаються у відповідних файлах. При повторній авторизації в грі вся

збережена інформація про досягнення гравця оновлюється і фіксується у відповідних рівнях (рис. 2.25).

Рис. 2.25. Відкриття нового рівня

На наступних рівнях функціонал повністю аналогічний, весь інтерфейс і вікна меню такі самі, як і в першому рівні.

Головне меню містить дві додаткові опції: магазин та налаштування. Протягом гри користувач не тільки проходить рівень на час постійно покращуючи свою майстерність скорочуючи цей показник, але й збирає монети. Вони мають різну текстуру та різний номінал. За них можна робити абгрейд ігрового персонажу або теми заднього фону гри.

Для того, щоб зайти в магазин необхідно скористатися кнопкою «Shop» в головному меню гри. Вікно магазину має наступне меню (рис. 2.26):

- загальна кількість монет, розташована в верхньому правому куті;

- вибір персонажу, в лівій частині меню розташовані персонажі, яких можна обрати після того як на загальному рахунку буде накопичена певна кількість монет;

- вибір теми заднього фону, в правій частині меню розташована тема гри, яку можна обрати після того як на рахунку буде накопичена певна кількість моне;

- кнопка «Save and Back» (зберегти та повернутися), після натискання зберігає змінені значки головного героя та теми фону і повертає користувача до головного меню гри.

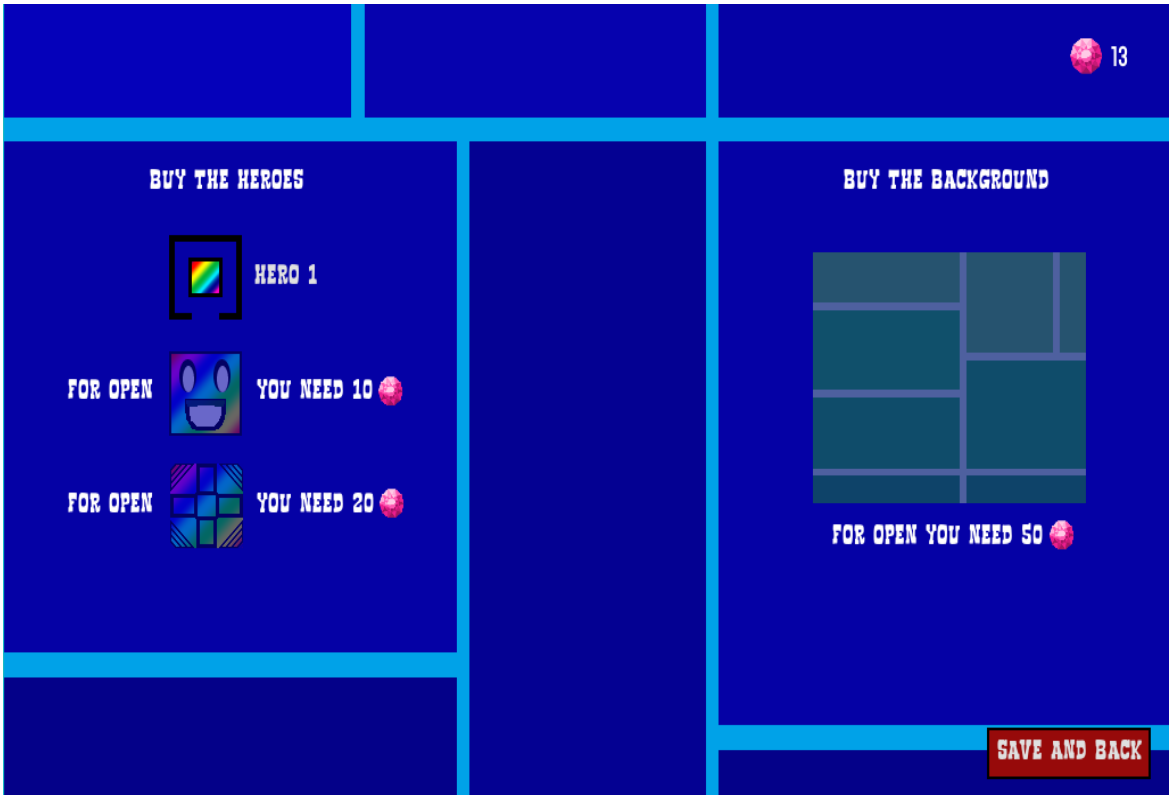

Рис. 2.26. Меню магазину

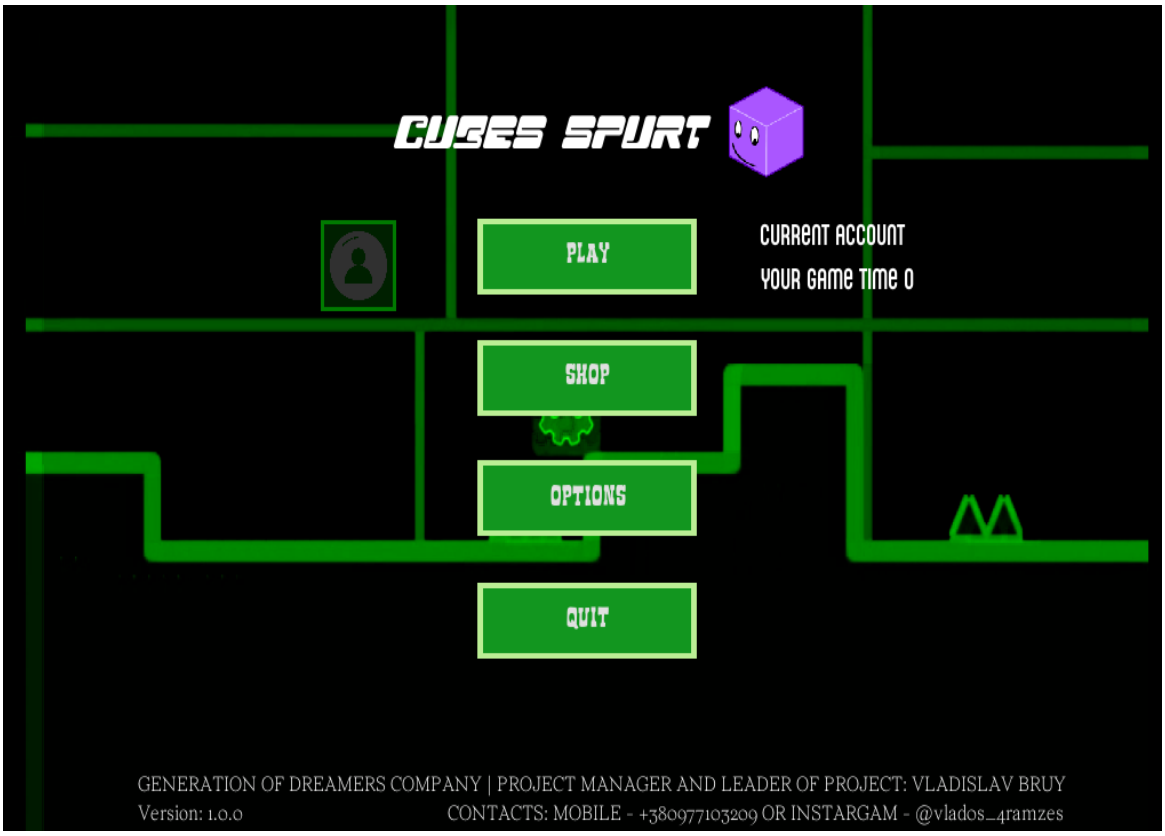

Рис. 2.27. Змінена тема фону

Для того щоб зайти до вікна налаштувань необхідно натиснути на кнопку «Options» в головному меню гри (рис. 2.28). Вікно містить в собі меню з наступними функціями налаштування:

- два повзунка, один для зміни гучності в меню гри, другий для зміни гучності безпосередньо в рівні;

- дві кнопки, перша вмикає або вимикає музика в меню гри, а друга має аналогічний функціонал але при проходженні рівня;

- кнопка «Save and Back» (зберегти та повернутися), після натискання зберігає всі налаштування, статус видимості повзунків, а після повертає користувача до головного меню.

Також вимикаючи або вмикаючи кнопки зникають або з’являються повзунки гучності (рис. 2.29).

Рис. 2.28. Меню опцій

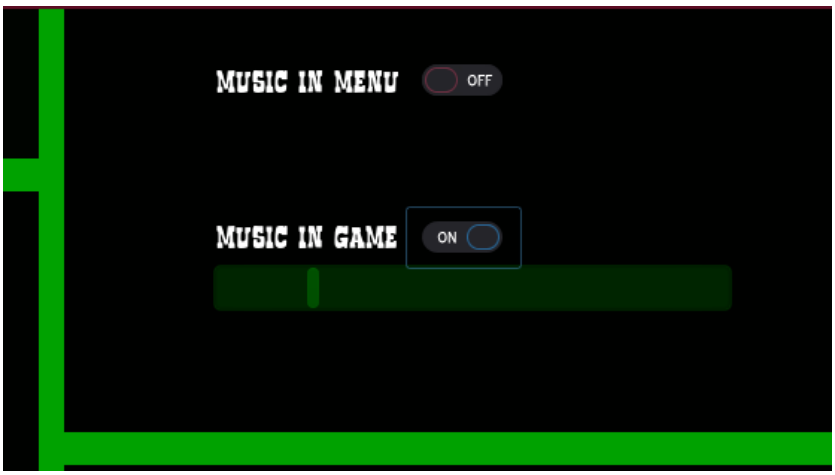

Рис. 2.29. Змінені налаштування

Для зручності користувача в процесі ознайомлення з грою було створено інструкцію користування, в який описано функціонал управління головним героєм та користувацьким інтерфейсом (рис. 2.30). Створювалася в спеціальному додатку для розробки інструкцій «Help+Manual». Ця програма дозволяє створювати електронні багаторівневі інструкції в будь-яких сферах діяльності з можливістю конвертації різні формати (наприклад .pdf, .chm, .docx та інші). Інструкція знаходиться в директорії з експортованими файлами гри.

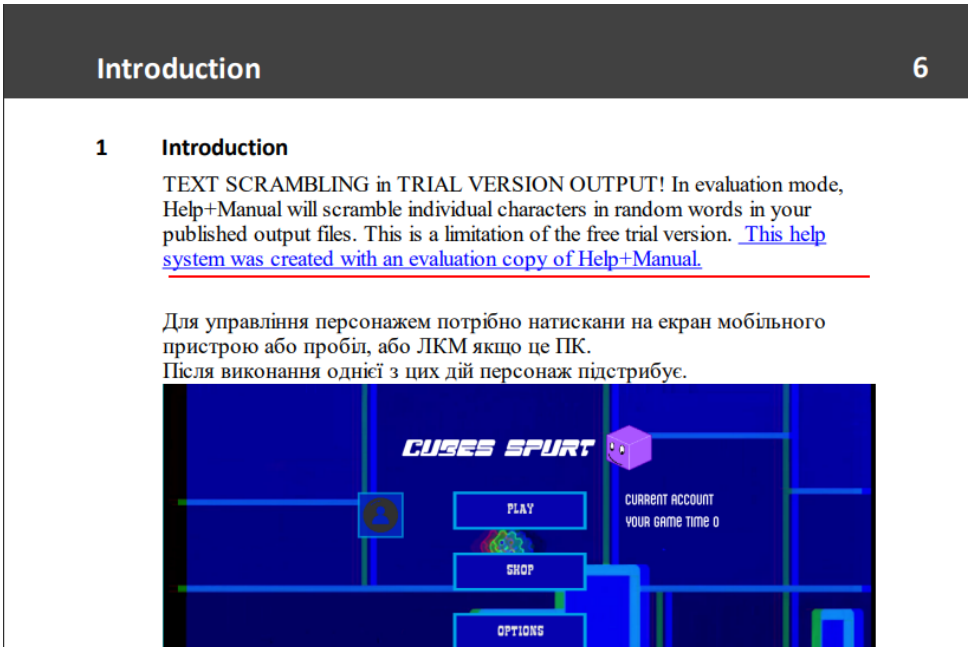

Рис. 2.30. Сторінка з інструкції користувача

РОЗДІЛ 3
ЕКОНОМІЧНИЙ РОЗДІЛ

3.1 Розрахунок трудомісткості розробки програмного забезпечення

Для розрахунків необхідні дані наведені нижче:

- споживча потужність комп'ютера P = 0,4 кВт;
- первісна вартість комп'ютера $В_п$ = 32000 грн;
- ліквідаційна вартість комп'ютера $В_л$ = 14000 грн;
- термін корисного використання комп'ютера $Т_{вик}$ = 5 років;
- розмір погодинної заробітної плати програміста $ЗП_{пог}$ = 312,5 грн/год;
- ціна 1 кВт години електроенергії Ц = 1,68 грн/кВт · год;
- ціна 1 $м^3$ води $Ц_в$ = 31,36 грн/$м^3$;
- витрати води за годину $В_в$ = 0,02 $м^3$;
- кількість годин, витрачених на розробку програми $Т_{прог}$ = 120 год;
- місячний фонд робочого часу МФ = 176 год.

У процесі експлуатації основні фонди піддаються втраті основними фондами своєї вартості. Процес відшкодовування зношування основних фондів здійснюється шляхом амортизації.

При цьому вартість, яка амортизується, - це первісна або переоцінена вартість необоротних активів, зменшена на ліквідаційну вартість. У свою чергу, ліквідаційна вартість є сумою коштів або вартістю інших активів, яку підприємство очікує отримати від реалізації (ліквідації) необоротних активів після закінчення терміну їх корисного використання, за вирахуванням витрат, пов'язаних з продажем (ліквідацією).

Визначення терміну корисного використання. Це очікуваний період часу, протягом якого необоротні активи будуть використовуватися підприємством, обсяг продукції (робіт, послуг).

Ліквідаційну вартість і термін корисного використання підприємство встановлює самостійно та фіксує в наказі по підприємству при визнанні об'єкта

основні затрати активом (при зарахуванні на баланс).

При визначенні терміну корисного використання (експлуатації) основних затрат слід враховувати:

- очікуване використання об’єкта підприємством з урахуванням його потужності або продуктивності;

- фізичний і моральний знос, що передбачається;

- правові або інші обмеження щодо термінів використання об’єкта та інші фактори.

Об’єктом амортизації є вартість основних засобів, окрім вартості земельних ділянок і природних ресурсів.

Розрахуємо норму амортизації за формулою (3.1)

$$Н_а = \frac{В_п - В_л}{В_п \cdot Т_{вик}} \cdot 100\%, \quad (3.1)$$

$$Н_а = ((32 - 14) \div (32 \cdot 5)) \cdot 100\%$$

Річна норма амортизації складає 11,25%.

Для здійснення економічно вигідного процесу нарахування амортизації важливо правильно вибрати метод. Амортизація основних засобів проводиться із застосуванням таких методів:

- прямолінійного;

- зменшення залишкової вартості;

- прискореного зменшення залишкової вартості;

- кумулятивного;

- виробничого.

За прямолінійним методом річна сума амортизації визначається діленням вартості яка амортизується, на термін корисного використання об’єкта основних засобів.

При застосуванні прямолінійного методу річна сума амортизації розраховується за формулою:

$$А = \frac{В_п - Н_а}{100}, \quad (3.2)$$

$$А = (32000 – 11,25) ÷ 100 = 3600 \text{ грн}$$

Відповідно до статті 145 Податкового кодексу Україні, мінімально допустимий термін корисного використання ПК становить 2 роки.

Потім визначається сума амортизації за 1 місяць і за час, витрачений на створення ПП.

Амортизаційні відрахування за 1 місяць визначаються шляхом ділення суми амортизації (А) на 12 місяців:

$$А_{міс} = \frac{А}{12}, \quad (3.3)$$

$$А_{міс} = А ÷ 12 = 300 \text{ грн}$$

Залежно від стану основних фондів вони оцінюються за повною (первісною) та залишковою вартістю, яка ще називається ліквідаційною вартістю.

Ліквідаційна вартість - це залишкова вартість основних фондів на час їх вибуття з експлуатації, спричиненого зношенням. Визначаємо ліквідаційну вартість комп'ютера за місяць за формулою, для цього використовуємо амортизаційний період t, який дорівнює 0,083.

Розраховуємо ліквідаційну вартість комп'ютера за місяць за формулою:

$$В'_п = В_п – t \cdot А_{міс.}, \quad (3.4)$$

$$В'_п = 32000 – 0,083 \cdot 300 = 31975 \text{ грн}$$

Суму амортизації за час, витрачений на створення програми визначаємо за формулою, де $Т_{прог}$ - кількість витрачених часів на розробку програми.

$$З_{аморт} = \frac{А_{міс} \cdot Т_{прог}}{МФ}, \quad (3.5)$$

$$З_{арморт} = 300 \cdot 120 \div 176 = 204,5 \text{ грн}$$

Структура бюджету.

Пряма заробітна плата - це плата за час, витрачений працівником на безпосереднє виготовлення продукції, надання послуги; непряма заробітна плата – це плата за час, витрачений на виконання операцій, які є необхідною та невід'ємною частиною виробничого процесу (не будучи операціями з безпосереднього виробництва продукції). Непряма зарплата може бути включена до виробничих накладних витрат та розподілена пропорційно до кількості випущеної продукції, включена до собівартості реалізованої продукції або вартості запасів. У бюджеті прямих витрат на оплату праці враховуються: витрати на оплату праці тільки виробничих робітників. Зазначимо, що ці витрати вважаються змінними, оскільки змінюються пропорційно до зміни обсягу виробництва; зарплата інженерів, начальників дільниць, ремонтного (цехового) персоналу, якщо вони працюють позмінно. При цьому зарплата чергового персоналу, що обслуговує всі зміни, незалежно від обсягів виробництва, відноситься до умовно-постійних витрат і враховується в бюджеті загальновиробничих накладних витрат. Крім прямих витрат на зарплату можна виділити загальні витрати на оплату праці, куди входить ЄСВ, який нараховується на зарплату, різні виплати та компенсації. Сюди ж можна віднести додаткові затрати, наприклад на додаткове страхування персоналу.

Інші заохочувальні та компенсаційні виплати. До них відносяться виплати у формі винагород за результатами роботи за рік, премії за спеціальними системами та положеннями, компенсаційні та інші грошові й матеріальні виплати, які передбачені актами діючого законодавства або ті, що проводяться понад встановлені затверджені акти норм.Оплата праці (заробітна плата) - це грошовий вираз вартості і ціни робочої сили, який виступає у формі будь-якого заробітку, виплаченого власником підприємства працівникові за виконану роботу.

Основна заробітна плата працівника визначається тарифними ставками, посадовими окладами, відрядним розцінками, а також доплатами у розмірах, встановлених чинним законодавством. Її розмір залежить від результатів роботи самого працівника.

Розмір основної заробітної плати визначають за формулою:

$$ЗП_{осн} = ЗП_{пог} \cdot Т_{прог}, \quad (3.6)$$

$$ЗП_{осн} = 312,5 \cdot 120 = 34375 \text{ грн}$$

Величина додаткової заробітної плати визначається кінцевим результатами діяльності підприємства і виступає у формі премій, винагород, заохочувальних виплат, а також доплат у розмірах, що перевищують встановлені чинним законодавством. Розмір додаткової заробітної плати визначають за формулою, де Пд - відсоток доплат за працю понад установлені норми:

$$ЗП_{дод} = ЗП_{осн} \cdot П_{д}, \quad (3.7)$$

$$ЗП_{дод} = 34375 \cdot 12\% = 4125 \text{ грн}$$

На підприємствах найчастіше використовують дві форми оплати праці: погодинну і відрядну.

Погодинна форма передбачає оплату праці в залежності від відпрацьованого часу і рівня кваліфікації. Вона застосовується тоді, коли недоцільно нормувати роботи або вони взагалі не піддаються нормуванню.

Відрядна форма передбачає залежність суми заробітку від кількості виготовлених виробів або обсягу виконаних робіт за певний проміжок часу. Основними умовами застосування відрядної форми оплати праці є наявність кількісних показників роботи, що безпосередньо залежать від конкретного працівника і піддаються точному обліку, а також необхідність стимулювання зростання обсягу.

Підприємства усіх форм власності зобов`язані здійснювати відрахування на державне соціальне страхування в певному проценті від фонду оплати праці. До цієї статті відносяться затрати, встановлені законодавством України:

- податок з доходів фізичних осіб (ПДФО) - 18%;

- військовий збір (ВЗ) - 1,5%;

- всього страхування - 19,5%.

Ці відрахування здійснюються щомісячно по всім видам зарплати і премій. Сума відрахувань визначається шляхом помноження встановленого тарифу на суму нарахованої зарплати.

Відрахування на соціальні заходи складають 22% від фондів основної і додаткової заробітної плати і включають відрахування на обов'язкове соціальне страхування, пенсійне страхування, фонд безробіття та інше.

Витрати на зарплату ($З_{з.пл.}$) визначаються за формулою, де $П_{єд}$ - розмір єдиного внеску:

$$З_{з.пл.} = ЗП_{осн} + ЗП_{дод} \cdot \frac{100 \cdot ЗП_{єд}}{100}, \quad (3.8)$$

$$З_{з.пл.} = 34375 + 4125 \cdot 100 \cdot 22\% \div 100 = 46970 \text{ грн}$$

Витрати на технологічну електроенергію визначаються за такою формулою, де Ц - тариф 1 кВт години електроенергії, Р - споживча потужність комп'ютера в кВт, $Т_{прог}$ - кількість годин, витрачених на розробку програми.

Розрахуємо за преставленою формулою:

$$З_{ен} = Т_{прог} \cdot Ц \cdot Р, \quad (3.9)$$

$$З_{ен} = 120 \cdot 1{,}68 \cdot 0{,}4 = 80{,}64 \text{ грн}$$

Для розрахунку витрат на воду, використовуємо вхідні дані, а саме: тариф за 1м^3 води ($Ц_в$), який становить 31,36 грн/м^3 і об'єм витрат води за годину ($В_в$).

Витрати на воду ($З_в$.) визначаються за такою формулою, де $Ц_в$ - ціна 1 м^3

води, $В_в$ - витрати води за годину:

$$З_в = Т_{прог} \cdot Ц_в \cdot В_в, \quad (3.10)$$

$$З_в = 120 \cdot 31,36 \cdot 0,02 = 75,26 \text{ грн}$$

Загальні витрати на енергію та воду розраховуються за формулою:

$$З = З_{ен} + З_в, \quad (3.11)$$

$$З = 80,64 + 75,26 = 156 \text{ грн}$$

3.2 Розрахунок витрат на створення програми

Для досягнення своєї основної мети - максимізації прибутку підприємство повинно понести певну суму витрат. Собівартість є узагальнюючим показником економічної ефективності підприємства і може розраховуватись на одиницю продукції, робіт або послуг. Собівартість продукції розраховується за економічними елементами витрат та статтями калькуляції. Єдина і обов`язкова для всіх підприємств номенклатура економічних елементів витрат на виробництво:

- сировина й основні матеріали;
- допоміжні матеріали;
- паливо, енергія;
- заробітна плата. Основна і додаткова;
- відрахування на соціальне страхування;
- амортизація основних фондів;
- інші витрати.

Згідно з прийнятим порядком обліку може враховуватися результат праці кожного виконавця окремо або колективний груповий результат по всій групі працюючих. Далі потрібно визначитися самим процесом розробки, який найчастіше визначається розробленими всередині компанії стандартами[8].

Прямі витрати безпосередньо відносяться до конкретного об'єкта

калькуляції видів виробів або груп однорідних виробів, робіт, послуг - це витрати сировини та матеріалів, заробітної плати виробничих робітників, зайнятих виготовленням продукції, разом з відрахуваннями на соціальні потреби та інші витрати, які можна віднести на собівартість продукції виходячи з первинних документів. По завершенні своєї роботи менеджер по роботі з клієнтом подає звіт про послуги, продаж або новий замовлення начальнику відділу маркетингу. Така послідовність пропонується два етапи перших яких безпосередньо пов'язаний з розрахунком виробничої собівартості туристичної продукції та розподілом витрат на реалізовану продукцію на другий етап регламентними діями, які доцільно виконувати тільки після формування виробничої собівартості та собівартості реалізованої продукції[12]. Розрахуємо загальну суму витрат на програму:

$$З_{заг} = З_{аморт} + З_{з.пл} + З, \quad (3.12)$$

$$З_{заг} = 204,5 + 46970 + 156 = 47330,5 \text{ грн}$$

Ціна - це грошовий вираз вартості товару, кількість грошей, що сплачується або одержується за одиницю товару або послуги. Ціна на будь-який товар складається з собівартості, прибутку, податку на додану вартість, також до складу ціни може входити акцизний збір, націнки постачальницько-збутових організацій, торговельні надбавки або знижки. При розрахунку ціни виробника ($Ц_{вир.}$) необхідно врахувати очікуваний прибуток (П) за формулою:

$$П = \frac{С \cdot \%рентабельність}{100}, \quad (3.13)$$

$$П = 47330,5 \cdot 4\% \div 100 = 1893,22 \text{ грн}$$

Розрахуємо ціну виробника за формулою:

$$Ц_{вир} = С + П, \quad (3.14)$$

$$Ц_{вир} = 47330,5 + 1893,22 = 49223,72 \text{ грн}$$

Залежно від особливостей процесу купівлі-продажу існують оптові і роздрібні ціни. Оптові (відпускні) ціни - це ціни, за якими державні, колективні та приватні підприємства розраховуються між собою або за великі партії товарів.

Роздрібні ціни - ціни, за якими здійснюється продаж товарів населенню торговельними підприємствами або закладами громадського харчування. Такі ціни встановлюються підприємствами самостійно, виходячи із якості товару, кон'юнктури ринку, ціни закупівлі.

В основі формування оптових та роздрібних цін лежить собівартість продукції, яка є нижньою межею ціни. При формуванні оптової ціни підприємства до собівартості продукції, представленої у формі калькуляції, додаються: величина прибутку, податок на додану вартість, акцизний збір.

Податок на додану вартість (ПДВ) є видом універсального акцизу, який встановлюється за єдиною ставкою до всього обороту. Це основний вид непрямого оподаткування, який забезпечує основну масу податкових надходжень до бюджету. ПДВ встановлюється у вигляді процентної надбавки до цін[13].

Розрахуємо податок на додану вартість за формулою, де $N_{пдв}$ = ставка податку на додану вартість:

$$ПДВ = \frac{Ц_{вир} \cdot N_{пдв}}{100}, \quad (3.15)$$

$$ПДВ = 49223{,}72 \cdot 20\% \div 100 = 9884{,}7 \text{ грн}$$

Оптова ціна ($Ц_{опт}$) розраховується наступним чином:

$$Ц_{опт} = Ц_{вир} + ПДВ, \quad (3.16)$$

$$Ц_{опт} = 49223{,}72 + 9884{,}74 = 59068{,}5$$

Реалізація програмного продукту здійснюється послідовними кроками. Спочатку розробник проводить аналіз вимог, тобто визначаються потреби та

вимоги до мобільного додатку. Далі виконується процес визначення архітектури, компонентів (кнопок, текстових полів та ін.) розташованих на інтерфейсах, потім розробник програмує мобільний додаток, а саме:

- проєктує систему;

- обирає мову програмування для написання додатку, а саме GD Script;

- виконує процес написання коду з урахування виняткових ситуацій.

Програмне забезпечення супроводжується двома документаціями, а саме:

- документація для програміста з високорівневої мови програмування GD Script, дана документація детально описує логіку, структуру програми за допомогою різноманітних схем, коментарів до коду та пояснює виняткові ситуації;

- документація для користувача, дана документація повністю детально описує кожен інтерфейс з його елементами візуалізації, тобто наприклад елемент візуалізації кнопка, в документації описується, що виконує даний елемент та для чого він призначений[8].

Для використання додатку його потрібно встановити на мобільний пристрій або комп’ютер.

Програма продається замовнику за оптовою ціною або є можливість продати копію за 350 грн.

Ефективність виробництва - це узагальнене і повне відображення кінцевих результатів використання засобів, предметів праці і робочої сили на підприємстві за певний проміжок часу. Загальну економічну ефективність виробництва ще називають загальною продуктивністю виробничої системи.

Для визначення економічної ефективності втілення програмного продукту застосовується два аспекти:

- економічна оцінка відповідних результатів;

- економія витрат (собівартості) вирішення визначених задач за рахунок скорочення часу на виконання відповідних робіт, застосовуючи програмний продукт (програмне забезпечення).

Економічна оцінка відповідних результатів здійснюється бальним

методом. У зв'язку із застосуванням нового програмного продукту соціальний результат нового інноваційного продукту виявляється в тому, що під час його використанні покращуються:

- дозволяюча можливість;
- точність;
- діапазон сприйняття;
- надійність;
- скорочується час пошуку;
- знижується похибка;
- збільшується точність отримання сигналу, його швидкодійність;
- функціональність ПЗ;
- надійність функціонування ПЗ;
- зручність використання ПЗ;
- раціональність ПЗ;
- супроводжуванність ПЗ.

Річний економічний ефект розраховується за формулою, де $З_р$ – затрати при використанні ручної праці, а $З_п$ – затрати на розроблену програму:

$$Е = З_р - З_п, \quad (3.17)$$

$$Е = 95160 - 59068,5 = 36091,5 \text{ грн}$$

Термін окупності розраховується за формулою:

$$Т_{ок} = В_п + З_п \cdot \frac{З_н}{Е}, \quad (3.18)$$

$$Т_{ок} = 32000 + 36091,5 + 0 \div 48190 = 1,41 \approx 1,5 \text{ року}$$

Таким чином, із економічних розрахунків випливає, що впровадження спроєктованої системи є економічно доцільним і дозволить підвищити продуктивність праці працівника[14].

За незалежними дослідженнями, велика кількість користувачів Україні

використовують неліцензійне програмне забезпечення. При цьому всьому, скорочення ринку піратського програмного забезпечення відбувається всього на кілька відсотків у рік.

Проблема ускладнюється ще тим, що і в деяких організаціях до сих пір застосовується неліцензійні операційні системи та офісні програми. Для придбання ліцензійного ПЗ організації повинні закладати в бюджет істотні витрати на необхідну апаратну і програмну частину. В умовах кризи такі суми знайти не просто. Але, дуже важливо.

Щоб зрозуміти, для чого необхідно переходити на ліцензійне програмне забезпечення, користувач повинен знати всі переваги ліцензійного програмного забезпечення:

- юридичний захист вашого бізнесу. У зв’язку з діючими законами в Україні стосовно захисту інтелектуальної власності, за продаж піратського ПЗ і використання його в своїй діяльності, суб'єкт діяльності притягається до різних видів відповідальності. Тому наявність ліцензійного ПЗ - це запорука законності і чистоти бізнесу;

- технічна і сервісна підтримка виробника програмного забезпечення. Може включати весь перелік консультаційних послуг - передпродажне консультування, технічна підтримка, підтримка по телефону «гарячої лінії», своєчасні оновлення програмних продуктів, доступ до закритих форумів і чатів, індивідуальний профіль на сайті виробника, підписки на новини по email, актуальні версії драйверів і багато іншого;

- гарантія та повна впевненість в працездатності ліцензійної програми. У піратських версіях програм зазвичай модифікується програмний код, а також впроваджуються троянські програми, на які навіть ліцензійний оновлений антивірус може не реагувати. Це призводить до втрати конфіденційної інформації і несанкціонованого доступу до персонального комп'ютера користувача. Крім того, в роботі піратського ПЗ частіше трапляються критичні помилки з раптовим завершенням роботи, виникають помилки в деяких функціях ПЗ, можуть містити різні вразливості;

- постійні оновлення і нові версії програм. Піратська програма не має можливості постійно отримувати необхідні оновлення. Якщо оновлення ПЗ все-таки відбуваються, то правовласник (розробник) може віддалено заблокувати неліцензійне програмне забезпечення та повідомити користувача про різні способи придбання даної програми.;

- належність до сучасного і цивілізованого співтовариства. Використання ліцензійного програмного забезпечення означає дотримання законодавства, відповідність стандартам і вимогам при сертифікації, повагу праці розробників і прав інтелектуальної власності, повага до інтересів партнерів і замовників, а в кінцевому підсумку – самоповага;

- імідж та подальший розвиток підприємств та організацій. Робота з ліцензійним ПЗ є обов'язковою вимогою для проходження сертифікації організацією на відповідність вимогам міжнародних стандартів ISO;

- раціональне використання коштів. Витрати на відновлення інформаційної системи в разі її краху через піратського ПЗ можуть бути значно вище, ніж своєчасні витрати на покупку ліцензійних програмних засобів. Сукупна вартість володіння комплексом саме ліцензійних програмних засобів та інформаційних систем управління, завжди виявляється нижче, ніж при використанні піратського (нелегального) ПЗ[9].

Таким чином купуючи ліцензію можна уникнути багатьох проблем. Тому продаж кожноно ліцензійного продукту буде продаватися в роздріб по ціні розраховоній за формулою 2.19, де $Ц_{роздр.}$ – роздрібна ціна, $Ц_{опт}$ – оптова ціна, $Н_{роздр}$ – роздрібна надбавка(%).

$$Ц_{роздр} = Ц_{опт} + Ц_{опт} \cdot \frac{\%Н_{роздр}}{100}, \quad (3.19)$$

$$Ц_{роздр} = (59068{,}5 + 59068{,}5 \cdot 5\%) \div 100 = 62022 \text{ грн/п.}$$

Тепер розрахувавши роздрібну ціну за 1 одиницю ліцензійного продукту можна розпочати продаж товару. Ціна 1 одиниці товару коштуватиме по роздрібній ціні 62022 грн.

ВИСНОВКИ

У ході виконання даної кваліфікаційної роботи була розроблена 2D гра, призначена для створення захоплюючого ігрового досвіду для користувачів. З метою досягнення поставлених цілей, були використані сучасні технології та програмні засоби, такі як Godot Engine, GDScript та Visual Code для розробки ігрової логіки та анімацій. В процесі роботи були враховані вимоги до функціональних характеристик, графічної якості, інформаційної безпеки, а також вимоги до технічних та програмних засобів гри.

Результати роботи підтвердили, що розроблена гра забезпечує захоплюючий та інтуїтивно зрозумілий ігровий процес, а також дозволяє користувачам взаємодіяти з ігровим середовищем через різні елементи керування. Крім того, було забезпечено інтерактивність і зручність використання гри завдяки використанню зрозумілого інтерфейсу користувача та якісної графіки.

Основні висновки, отримані під час розробки 2D гри, полягають у тому, що ефективне використання сучасних технологій та програмних засобів дозволяє покращити якість та продуктивність ігрових проектів. Використання гри на основі Godot Engine допомагає створити привабливий ігровий продукт та забезпечує зручний доступ до розваг для користувачів.

Актуальність такої гри полягає в тому, що вона не тільки допомагає задовольнити попит на якісні та захоплюючі ігри, а й сприяє розвитку навичок програмування та творчого мислення у розробників.

Перспективи подальшого розвитку даної гри включають можливість розширення функціональності, таку як додавання нових рівнів та персонажів, створення онлайн-режимів для багатокористувацької гри, а також покращення інтерфейсу та оптимізацію продуктивності гри.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Аркада. URL: https://ru.wikipedia.org/wiki/Аркада. (дата звернення: 15.06.2024).

2. Платформер. URL: https://ru.wikipedia.org/wiki/Платформер. (дата звернення: 15.06.2024).

3. Документація «Godot Engine». URL: https://docs.godotengine.org/ru/stable/. (дата звернення: 15.06.2024).

4. Версії ОС Windows URL: https://ru.wikipedia.org/wiki/Windows. (дата звернення: 15.06.2024).

5. Версії ОС Android URL: https://ru.wikipedia.org/wiki/Android. (дата звернення: 15.06.2024).

6. Процесори. URL: https://ru.wikipedia.org/wiki/Процессор. (дата звернення: 22.06.2024).

7. Відеокарти. URL: https://ru.wikipedia.org/wiki/Видеокарта. (дата звернення: 22.06.2024).

8. Методичні вказівки по виконанню розділу «ЕР».

9. Розрахунок роздрібної ціни URL: https://journal.ostapp.com.ua/articles/post/marza-i-nacenka. (дата звернення: 15.06.2024).

10. Інформаційні системи URL: https://ru.wikipedia.org/wiki/Информационная_система. (дата звернення: 15.06.2024).

11. Векторна графіка URL: https://ru.wikipedia.org/wiki/Векторная_графика. (дата звернення: 15.06.2024).

12. Вартість розробки програмного забезпечення URL: https://ru.wikipedia.org/wiki/Стоимость_разработки_программного_обеспечения. (дата звернення: 15.06.2024).

13. Метод оцінки вартості проектів URL: https://ru.wikipedia.org/wiki/Метод_оценки_стоимости_проектов. (дата звернення: 15.06.2024).

14. Економічний аналіз проектів URL: https://ru.wikipedia.org/wiki/Экономический_анализ_проектов. (дата звернення: 15.06.2024).

15. Теорія ігор URL: https://ru.wikipedia.org/wiki/Теория_игр. (дата звернення: 15.06.2024).

16. Математичне моделювання в іграх URL: https://ru.wikipedia.org/wiki/Математическое_моделирование. (дата звернення: 15.06.2024).

17. Лінійна алгебра для розробників ігор URL: https://ru.wikipedia.org/wiki/Линейная_алгебра. (дата звернення: 15.06.2024).

18. Колізії та фізика в 2D іграх URL: https://docs.godotengine.org/ru/stable/tutorials/physics/2d_physics_introduction.html. (дата звернення: 15.06.2024).

19. Основи створення ігор URL: https://ru.wikipedia.org/wiki/Создание_компьютерных_игр. (дата звернення: 15.06.2024).

20. Алгоритми в ігровій розробці URL: https://ru.wikipedia.org/wiki/Алгоритмы. (дата звернення: 15.06.2024).

21. Основи програмування ігор URL: https://ru.wikipedia.org/wiki/Программирование_игр. (дата звернення: 15.06.2024).

22. Геймдизайн URL: https://ru.wikipedia.org/wiki/Геймдизайн. (дата звернення: 15.06.2024).

23. Графічні алгоритми URL: https://ru.wikipedia.org/wiki/Графические_алгоритмы. (дата звернення: 15.06.2024).

24. Git. URL: https://ru.wikipedia.org/wiki/Git. (дата звернення: 22.06.2024).

25. Історія комп'ютерних ігор URL: https://ru.wikipedia.org/wiki/История_компьютерных_игр. (дата звернення: 15.06.2024).

ДОДАТОК А

КОД ПРОГРАМИ

globals.gd:

```
extends Node
export(int) var level_attemps = 1
export(int) var coins = 0
var icon = load("res://icon.png")
func _ready():
 if data_handling.load_data("icon.dat"):
       icon = load(data_handling.load_data("icon.dat"))
 if data_handling.load_data("coins.dat"):
       coins = int(data_handling.load_data("coins.dat"))
const PREV = 'previous'
const NEXT = 'next'
const GRAVITY = Vector2(0, 1200)
const UP = Vector2(0, -1)
const GROUP_TILES = "tiles"
const RAY_LEN = 8
func modulo(num, mod):
return num - mod * floor(num / mod)

data_handling.gd:
extends Node

var file = File.new()
var directory = Directory.new()

func save_data(data, file_name):
 file.open("user://" + file_name, File.WRITE)
 file.store_var(data)
 file.close()

func load_data(file_name):
 file.open("user://" + file_name, File.READ)
 var data = file.get_var()
 file.close()
 return data
```

standart_state.gd:

```
extends '_i_state.gd'

export(int) var jump_strength = 300
export(int) var speed = 200

var key_maps = {
 jump = false
}

var motion = Vector2(0, 0)

func jump(host):
 motion.y = -jump_strength * host.gravity_vector

func enter(host):
 var ray_right = host.get_node("Rays/RayRight")
 var ray_right_2 = host.get_node("Rays/RayRight2")

 var ray_left = host.get_node("Rays/RayLeft")
```

```
	var ray_left_2 = host.get_node("Rays/RayLeft2")

	var target = host.get_node("Target")

	ray_right.cast_to = Vector2(globals.RAY_LEN, 0)
	ray_left.cast_to = Vector2(-globals.RAY_LEN, 0)

	ray_right_2.cast_to = Vector2(0, 0)
	ray_left_2.cast_to = Vector2(0, 0)

	target.position = Vector2(0, 0)

func handle_input(host, event):
	if event.is_action_pressed("move_jump"):
#		host.get_node("Rays/RayLeft").cast_to = Vector2(-8, 0)
#		host.get_node("Rays/RayRight").cast_to = Vector2(8, 0)
		key_maps.jump = true

	if event.is_action_released("move_jump"):
#		host.get_node("Rays/RayLeft").cast_to = Vector2(-4, 0)
#		host.get_node("Rays/RayRight").cast_to = Vector2(4, 0)
		key_maps.jump = false

	return

func update(host, delta):
	if not host.alive:
		return

	var ray_right_2 = host.get_node("Rays/RayRight2")
	var ray_left_2 = host.get_node("Rays/RayLeft2")

	var force = globals.GRAVITY * host.gravity_vector

	motion += force * delta

	motion.x = speed

	motion = host.move_and_slide(motion, globals.UP * host.gravity_vector)

	if key_maps.jump and host.is_on_floor():
		jump(host)
		ray_right_2.cast_to = Vector2(globals.RAY_LEN, -globals.RAY_LEN/4)
		ray_left_2.cast_to = Vector2(-globals.RAY_LEN, -globals.RAY_LEN/4)
	elif !key_maps.jump and host.is_on_floor():
		ray_right_2.cast_to = Vector2(0, 0)
		ray_left_2.cast_to = Vector2(0, 0)

	if not host.is_on_floor():
		var rotate_speed = deg2rad(5)

		host.get_node("icon").rotate(rotate_speed)

	if host.is_on_floor():
		var current_rotate =
globals.modulo(rad2deg(host.get_node("icon").rotation), 360)
		var current_right_rotate = globals.modulo(current_rotate, 90)

		var diff = current_rotate - current_right_rotate
		var new_rotation = diff + 90

		if not round(current_rotate) == diff:
```

```
            host.get_node("icon").rotation = deg2rad(new_rotation)

 return
```

Hero.gd:

```
extends KinematicBody2D

signal state_changed
signal died
signal die_timer

var states_stack = []
var current_state = null

var alive = true

var gravity_vector = 1

onready var states_map = {
 standard = $States/Standard,
 leftright = $States/LeftRight,
 flappy = $States/Flappy,
 stagger = $States/Stagger
}

func die():
 if alive:
       emit_signal("died")
 $Timers/DieTimer.start()
 $icon.hide()
 $DieParticles.emitting = true
 alive = false

func jump():
 if current_state.has_method("jump"):
       current_state.jump(self)

func change_gravity():
 gravity_vector *= -1

func check_ray_collide(ray):
 var collider = ray.get_collider()
 if collider and collider.is_in_group(globals.GROUP_TILES):
       die()

func cheack_finish():
 states_map.clear()

func _ready():
 $icon.texture = globals.icon
 alive = true
 states_stack.push_front($States/Standard)
 current_state = states_stack[0]
 _change_state('standard')

 $Rays/RayLeft.cast_to = Vector2(-8, 0)
 $Rays/RayRight.cast_to = Vector2(8, 0)
 $DieParticles.emitting = false
 $Target/icon.hide()

func _physics_process(delta):
```

```
 var state_name = current_state.update(self, delta)

 check_ray_collide($Rays/RayRight)
 check_ray_collide($Rays/RayRight2)
 check_ray_collide($Rays/RayLeft)
 check_ray_collide($Rays/RayLeft2)

 if state_name:
       _change_state(state_name)

func _input(event):
 var state_name = current_state.handle_input(self, event)

 if state_name:
       _change_state(state_name)

func _change_state(state_name):
 current_state.exit(self)

 if state_name == globals.PREV:
       states_stack.pop_front()
 else:
       var new_state = states_map[state_name]
       states_stack[0] = new_state

 current_state = states_stack[0]

 if state_name != globals.PREV:
       current_state.enter(self)

 emit_signal('state_changed', states_stack)

 pass

func _on_DieTimer_timeout():
 emit_signal("die_timer")
```

CameraHelper.gd:

```
extends Node2D

export(NodePath) var target_path = null
var target

var new_camera_zoom = 0.5
var new_camera_rotate = 0

var is_child_node = false

var pause = load("res://Objects/Menu/PauseMenu.tscn").instance()

func _ready():
 var temp_target = get_node(String(target_path) + "/Target")
 if not temp_target:
       target = get_node(target_path)
 else:
       is_child_node = true
       target = temp_target

 print(target)

 $icon.hide()
```

```
func _physics_process(delta):
 var target_position = target.position

 if is_child_node:
       target_position = target.position + target.get_parent().position

 position += (target_position - position + Vector2(64, 0)) * .5

 var current_zoom = $Camera2D.zoom[0]
 var new_zoom = lerp(current_zoom, new_camera_zoom, 0.05)
 $Camera2D.zoom = Vector2(new_zoom, new_zoom)

 var current_rotate = $Camera2D.rotation
 var new_rotate = lerp(current_rotate, new_camera_rotate, 0.1)
 $Camera2D.rotation = new_rotate
```

LevelBase.gd:

```
extends Node2D

const ATTEMPT = "Attempt "

func stop_music():
 $music.stop()

func _ready():

 $Hero.connect("died", self, "_on_hero_die")
 $Hero.connect("die_timer", self, "_on_hero_die_timer")
 $Attempt.text = ATTEMPT + String(globals.level_attemps)

func _physics_process(delta):
 $Shades.position = $Hero.position

func _on_hero_die():
 globals.level_attemps += 1
 stop_music()

func _on_hero_die_timer():
# warning-ignore:return_value_discarded
 get_tree().reload_current_scene()

LevelBase.gd:
extends Node2D

const ATTEMPT = "Attempt "

func stop_music():
 $music.stop()

func _ready():

 $Hero.connect("died", self, "_on_hero_die")
 $Hero.connect("die_timer", self, "_on_hero_die_timer")
 $Attempt.text = ATTEMPT + String(globals.level_attemps)

func _physics_process(delta):
 $Shades.position = $Hero.position

func _on_hero_die():
 globals.level_attemps += 1
```

```
 stop_music()

func _on_hero_die_timer():
# warning-ignore:return_value_discarded
 get_tree().reload_current_scene()
```

Trampoline.gd:

```
extends Area2D

func _on_Trampoline_body_entered(body):
 if body.is_in_group("hero"):
        body.jump()
```

```
GravitiChanger.gd:
extends Area2D

func _on_GravityChanger_body_entered(body):
 if body.is_in_group("hero"):
        body.change_gravity()
```

```
SpikeNeon.gd:
extends Area2D

func _on_SpikeNeon_body_entered(body):
 if body.is_in_group("hero"):
        body.die()
```

MainMenu.gd:

```
extends Control

var nextScene

func _ready():
 pass

func _on_ButtonPlay_pressed():
 nextScene = load("res://Objects/Menu/LevelsMenu.tscn").instance()
 get_tree().current_scene.add_child(nextScene)

func _on_ButtonShop_pressed():
 nextScene = load("res://Objects/Menu/ShopMenu.tscn").instance()
 get_tree().current_scene.add_child(nextScene)

func _on_ButtonQuite_pressed():
 get_tree().quit()
```

```
LevelsMenu.gd:
extends Control

func _ready():
 pass
func _on_ButtonLvl1_pressed():
 get_tree().change_scene("res://Objects/Level/LevelBase.tscn")
func _on_Back_pressed():
 hide()
```

ShopMenu.gd:

```
extends Control

var icon1 = load("res://gfx/icons/icon_1.png")
```

```
var icon2 = load("res://gfx/icons/icon_2.png")
var icon3 = load("res://gfx/icons/icon_3.png")

func _ready():
 $Coins.text = String(globals.coins)
 if globals.coins >= 1:
       $Buttons/ButtonsBuyHero/Hero1.disabled = false
 if globals.coins >= 5:
       $Buttons/ButtonsBuyHero/Hero2.disabled = false
 if globals.coins >= 10:
       $Buttons/ButtonsBuyHero/Hero3.disabled = false


func _on_Back_pressed():
 hide()

func _on_Hero1_pressed():
 globals.icon = icon1
 data_handling.save_data("res://gfx/icons/icon_1.png", "icon.dat")

func _on_Hero2_pressed():
 globals.icon = icon2
 data_handling.save_data("res://gfx/icons/icon_2.png", "icon.dat")

func _on_Hero3_pressed():
 globals.icon = icon3
 data_handling.save_data("res://gfx/icons/icon_3.png", "icon.dat")
```

OptionsMenu.gd:

```
extends Control

var defaultColor = Color(0, 0, 1)
var buttonMenuPressed = true
var buttonGamePressed = true

func _ready():
 $Textures/Background.modulate = globals.color
 if globals.color == Color(0, 1, 0):
       $Buttons/GameMusicScrollBar.modulate = globals.color
       $Buttons/MenuMusicScrollBar.modulate = globals.color
 else:
       $Buttons/GameMusicScrollBar.modulate = defaultColor
       $Buttons/MenuMusicScrollBar.modulate = defaultColor
 $Buttons/MenuMusicScrollBar.value = globals.volumeMenu
 $Buttons/GameMusicScrollBar.value = globals.volumeGame
 if data_handling.load_data("button_menu.dat"):
       buttonMenuPressed                                                    =
bool(int(data_handling.load_data("button_menu.dat")))
 if data_handling.load_data("button_game.dat"):
       buttonGamePressed                                                    =
bool(int(data_handling.load_data("button_game.dat")))
 $Buttons/MenuMusicCheckButton.pressed = buttonMenuPressed
 $Buttons/GameMusicCheckButton.pressed = buttonGamePressed

func _process(delta):
 if $Buttons/MenuMusicCheckButton.pressed:
       $Buttons/MenuMusicScrollBar.visible = true
       buttonMenuPressed = true
       data_handling.save_data("1", "button_menu.dat")
 else:
       $Buttons/MenuMusicScrollBar.visible = false
```

```
        globals.volumeMenu = -80
        data_handling.save_data(globals.volumeMenu, "volume_menu.dat")
        buttonMenuPressed = false
        data_handling.save_data("0", "button_menu.dat")
 if $Buttons/GameMusicCheckButton.pressed:
        $Buttons/GameMusicScrollBar.visible = true
        buttonGamePressed = true
        data_handling.save_data("1", "button_game.dat")
 else:
        $Buttons/GameMusicScrollBar.visible = false
        globals.volumeGame = -80
        data_handling.save_data(globals.volumeGame, "volume_game.dat")
        buttonGamePressed = false
        data_handling.save_data("0", "button_game.dat")

func _on_MenuMusicScrollBar_value_changed(value):
 globals.volumeMenu = value
 data_handling.save_data(globals.volumeMenu, "volume_menu.dat")

func _on_GameMusicScrollBar_value_changed(value):
 globals.volumeGame = value
 data_handling.save_data(globals.volumeGame, "volume_game.dat")

func _on_Back_pressed():
 get_tree().reload_current_scene()
```

PauseMenu.gd:

```
extends Control

var is_paused = false setget set_paused
onready var camera_pos = get_parent().get_node("Camera2D")

func _physics_process(delta):
 set_position(Vector2(camera_pos.position.x - 50, camera_pos.position.y - 2))

func _ready():
 pass

func set_paused(value):
 is_paused = value
 get_tree().paused = is_paused
 visible = value

func _unhandled_input(event):

 if event.is_action_pressed("set_pasue"):
        set_paused(!is_paused)

func _on_Resume_pressed():
 set_paused(false)

func _on_Restart_pressed():
 get_tree().change_scene("res://Objects/Level/LevelBase.tscn")
 set_paused(false)

func _on_MainMenu_pressed():
 get_tree().change_scene("res://Objects/Menu/MainMenu.tscn")
 set_paused(false)

func _on_Qiut_pressed():
 get_tree().quit()
```

FinishMenu.gd:

```
extends Control

func _ready():
 pass

func _on_ButtonMainMenu_pressed():
 get_tree().change_scene("res://Objects/Menu/MainMenu.tscn")

func _on_Exit_pressed():
 get_tree().quit()
```

Chest.gd:

```
extends Area2D


func _ready():
 pass


func _physics_process(delta):
 rotation_degrees += 1


func _on_Coin_body_entered(body):
 if body.is_in_group("hero"):
        globals.coins += 5
        hide()
```

Coin.gd:

```
extends Area2D


func _ready():
 pass


func _physics_process(delta):
 rotation_degrees += 1


func _on_Coin_body_entered(body):
 if body.is_in_group("hero"):
        globals.coins += 1
        hide()
```

TriggerColor.gd:

```
extends "_TriggerBase.gd"


export(Color) var color_modulate
```

```
func _ready():
 ._ready()

func _on__TriggerBase_body_entered(body):
 if body.is_in_group("hero"):
        target.modulate = color_modulate
```

TriggerScreenInventory.gd:

```
extends "_TriggerBase.gd"

export(float, 0, 1, 0.1) var new_intensity

func _ready():
 ._ready()

func _on__TriggerBase_body_entered(body):
 if body.is_in_group("hero"):
        target.new_intensity = new_intensity
```

TriggerSpeed.gd:

```
extends "_TriggerBase.gd"

export(float, 0, 1, 0.1) var new_intensity

func _ready():
 ._ready()

func _on__TriggerBase_body_entered(body):
 if body.is_in_group("hero"):
        target.new_intensity = new_intensity
```

TriggerZoom.gd:

```
extends "_TriggerBase.gd"

export(float, 0, 2, 0.1) var new_zoom = 0.5

func _ready():
 ._ready()

func _on__TriggerBase_body_entered(body):
 if body.is_in_group("hero"):
```

```
        target.new_camera_zoom = new_zoom
```

_TriggerBase.gd:

```
extends Area2D

var target
export(NodePath) var target_path

func _ready():
 target = get_node(target_path)

 $Label.hide()

func _on__TriggerBase_body_entered(body):
 pass # replace with function body
```

Spike.gd:

```
extends Area2D

func _on_SpikeNeon_body_entered(body):
 if body.is_in_group("hero"):
        body.die()
```

GravityChanger.gd:

```
extends Area2D

func _on_GravityChanger_body_entered(body):
 if body.is_in_group("hero"):
        body.change_gravity()
```

data_handling.gd:

```
extends Node

var file = File.new()

func save_data(data, file_name):
 file.open("user://" + file_name, File.WRITE)
 file.store_var(data)
 file.close()

func load_data(file_name):
```

```
 file.open("user://" + file_name, File.READ)
 var data = file.get_var()
 file.close()
 return data
```

_i_state.gd:

```
extends Node


# Initialize the state. E.g. change the animation
func enter(host):
 return

# Clean up the state. Reinitialize values like a timer
func exit(host):
 return

func handle_input(host, event):
 return

func update(host, delta):
 return

func _on_animation_finished(anim_name):
 return
```

tiled_xml_to_dict.gd:

```
tool
extends Reference

# Reads a TMX file from a path and return a Dictionary with the same structure
# as the JSON map format
# Returns an error code if failed
func read_tmx(path):
 var parser = XMLParser.new()
 var err = parser.open(path)
 if err != OK:
        printerr("Error opening TMX file '%s'." % [path])
        return err

 while parser.get_node_type() != XMLParser.NODE_ELEMENT:
```

```
		err = parser.read()
		if err != OK:
			printerr("Error parsing TMX file '%s' (around line %d)." % [path,
parser.get_current_line()])
			return err

	if parser.get_node_name().to_lower() != "map":
		printerr("Error parsing TMX file '%s'. Expected 'map' element.")
		return ERR_INVALID_DATA

	var data = attributes_to_dict(parser)
	if not "infinite" in data:
		data.infinite = false
	data.type = "map"
	data.tilesets = []
	data.layers = []

	err = parser.read()
	if err != OK:
		printerr("Error parsing TMX file '%s' (around line %d)." % [path,
parser.get_current_line()])
		return err

	while err == OK:
		if parser.get_node_type() == XMLParser.NODE_ELEMENT_END:
			if parser.get_node_name() == "map":
				break
		elif parser.get_node_type() == XMLParser.NODE_ELEMENT:
			if parser.get_node_name() == "tileset":
				# Empty element means external tileset
				if not parser.is_empty():
					var tileset = parse_tileset(parser)
					if typeof(tileset) != TYPE_DICTIONARY:
						# Error happened
						return err
					data.tilesets.push_back(tileset)
				else:
					var tileset_data = attributes_to_dict(parser)
					if not "source" in tileset_data:
						printerr("Error parsing TMX file '%s'. Missing
tileset source (around line %d)." % [path, parser.get_current_line()])
```

```
                                return ERR_INVALID_DATA
                            data.tilesets.push_back(tileset_data)

                elif parser.get_node_name() == "layer":
                        var layer = parse_tile_layer(parser, data.infinite)
                        if typeof(layer) != TYPE_DICTIONARY:
                                printerr("Error parsing TMX file '%s'. Invalid tile
layer data (around line %d)." % [path, parser.get_current_line()])
                                return ERR_INVALID_DATA
                        data.layers.push_back(layer)

                elif parser.get_node_name() == "imagelayer":
                        var layer = parse_image_layer(parser)
                        if typeof(layer) != TYPE_DICTIONARY:
                                printerr("Error parsing TMX file '%s'. Invalid image
layer data (around line %d)." % [path, parser.get_current_line()])
                                return ERR_INVALID_DATA
                        data.layers.push_back(layer)

                elif parser.get_node_name() == "objectgroup":
                        var layer = parse_object_layer(parser)
                        if typeof(layer) != TYPE_DICTIONARY:
                                printerr("Error parsing TMX file '%s'. Invalid object
layer data (around line %d)." % [path, parser.get_current_line()])
                                return ERR_INVALID_DATA
                        data.layers.push_back(layer)

                elif parser.get_node_name() == "group":
                        var layer = parse_group_layer(parser, data.infinite)
                        if typeof(layer) != TYPE_DICTIONARY:
                                printerr("Error parsing TMX file '%s'. Invalid group
layer data (around line %d)." % [path, parser.get_current_line()])
                                return ERR_INVALID_DATA
                        data.layers.push_back(layer)

                elif parser.get_node_name() == "properties":
                        var prop_data = parse_properties(parser)
                        if typeof(prop_data) == TYPE_STRING:
                                return prop_data

                        data.properties = prop_data.properties
```

```
                        data.propertytypes = prop_data.propertytypes

                err = parser.read()

        return data

    # Reads a TSX and return a tileset dictionary
    # Returns an error code if fails
    func read_tsx(path):
        var parser = XMLParser.new()
        var err = parser.open(path)
        if err != OK:
                printerr("Error opening TSX file '%s'." % [path])
                return err

        while parser.get_node_type() != XMLParser.NODE_ELEMENT:
                err = parser.read()
                if err != OK:
                        printerr("Error parsing TSX file '%s' (around line %d)." % [path,
parser.get_current_line()])
                        return err

        if parser.get_node_name().to_lower() != "tileset":
                printerr("Error parsing TMX file '%s'. Expected 'map' element.")
                return ERR_INVALID_DATA

        var tileset = parse_tileset(parser)

        return tileset

    # Parses a tileset element from the XML and return a dictionary
    # Return an error code if fails
    func parse_tileset(parser):
        var err = OK
        var data = attributes_to_dict(parser)
        data.tiles = {}

        err = parser.read()
        while err == OK:
                if parser.get_node_type() == XMLParser.NODE_ELEMENT_END:
                        if parser.get_node_name() == "tileset":
```

```
                        break

            elif parser.get_node_type() == XMLParser.NODE_ELEMENT:
                  if parser.get_node_name() == "tile":
                        var attr = attributes_to_dict(parser)
                        var tile_data = parse_tile_data(parser)
                        if typeof(tile_data) != TYPE_DICTIONARY:
                              # Error happened
                              return tile_data
                        if "properties" in tile_data and "propertytypes" in
tile_data:
                              if not "tileproperties" in data:
                                    data.tileproperties = {}
                                    data.tilepropertytypes = {}
                              data.tileproperties[str(attr.id)]                    =
tile_data.properties
                              data.tilepropertytypes[str(attr.id)]                 =
tile_data.propertytypes
                              tile_data.erase("tileproperties")
                              tile_data.erase("tilepropertytypes")
                        data.tiles[str(attr.id)] = tile_data

                  elif parser.get_node_name() == "image":
                        var attr = attributes_to_dict(parser)
                        if not "source" in attr:
                              printerr("Error loading image tag. No source attribute
found (around line %d)." % [parser.get_current_line()])
                              return ERR_INVALID_DATA
                        data.image = attr.source
                        if "width" in attr:
                              data.imagewidth = attr.width
                        if "height" in attr:
                              data.imageheight = attr.height

                  elif parser.get_node_name() == "properties":
                        var prop_data = parse_properties(parser)
                        if typeof(prop_data) != TYPE_DICTIONARY:
                              # Error happened
                              return prop_data

                        data.properties = prop_data.properties
```

```
                        data.propertytypes = prop_data.propertytypes

                err = parser.read()

        return data


      # Parses the data of a single tile from the XML and return a dictionary
      # Returns an error code if fails
      func parse_tile_data(parser):
        var err = OK
        var data = {}
        var obj_group = {}
        if parser.is_empty():
                return data

        err = parser.read()
        while err == OK:

                if parser.get_node_type() == XMLParser.NODE_ELEMENT_END:
                        if parser.get_node_name() == "tile":
                                return data
                        elif parser.get_node_name() == "objectgroup":
                                data.objectgroup = obj_group

                elif parser.get_node_type() == XMLParser.NODE_ELEMENT:
                        if parser.get_node_name() == "image":
                                # If there are multiple images in one tile we only use the
last one.
                                var attr = attributes_to_dict(parser)
                                if not "source" in attr:
                                        printerr("Error loading image tag. No source attribute
found (around line %d)." % [parser.get_current_line()])
                                        return ERR_INVALID_DATA
                                data.image = attr.source
                                data.imagewidth = attr.width
                                data.imageheight = attr.height

                        elif parser.get_node_name() == "objectgroup":
                                obj_group = attributes_to_dict(parser)
                                for attr in ["width", "height", "offsetx", "offsety"]:
```

```
                        if not attr in obj_group:
                            data[attr] = 0
                    if not "opacity" in data:
                        data.opacity = 1
                    if not "visible" in data:
                        data.visible = true
                    if parser.is_empty():
                        data.objectgroup = obj_group

                elif parser.get_node_name() == "object":
                    if not "objects" in obj_group:
                        obj_group.objects = []
                    var obj = parse_object(parser)
                    if typeof(obj) != TYPE_DICTIONARY:
                        # Error happened
                        return obj
                    obj_group.objects.push_back(obj)

                elif parser.get_node_name() == "properties":
                    var prop_data = parse_properties(parser)
                    data["properties"] = prop_data.properties
                    data["propertytypes"] = prop_data.propertytypes

        err = parser.read()

 return data

# Parses the data of a single object from the XML and return a dictionary
# Returns an error code if fails
static func parse_object(parser):
 var err = OK
 var data = attributes_to_dict(parser)

 if not parser.is_empty():
        err = parser.read()
        while err == OK:
            if parser.get_node_type() == XMLParser.NODE_ELEMENT_END:
                if parser.get_node_name() == "object":
                    break

            elif parser.get_node_type() == XMLParser.NODE_ELEMENT:
```

```
                        if parser.get_node_name() == "properties":
                                var prop_data = parse_properties(parser)
                                data["properties"] = prop_data.properties
                                data["propertytypes"] = prop_data.propertytypes

                        elif parser.get_node_name() == "point":
                                data.point = true

                        elif parser.get_node_name() == "ellipse":
                                data.ellipse = true

                        elif     parser.get_node_name()     ==     "polygon"     or
parser.get_node_name() == "polyline":
                                var points = []
                                var                    points_raw                    =
parser.get_named_attribute_value("points").split(" ", false, 0)

                                for pr in points_raw:
                                        points.push_back({
                                                "x": float(pr.split(",")[0]),
                                                "y": float(pr.split(",")[1]),
                                        })

                                data[parser.get_node_name()] = points

                err = parser.read()

        return data


    # Parses a tile layer from the XML and return a dictionary
    # Returns an error code if fails
    func parse_tile_layer(parser, infinite):
        var err = OK
        var data = attributes_to_dict(parser)
        data.type = "tilelayer"
        if not "x" in data:
                data.x = 0
        if not "y" in data:
                data.y = 0
        if infinite:
```

```
		data.chunks = []
	else:
		data.data = []

	var current_chunk = null
	var encoding = ""

	if not parser.is_empty():
		err = parser.read()

		while err == OK:
			if parser.get_node_type() == XMLParser.NODE_ELEMENT_END:
				if parser.get_node_name() == "layer":
					break
				elif parser.get_node_name() == "chunk":
					data.chunks.push_back(current_chunk)
					current_chunk = null

			elif parser.get_node_type() == XMLParser.NODE_ELEMENT:
				if parser.get_node_name() == "data":
					var attr = attributes_to_dict(parser)

					if "compression" in attr:
						data.compression = attr.compression

					if "encoding" in attr:
						encoding = attr.encoding
						if attr.encoding != "csv":
							data.encoding = attr.encoding

						if not infinite:
							err = parser.read()
							if err != OK:
								return err

							if attr.encoding != "csv":
								data.data                       =
parser.get_node_data().strip_edges()
							else:
								var             csv             =
parser.get_node_data().split(",", false)
```

```
                                                for v in csv:

        data.data.push_back(int(v.strip_edges()))

                            elif parser.get_node_name() == "tile":
                                  var                          gid                          =
int(parser.get_named_attribute_value_safe("gid"))
                                  if infinite:
                                        current_chunk.data.push_back(gid)
                                  else:
                                        data.data.push_back(gid)

                            elif parser.get_node_name() == "chunk":
                                  current_chunk = attributes_to_dict(parser)
                                  current_chunk.data = []
                                  if encoding != "":
                                        err = parser.read()
                                        if err != OK:
                                              return err
                                        if encoding != "csv":
                                              current_chunk.data                       =
parser.get_node_data().strip_edges()
                                        else:
                                              var                 csv                 =
parser.get_node_data().split(",", false)
                                              for v in csv:

        current_chunk.data.push_back(int(v.strip_edges()))

                            elif parser.get_node_name() == "properties":
                                  var prop_data = parse_properties(parser)
                                  if typeof(prop_data) == TYPE_STRING:
                                        return prop_data

                                  data.properties = prop_data.properties
                                  data.propertytypes = prop_data.propertytypes

                      err = parser.read()

        return data
```

```
# Parses an object layer from the XML and return a dictionary
# Returns an error code if fails
func parse_object_layer(parser):
 var err = OK
 var data = attributes_to_dict(parser)
 data.type = "objectgroup"
 data.objects = []

 if not parser.is_empty():
       err = parser.read()
       while err == OK:
             if parser.get_node_type() == XMLParser.NODE_ELEMENT_END:
                   if parser.get_node_name() == "objectgroup":
                         break
             if parser.get_node_type() == XMLParser.NODE_ELEMENT:
                   if parser.get_node_name() == "object":
                         data.objects.push_back(parse_object(parser))
                   elif parser.get_node_name() == "properties":
                         var prop_data = parse_properties(parser)
                         if typeof(prop_data) != TYPE_DICTIONARY:
                               # Error happened
                               return prop_data
                         data.properties = prop_data.properties
                         data.propertytypes = prop_data.propertytypes

             err = parser.read()

 return data

# Parses an image layer from the XML and return a dictionary
# Returns an error code if fails
func parse_image_layer(parser):
 var err = OK
 var data = attributes_to_dict(parser)
 data.type = "imagelayer"
 data.image = ""

 if not parser.is_empty():
       err = parser.read()
```

```
		while err == OK:
			if parser.get_node_type() == XMLParser.NODE_ELEMENT_END:
				if parser.get_node_name().to_lower() == "imagelayer":
					break
			elif parser.get_node_type() == XMLParser.NODE_ELEMENT:
				if parser.get_node_name().to_lower() == "image":
					var image = attributes_to_dict(parser)
					if not image.has("source"):
						printerr("Missing source attribute in
imagelayer (around line %d)." % [parser.get_current_line()])
						return ERR_INVALID_DATA
					data.image = image.source

				elif parser.get_node_name() == "properties":
					var prop_data = parse_properties(parser)
					if typeof(prop_data) != TYPE_DICTIONARY:
						# Error happened
						return prop_data
					data.properties = prop_data.properties
					data.propertytypes = prop_data.propertytypes

			err = parser.read()

	return data

# Parses a group layer from the XML and return a dictionary
# Returns an error code if fails
func parse_group_layer(parser, infinite):
	var err = OK
	var result = attributes_to_dict(parser)
	result.type = "group"
	result.layers = []

	if not parser.is_empty():
		err = parser.read()

		while err == OK:
			if parser.get_node_type() == XMLParser.NODE_ELEMENT_END:
				if parser.get_node_name().to_lower() == "group":
					break
			elif parser.get_node_type() == XMLParser.NODE_ELEMENT:
```

```
                        if parser.get_node_name() == "layer":
                                var layer = parse_tile_layer(parser, infinite)
                                if typeof(layer) != TYPE_DICTIONARY:
                                        printerr("Error parsing TMX file. Invalid tile
layer data (around line %d)." % [parser.get_current_line()])
                                        return ERR_INVALID_DATA
                                result.layers.push_back(layer)

                        elif parser.get_node_name() == "imagelayer":
                                var layer = parse_image_layer(parser)
                                if typeof(layer) != TYPE_DICTIONARY:
                                        printerr("Error parsing TMX file. Invalid image
layer data (around line %d)." % [parser.get_current_line()])
                                        return ERR_INVALID_DATA
                                result.layers.push_back(layer)

                        elif parser.get_node_name() == "objectgroup":
                                var layer = parse_object_layer(parser)
                                if typeof(layer) != TYPE_DICTIONARY:
                                        printerr("Error  parsing  TMX  file.  Invalid
object layer data (around line %d)." % [parser.get_current_line()])
                                        return ERR_INVALID_DATA
                                result.layers.push_back(layer)

                        elif parser.get_node_name() == "group":
                                var layer = parse_group_layer(parser, infinite)
                                if typeof(layer) != TYPE_DICTIONARY:
                                        printerr("Error parsing TMX file. Invalid group
layer data (around line %d)." % [parser.get_current_line()])
                                        return ERR_INVALID_DATA
                                result.layers.push_back(layer)

                        elif parser.get_node_name() == "properties":
                                var prop_data = parse_properties(parser)
                                if typeof(prop_data) == TYPE_STRING:
                                        return prop_data

                                result.properties = prop_data.properties
                                result.propertytypes = prop_data.propertytypes

                err = parser.read()
```

```
	return result

	# Parses properties data from the XML and return a dictionary
	# Returns an error code if fails
	static func parse_properties(parser):
		var err = OK
		var data = {
			"properties": {},
			"propertytypes": {},
		}

		if not parser.is_empty():
			err = parser.read()

			while err == OK:
				if parser.get_node_type() == XMLParser.NODE_ELEMENT_END:
					if parser.get_node_name() == "properties":
						break
				elif parser.get_node_type() == XMLParser.NODE_ELEMENT:
					if parser.get_node_name() == "property":
						var prop_data = attributes_to_dict(parser)
						if not (prop_data.has("name") and prop_data.has("value")):
							printerr("Missing information in custom properties (around line %d)." % [parser.get_current_line()])
							return ERR_INVALID_DATA

						data.properties[prop_data.name] = prop_data.value
						if prop_data.has("type"):
							data.propertytypes[prop_data.name] = prop_data.type
						else:
							data.propertytypes[prop_data.name] = "string"

				err = parser.read()

		return data

	# Reads the attributes of the current element and return them as a dictionary
	static func attributes_to_dict(parser):
		var data = {}
```

```
 for i in range(parser.get_attribute_count()):
        var attr = parser.get_attribute_name(i)
        var val = parser.get_attribute_value(i)
        if val.is_valid_integer():
               val = int(val)
        elif val.is_valid_float():
               val = float(val)
        elif val == "true":
               val = true
        elif val == "false":
               val = false
        data[attr] = val
 return data
```

polygon_sorter.gd:

```
tool
extends Reference

var center

# Sort the vertices of a convex polygon to clockwise order
# Receives a PoolVector2Array and returns a new one
func sort_polygon(vertices):
 vertices = Array(vertices)

 var centroid = Vector2()
 var size = vertices.size()

 for i in range(0, size):
        centroid += vertices[i]

 centroid /= size

 center = centroid
 vertices.sort_custom(self, "is_less")

 return PoolVector2Array(vertices)

# Sorter function, determines which of the poins should come first
func is_less(a, b):
 if a.x - center.x >= 0 and b.x - center.x < 0:
```

```
            return false
      elif a.x - center.x < 0 and b.x - center.x >= 0:
            return true
      elif a.x - center.x == 0 and b.x - center.x == 0:
            if a.y - center.y >= 0 or b.y - center.y >= 0:
                  return a.y < b.y
            return a.y > b.y

      var det = (a.x - center.x) * (b.y - center.y) - (b.x - center.x) * (a.y -
center.y)
      if det > 0:
            return true
      elif det < 0:
            return false

      var d1 = (a - center).length_squared()
      var d2 = (b - center).length_squared()

      return d1 < d2
```

ДОДАТОК Б

ВІДГУК

Керівника економічного розділу

на кваліфікаційну роботу бакалавра на тему:

«Розробка кросплатформеної гри Jumping cube в жанрі платформера на базі GodotEngine та мови програмування GDScript»

Студента групи 122-20-3 Бруя Владислава Валерійовича

Керівник економічного розділу
доц. каф. ПЕП та ПУ, к.е.н **Л.В. Касьяненко**

ДОДАТОК В

ПЕРЕЛІК ФАЙЛІВ НА ДИСКУ

Ім’я файлу	Опис
Пояснювальні документи	
Бруй.doc	Пояснювальна записка до кваліфікаційної роботи. Документ Word.
Бруй.pdf	Пояснювальна записка до кваліфікаційної роботи в форматі PDF
Програма	
Бруй.rar	Архів. Містить усі файли програми
Презентація	
Бруй.ppt	Презентація кваліфікаційної роботи