МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра програмної інженерії

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

на тему: **«РОЗРОБКА ІНТЕРАКТИВНИХ ГРАФІЧНИХ ЗАСТОСУНКІВ ІЗ ЗАСТОСУВАННЯМ UNITY»**

Виконав: студент 4 курсу, групи 6.1210-2пі
спеціальності 121 інженерія програмного забезпечення
(шифр і назва спеціальності)
освітньої програми програмна інженерія
(назва освітньої програми)
М.В. Слободян
(ініціали та прізвище)

Керівник декан математичного факультету, професор, д.т.н. Гоменюк С.І.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Рецензент завідувач кафедри фундаментальної та прикладної математики, професор, д.т.н. Гребенюка С.М.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя – 2024

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра програмної інженерії

Рівень вищої освіти бакалавр

Спеціальність 121 інженерія програмного забезпечення
(шифр і назва)

Освітня програма програмна інженерія

ЗАТВЕРДЖУЮ
Завідувач кафедри програмної інженерії, к.ф.-м.н., доцент

_____ Лісняк А.О.
(підпис)

“ ____ ” _____ 2023 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Слободяну Микиті Володимировичу
(прізвище, ім’я та по-батькові)

1. Тема роботи Розробка інтерактивних графічних застосунків із застосуванням Unity

керівник роботи Гоменюк Сергій Іванович, д.т.н., професор
(прізвище, ім’я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 21 » грудня 2023 року № 2180-с

2. Строк подання студентом роботи 03.06.2024 р.

3. Вихідні дані до роботи 1. Постановка задачі.
2. Перелік літератури.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Постановка задачі.
2. Основні теоретичні відомості.
3. Розробка інтерактивної гри.

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)
презентація за темою доповіді

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання 25.12.2023 р.

К А Л Е Н Д А Р Н И Й П Л А Н

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Розробка плану роботи.	09.01.2024	
2.	Збір вихідних даних.	15.02.2024	
3.	Обробка методичних та теоретичних	11.03.2024	
	джерел.		
4.	Розробка першого та другого розділу.	25.04.2024	
5.	Розробка третього розділу.	20.05.2024	
6.	Оформлення та нормоконтроль	27.05.2024	
	кваліфікаційної роботи бакалавра.		
7.	Захист кваліфікаційної роботи.	21.06.2024	

Студент _____ М.В. Слободян
(підпис) (ініціали та прізвище)

Керівник роботи _____ С.І. Гоменюк
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

Нормоконтролер _____ А.В. Столярова
(підпис) (ініціали та прізвище)

РЕФЕРАТ

Кваліфікаційна робота бакалавра «Розробка інтерактивних графічних застосунків із застосуванням Unity»: 67 с., 32 рис., 24 джерела, 1 додаток.

ГРАФІЧНИЙ ЗАСТОСУНОК, ІГРОВИЙ ДВИГУН, КРИВА НАВЧАННЯ, ПРОГРАМУВАННЯ НА C#, РОЗРОБКА ГРИ, СТВОРЕННЯ СПРАЙТІВ, СЦЕНАРІЙ, ЮНІТІ.

Об’єкт дослідження – кросплатформове середовище розробки комп’ютерних ігор Unity.

Мета роботи: дослідити можливість створення 2D ігор використовуючи кросплатформове середовище розробки комп’ютерних ігор Unity.

Метод дослідження – моделювання.

У кваліфікаційній роботі викладено теоретичні відомості про інструменти для розробки комп’ютерних ігор. Використовуючи дані, отримані за час дослідження методичних та теоретичних джерел, було спроєктовано та розроблено комп’ютерну гру у 2D-вимірі. Сетинг гри – середньовічний фентезійний світ у 2D.

SUMMARY

Bachelor’s qualifying paper “Development of Interactive Graphical Applications Using Unity”: 67 pages, 32 figures, 24 references, 1 supplement.

GRAPHIC APPLICATION, GAME ENGINE, LEARNING CURVE, C# PROGRAMMING, GAME DEVELOPMENT, CREATION OF SPIRES, SCRIPTS, UNITY.

The object of the study is the Unity cross-platform computer game development environment.

The aim of the study is investigate the possibility of creating 2D games using the cross-platform Unity computer game development environment.

The method of research is modeling.

The qualification paper contains theoretical information about tools for developing computer games. Using the data obtained during the research of methodological and theoretical sources, a computer game in 2D dimension was designed and developed. The setting of the game is a medieval fantasy world in 2D.

ЗМІСТ

ВСТУП

Дипломна робота є невід'ємною складовою частиною процесу професійної підготовки бакалаврів. Дипломна робота передбачає отримання необхідного обсягу практичних знань та умінь відповідно до кваліфікаційного рівня бакалавра.

Починаючи з давніх часів люди постійно вигадували різноманітні пристрої, або як ми називаємо їх зараз, гаджети, які вони використовували для задач різного характеру починаючи зі списів для полювання та плодозбирачів для збору плодів з високих дерев, закінчуючи інструментами для видобування руди та настільними іграми. Якщо зі списами та різними інструментами зрозуміло, тоді є питання стосовно настільних ігор. Навіщо людині взагалі ігри. Оскільки раніше єдиним способом відволіктися від повсякденного життя було спостереження за природними явищами, людина вирішила вигадати собі інші способи розбавити рутину. Саме тому з'явилися такі ігри як шахи, футбол, карти, хокей, крикет, пінг-понг та багато інших настільних та спортивних ігор.

З часом, після другої світової війни, коли люди створили таку неймовірну систему як комп'ютер, вони вирішили перенести ігри з реального світу, у віртуальний. З цього моменту почався розвиток комп'ютерних ігор. Однією з найперших комп'ютерних ігор була гра під назвою «Pong», сенс якої дуже простий. Гравець змагається з комп'ютером відбиваючи м'яч, який зображено як крапку, ракеткою, яка зображена як лінія. Хто заб'є більше голів, той виграв. Ця гра була досить популярною серед старшокласників і студентів та увійшла в історію, як найперша комп'ютерна гра. На цьому етапі почалась розробка грального автоматів, а згодом грального консолей.

Найпопулярнішою гральною консоллю була «Atari 2600» яка була розроблена американською компанією Atari, inc. у 1972 році. Саме на цій приставці виросло багато поколінь геймерів. Вона стала популярною завдяки

її технічним характеристикам та різноманіттям додаткових гаджетів, таких, як світловий пістолет, додаткові джойстики, кермо та трекбол. Основною її особливістю було те, що вона мала змінні картриджі з різними іграми, на відміну від її попередників в які було вбудована невелика кількість ігор. Бестселером цієї приставки була гра «Pac-Man» розроблена співробітником японської компанії «Namco» Тору Іватані. Гра отримала свою популярність завдяки тому, що у ній не було насильства, тим самим захопивши увагу жіночої аудиторії та лояльність батьків, що призвело до підвищення інтересу серед людей різного віку.

Трохи пізніше, у 1997 році невеличка, на той час, компанія «Valve», заснована вихідцями з компанії Microsoft Гейбом Ньюеллом та Майком Харрингтоном, публічно продемонструвала свою першу гру «Half-Life» на всесвітній виставці E3. Саме ця подія стала новим витком у світі створення ігор. Нова система анімацій та ігровий штучний інтелект, що для того часу здавався чимось неймовірним, вразив усіх гравців. У 1998 році на виставці E3 гра отримала нагороди від ігрових критиків як «Найкраща ПК-гра» та «Найкраща екшен-гра». Починаючи з цього моменту ігри почали стрімко розвиватись та вдосконалюватись, почали з’являтись нові механіки, стилі, сюжети, жанри, а завдяки компаніям Unity Technologies та Epic Games, які створили та випустили в загальний доступ свої кросплатформенні середовища розробки ігор, або як їх називають «ігрові двигуни», з’явилося багато інді-розробників, тобто спеціалістів які розроблюють ігри самостійно або з невеликою командою, які почали створювати свої проєкти, тим самим розвиваючи ігрову індустрію.

На сьогоднішній день існує більш ніж мільйон відеоігор та понад 30 різних жанрів. Майже в кожній країні світу є своя компанія яка створює та випускає відеоігри. Україна також має свої компанії. Найвідоміша з них – 4A Games яка створила три гри серії Metro. На цьому історія ігор не закінчується, а тільки починається, бо як казав персонаж відомої гри «The Legend of Zelda»: “Пригода триває. Сама подорож має сенс”.

1 ОСНОВНІ ПОЛОЖЕННЯ ТА ВИЗНАЧЕННЯ

1.1 Ігрові двигуни та їх особливості

У світі існує дуже багато різних ігрових двигунів, які були розроблені різними компаніями у різні роки. Зазвичай вони діляться на два типи:

– комерційні двигуни;

– двигуни у вільному доступі.

Якщо говорити про комерційні двигуни, то це двигуни, які були розроблені компанією для власного використання. Вони не існують у вільному доступі та навряд чи існуватимуть. Майже кожна велика компанія яка займає ключове місце у світі розробки ігор має свій власний двигун для розробки. Це пов’язано з метою заощадження грошей та позбавлення зайвих витрат. Кожен ігровий двигун, який знаходиться у вільному доступі, має ліцензійну угоду, підписуючи яку, інді-розробник або компанія погоджується з правилами використання двигуна. Зазвичай у такій угоді обов’язково прописані пункти про комерційну діяльність у яких вказано позиції стосовно комерційного використання двигуна для розробки ігор. Варіанти комерційного використання двигунів від різних компаній дуже відрізняються один від одного. Деякі компанії вказують виплату за використання двигуна, якщо кількість проданих копій гри перевищує певну кількість людей. Інші роблять залежність від кількості прибутку за певний проміжок часу. Варіантів сплати за використання комерційного двигуна багато, але їх об’єднує одне – що за них потрібно платити. Саме тому компанії використовують ігрові двигуни власної розробки.

Окрім заощадження грошей за використання власного двигуна, є ще один досить суттєвий плюс який нівелює усі гарні сторони двигунів у вільному доступі. Оскільки компанії не планують створювати двигун для широкого використання, вони розробляють його з певним функціоналом,

інструментами та налаштуваннями, які підійдуть тільки для певної гри. Наприклад компанія «Valve» має свій ігровий двигун під назвою «Source 2» який вони використовують для розробки та оновлень своїх ігор [1]. На цьому двигуні була створена дуже популярна гра «Counter-Strike 2» у яку, по статистиці інтернет-магазину «Steam», кожного дня грає понад п’ятсот тисяч гравців.

Ще одним прикладом комерційного ігрового двигуна є «RAGE», розроблений компанією «Rockstar Games» [2]. Ця компанія вважається «титаном» ігрової індустрії. Вона створила такі ігри, як: «Grand Theft Auto V», «Max Payne 3», «Red Dead Redemption 2». Усі ці ігри були розроблені на двигуні «RAGE». Вони стали прикладом того, як компанія переймається за якість свого продукту та постійно вдосконалює свій ігровий двигун.

Ігрові двигуни, які є у вільному доступі, також користуються великою популярністю. Найпопулярнішими з них є – «Unity», «Unreal Engine», «Godot», «Construct». Кожен з них має свої особливості, плюси та мінуси, але найбільший бонус цих ігрових двигунів – до них є документація та багато курсів, тому навіть новачок зможе створити свою гру розібравшись з основами. Завдяки цим дивовижним двигунам були створені такі проєкти, як: «Gears of War», «We Happy Few», «Cities Skylines», «Cuphead», «Kerbal Space Program». Це лише частина від великого переліку ігор які були створені саме на цих ігрових двигунах.

Короткий опис кожного з перерахованих двигунів.

«Unity» – це один з найпопулярніших ігрових двигунів, який до вподоби великим компаніям. Але на сьогоднішній день він дуже сильно припав до душі саме інді-розробникам через його простоту та можливості які він пропонує. Він підтримує широкий спектр різних платформ, таких як ПК, консолі, мобільні пристрої та VR-гарнітура. Окрім вбудованих інструментів, які ідуть разом з двигуном, до нього існує багато різних доповнень, які можна встановити не виходячи з самого Unity, що робить його майже автономним та дає можливість не відвертати увагу від створення проєкту [3].

«Unreal Engine» – станом на сьогодні найновіша версія цього двигуна була розроблена у 2022 році компанією «Epic Games» та має назву «Unreal Engine 5». Цей двигун дивує своїми потужними графічними можливостями. В першу чергу «Unreal Engine» звісно використовується для створення ігор, а саме ААА проєктів. Але окрім його основного призначення, двигун також використовують у кіноіндустрії. Саме на «Unreal Engine 4» переважно створювались задні фони для фільмів студії Marvel, в нових «Зоряних війнах» та у серіалі «Мандалорець» [4]. Сам двигун має можливість створювати як 2D, так і 3D проєкти, але все ж таки він більше налаштований для 3D.

«Godot» – це двигун з відкритим вихідним кодом, що робить його абсолютно безкоштовним для використання. «Godot» з'явився досить нещодавно у 2014 році, коли досяг своєї першої стабільної версії. Сутність двигуна досить проста – бути якомога простішим та самодостатнім середовищем розробки ігор. Саме тому він вважається одним з найлегших для вивчення двигунів розробки ігор. Його викладають дітям у школах та у комп'ютерних академіях аби зацікавити їх та залучити до IT сфери. Двигун підтримує 2D та 3D розробку ігор та має зрозумілу систему скриптів. Спільнота «Godot» активно допомагає розробникам розвивати двигун, тому він має регулярні оновлення [5].

«Construct» – це середовище розробки ігор, яке спеціалізується на створенні 2D-ігор. Наразі останньою версією вважається «Construct 3» який можна завантажити на комп'ютер або відкрити як вебдодаток у браузері. Його інтуїтивно зрозумілий інтерфейс дозволяє створювати ігри за допомогою візуального програмування, що робить його ідеальним варіантом для новачків, які тільки знайомлять зі світом розробки ігор [6].

Такі відомі ігри як Hollow Knight (створена на Unity), Fortnite (створена на Unreal Engine) та Sonic Colors: Ultimate (створена на Godot) також демонструють різноманітність і потенціал цих ігрових двигунів.

1.2 Жанри ігор

Поняття жанру з’явилося ще у ті часи, коли люди тільки почали займатися мистецтвом. Це було пов’язано з узагальненням певних типів робіт за певними подібностями, які їх об’єднували. Наприклад, жанри літератури були розділені за поетичною структурою на лірику, драму та епос, до яких входили ліричні вірші, пісні, казки, байки, легенди, комедії, драми, оповідання, повісті, романи тощо. Таким чином люди створили щось на кшталт сортування, яке допомагало орієнтуватися у виборі незліченної кількості творів. Така сама ситуація відбувається у всіх видах мистецтва починаючи з музикального та художнього мистецтва, закінчуючи кінематографією, скульптурним мистецтвом та архітектурою. Люди завжди намагаються все структурувати, зробити певну ієрархію аби полегшити розуміння складних систем і взагалі розуміння навколишнього світу.

Ігри, як і види мистецтва, також мають свої види, підвиди та жанри. Жанри ігор – це категорії, які використовуються для класифікації відеоігор за їх стилем, ігровою механікою та іншими характеристиками. Використовуючи пошук за жанрами гравці мають можливість легко знайти ігри, які відповідають їхнім культурним, релігійним та духовним уподобанням. На сьогоднішній день одна гра може поєднувати елементи різних жанрів, створюючи таке поняття, як гібридні жанри. Наприклад, гра може поєднувати елементи екшену, RPG, і стратегій, створюючи унікальний ігровий досвід. Загалом ігри діляться на такі жанри [7]:

- екшен (Action);
- пригодницькі ігри (Adventure);
- рольові ігри (RPG – Role-Playing Games);
- симулятори (Simulation);
- стратегії (Strategy);
- спортивні ігри (Sports);

– гоночні ігри (Racing);
– ігри жахів (Horror);
– пазли (Puzzle);
– бойові ігри (Fighting).

Жанр «Екшен» є одним із найпопулярніших і найрізноманітніших жанрів у відеоіграх. Ігри цього жанру відрізняються високою динамічністю, акцентом на рефлекси, координацію та швидке прийняття рішень [7]. У більшості ігор цього жанру значна частина ігрового процесу присвячена бою. Окрім бою важливим аспектом цього жанру є рівні та перешкоди які гравці повинні пройти, щоб досягти певних нагород. Хоча акцент більше сфокусований на діях, багато ігор мають глибокий сюжет і розвинених персонажів. Найкращим доповненням до цього жанру є мультиплеєр який додає багатокористувацький режим, що дає можливість гравцям змагатися або співпрацювати з іншими гравцями. Цей жанр поділяється на декілька піджанрів.

Шутери (Shooter) – ігри, в яких головна механіка полягає в стрільбі по ворогах за допомогою різних видів зброї. У іграх цього піджанру зустрічаються два види постановки камери. Вид від першої особи (FPS) коли Гравець бачить світ очима персонажа, що створює почуття повного занурення.

Платформери (Platformers) – ігри, де гравець переміщується за рівнями, стрибаючи по платформах та уникаючи перешкод [7]. В свою чергу жанр розділяється на класичні платформери, в яких гравець стрибає по платформам, збирає предмети і змагається з ворогами, та метроїдванію, яка характеризується відкритим світом, який досліджує гравець, отримуючи нові можливості для доступу до нових зон.

Сурвайвал-екшен (Survival Action) – ігри, де гравець повинен вижити в умовах обмежених ресурсів, часто зіткнувшись із ворогами чи небезпечним середовищем. Вони характеризуються виживанням, збиранням ресурсів та

управлінням інвентарем.

Рольові ігри – це подорож у світ фантазій та пригод, де гравець стає не просто спостерігачем, а активним учасником захопливих подій. Цей жанр відкриває нескінченні можливості для самовираження, взаємодії з іншими персонажами та впливу на події у вигаданому світі [7]. Наведемо основні елементи рольової гри.

Створення персонажа. Першим кроком, для деяких гравців найголовнішим кроком, є створення унікального героя за якого гравець буде відігравати, обрати расу, клас, властивості та початкові навички. Ця механіка додає до гри різноманіття та можливість максимально кастомізувати персонажа, а також обрати певну стратегію гри.

Розвиток персонажа. Протягом гри персонаж отримуватиме досвід, покращуватиме характеристики та підвищуватиме рівень володіння різними навичками. Це дозволяє адаптувати свого героя до різних викликів та складнощів гри.

Відкритий світ. Рольові ігри зазвичай пропонують великі відкриті світи для дослідження. Вони наповнені таємницями, завданнями та можливостями взаємодії з різними персонажами та фракціями.

Стратегічні ігри – жанр, що вимагає від гравця вміння планувати, аналізувати ситуацію та приймати обдумані рішення [7]. Цей жанр відкриває гравцеві можливість керувати арміями, розвивати цивілізації та створювати економічні імперії, де кожен крок може призвести до тріумфу чи поразки. Ігри цього жанру користуються великою популярністю завдяки своїй глибині та можливостям для інтелектуальних викликів. Вони залучають гравців, які цінують складні та багатогранні завдання, що вимагають обдуманих рішень та стратегічного мислення. Це справжнє мистецтво планування та завоювань, де кожен хід може змінити хід історії. Вони дозволяють гравцям поринути у світ великих імперій, грандіозних битв та дипломатичних інтриг. Наведемо основні піджанри стратегічних ігор.

Покрокові стратегії. Це піджанр стратегій, де гравці по черзі виконують дії, що дозволяють детально планувати кожен крок.

Стратегії в реальному часі. Гравці діють одночасно, що вимагає швидких рішень та вміння обмірковувати та аналізувати декілька завдань одночасно. Основні елементи включають будівництво баз, збір ресурсів та командування арміями в реальному часі.

Глобальні стратегії. Ігри, що охоплюють великі часові періоди та беруть глобальні масштаби, дозволяючи гравцям керувати цілими націями чи імперіями. Основними аспектами ігор такого плану є дипломатія, економіка, військова кампанія та керування державою.

Ігри жахів – це жанр, який занурює гравця в атмосферу напруження, страху та непередбачуваності. Вони майстерно поєднують елементи психологічного тиску, моторошної атмосфери та захопливих сюжетів, створюючи унікальний досвід, що тримає у напрузі до самого кінця. Наведемо піджанри ігор жахів.

Психологічний хоррор. Жанр фокусується на психологічному страху та напрузі. Гравці поринають у страшні події де зустрічаються зі страшними образами, які грають на їхніх глибоких страхах.

Екшен-хоррор. Цей піджанр поєднує елементи жахів з інтенсивним бойовим геймплеєм. Гравці мають більше можливостей для захисту, але все одно стикаються з численними загрозами. Вони відрізняються від інших піджанрів можливістю дати відсіч ворогам та наявністю інвентарю.

Хоррор-квест. Гравці повинні розкривати таємниці, досліджувати світ та уникати небезпек які ховаються на кожному розі. Основний концепт ігор розслідування злочинів, головоломки або пошук давніх таємничих артефактів з привидами та монстрами.

Сурвайвал-хоррор. У цьому піджанрі гравці повинні вижити за умов постійної загрози. Обмежені ресурси та сильні противники створюють почуття безпорадності. Головна задача таких ігор – вижити.

1.3 Інструменти для створення ігор

Процес створення ігор займає дуже багато часу, потребує чимало грошей та залучає немало людей. Але тут не вистачає ще однієї досить вагомої деталі – інструменти для створення ігор. Під інструментами мається на увазі усі програми без яких неможливо створити гарну гру, але якщо використовуються сучасні інструменти для розробки, кінцевий результат може виявитися дивовижним. На сьогоднішній день існує безліч платформ та програм, які значно спрощують процес створення ігор. Звичайно головним інструментом при розробці вважається ігровий двигун, але якщо до нього нема чого додавати, який у ньому сенс. Тому обов’язковими інструментами є інструменти для створення графіки. Створення візуальних елементів є ключовим аспектом розробки відеоігор. Для цього використовуються різноманітні графічні редактори та програми для 3D-моделювання. Ці інструменти забезпечують розробників необхідними засобами створення високоякісних зображень і моделей, та є основою для візуальної складової гри. Для різних типів ігор, 2D або 3D, зазвичай використовуються різні інструменти.

1.3.1 Інструменти створення графіки

Adobe Photoshop є одним з найпотужніших і найпоширеніших інструментів для створення та редагування 2D-графіки. Його функціональні можливості включають широкий спектр інструментів для малювання, редагування, ретушування і створення композицій. Photoshop підтримує роботу із шарами, що дозволяє розробникам зберігати складні проєкти у впорядкованому вигляді, а також створювати 2D анімації для різних персонажів, об’єктів та предметів. Завдяки своїм розширеним можливостям Photoshop часто використовується для створення текстур, концепт-артів та

інтерфейсів користувача [8]. Його багатофункціональність, гнучкість та інтеграція з іншими продуктами робить його найважливішим компонентом в арсеналі кожного розробника ігор. Завдяки постійному оновленню та вдосконаленню Photoshop залишається актуальним вибором для професійного використання у різних галузях, включаючи розробку відеоігор.

GIMP (GNU Image Manipulation Program) є безкоштовним та відкритим аналогом Adobe Photoshop. Він відіграє важливу роль у створенні різних графічних елементів, таких як текстури, спрайти, інтерфейси користувача і концепт-арт. Ця програма надає більшість функціональних можливостей, доступних у Photoshop, включаючи інструменти для малювання, редагування та роботи з шарами [9]. GIMP є популярним вибором серед незалежних розробників та невеликих студій завдяки своїй доступності та розширюваності за допомогою численних плагінів. Завдяки постійному розвитку та підтримці спільноти GIMP залишається актуальним інструментом професійного використання у створенні візуальних елементів для відеоігор.

Blender є безкоштовною програмою для 3D-моделювання, анімації та рендерингу. Вона надає повний набір інструментів для створення тривимірних моделей, текстуризації, скульптингу, анімації та симуляції фізики. Blender також підтримує відеомонтаж та створення інтерактивних 3D-додатків. Завдяки своїй багатофункціональності та відкритому коду, Blender використовується як незалежними розробниками, так і професійними студіями для створення як окремих елементів, так і повноцінних тривимірних сцен. Він підтримує багато форматів файлів, таких як FBX, OBJ, STL та інших, що полегшує обмін даними з іншими програмами. Також він має підтримку безкоштовних аддонів і скриптів на Python, що дозволяє розширювати функціонал Blender [10]. Тому кожен, хто має навички у програмуванні на мові програмування Python, може створити свій власний скрипт для Blender, та використовувати його у своїх проєктах.

Maya та 3ds Max, розроблені компанією Autodesk, є двома з найпотужніших та найпоширеніших програм для 3D-моделювання, анімації та рендерингу. Вони використовуються в різних галузях, включаючи кіноіндустрію, архітектуру, візуалізацію та, зокрема, розробку відеоігор.

Maya пропонує розширені інструменти для полігонального, NURBS та subdivision моделювання, що дозволяє створювати високодеталізовані та складні моделі. Інструменти для скульптингу та ретопології забезпечують додаткову пружність у розробці органічних форм [11]. 3ds Max має потужні інструменти для полігонального та сплайнового моделювання, що дозволяє створювати як прості, так і високодеталізовані моделі. Інструменти для процедурного моделювання та модифікатори значно спрощують процес створення складних геометричних форм. Обидві програми використовуються для рендерингу високоякісних зображень та анімацій, які можуть використовуватись для створення трейлерів, кінематографічних вставок та маркетингових матеріалів. 3ds Max підтримує мови програмування MaxScript та Python, що дозволяє користувачам автоматизувати рутинні завдання, створювати власні інструменти та розширювати функціональність програми [12]. Інструменти для симуляції фізики та процедурного моделювання дозволяють створювати реалістичні ефекти, такі як вибухи, руйнування, рідини та тканини. Завдяки своїй популярності, Maya та 3ds Max мають велику спільноту користувачів та безліч навчальних матеріалів.

Висновок. Ці редактори є незамінними інструментами для професійного 2D та 3D-моделювання та анімації. Завдяки своїм потужним можливостям, гнучкості та інтеграції з іншими програмами вони забезпечують високий рівень продуктивності та якості у процесі створення відеоігор. А наявність підтримки мов програмування дозволяє автоматизувати рутинні завдання. На сьогоднішній день вони залишаються популярними серед розробників ігор, художників та аніматорів завдяки своїм широким можливостям та підтримці промислових стандартів.

1.3.2 Інструменти для створення звуків і музики

Звуковий супровід є критично важливим елементом не тільки в ігровій індустрії, але й у кіноіндустрії та театральному мистецтві. Він допомагає людині з головою зануритись у той світ, той вимір, який представлений у фільмі або грі, надаючи глибину, атмосферу та емоційне забарвлення глядацького досвіду. Люди переживають емоції головного героя та емоції самої сцени. Для створення звуків та музики використовуються різні програмні та апаратні інструменти, що забезпечують високу якість та професійний рівень аудіовиробництва. Окрім програмних та апаратних інструментів також використовують звичайні предмети з повсякденного життя. Наприклад, для озвучування грози у фільмах та іграх використовують звичайний металевий лист, який просто трусять зі сторони в сторону, тим самим створюючи звукові коливання, які нагадують шум грози або грому. З цього можна зробити висновок, що використовуючи будь-який предмет, який лежить під рукою, можна створити звуковий супровід, перед цим обробивши його в спеціальних програмних інструментах.

Одним з таких інструментів є Ableton Live. Ableton Live – це цифрова аудіо робоча станція (DAW), яка стала дуже популярною серед музикантів, продюсерів та звукових дизайнерів завдяки своїй універсальності, інтуїтивно-зрозумілому інтерфейсу та потужним можливостям. Вперше випущена в 2001 році Ableton Live відрізняється своєю особливістю для живого виступу, а також студійного використання. Програма підтримує створення, запис, редагування та виконання музики в реальному часі, що робить її ідеальною як для студійної роботи, так і для живих виступів. Ableton Live має два основні режими: Arrangement View та Session View [13]. Arrangement View дозволяє користувачам працювати з традиційною лінійною шкалою часу, у той час як Session View надає мережевий інтерфейс для запуску кліпів (музичних фрагментів) у реальному часі, що є особливо корисним для живих виступів та імпровізації. Ще одним великим плюсом

програми є великий набір вбудованих віртуальних інструментів, таких як Analog, Operator, Wavetable, Simpler, Sampler та Drum Rack. Ці інструменти дозволяють створювати широкий спектр звуків від синтетичних до акустичних. Програма надає багатий вибір аудіоефектів, таких як реверберація, еквалізація, компресія, затримка та багато інших. Ефекти можуть застосовуватись у режимі реального часу, що дозволяє змінювати звук під час живих виступів чи запису. Використовуючи такий великий набір інструментів композитори мають змогу створювати саундтреки, які можна легко редагувати та аранжувати завдяки гнучкості програми, звукові дизайнери створюють звукові ефекти в реальному часі, паралельно використовуючи вбудовані ефекти та сторонні плагіни, які можуть реагувати на дії гравців або зміни в ігровому середовищі.

FL Studio, також відомий як Fruity Loops, є однією з найпопулярніших цифрових аудіо робочих станцій (DAW) для створення, запису, редагування та мікшування музики та звукових ефектів. Вперше випущена в 1997 році компанією Image-Line, FL Studio набула популярності завдяки своєму інтуїтивному інтерфейсу, великому набору вбудованих інструментів і ефектів та гнучкості у використанні [14]. Програма підходить як для професійних музикантів та продюсерів, так і для початківців. Інтерфейс FL Studio має зрозумілий та інтерактивний інтерфейс з кількома основними вікнами, включаючи Playlist, Piano Roll, Channel Rack та Mixer. Користувачі можуть легко організовувати свої проєкти, маючи звукові кліпи, MIDI-дані та ефекти зрозумілим та логічним способом. Piano Roll у FL Studio є одним із найпотужніших інструментів для роботи з MIDI. Він дозволяє користувачам точно редагувати ноти, налаштовувати їхню тривалість, висоту та динаміку, а також застосовувати різні MIDI ефекти. Mixer дозволяє користувачам мікшувати та обробляти звук з використанням різних ефектів, таких як еквалізація, реверберація, компресія та багато інших. Кожен канал Mixer може мати власний набір ефектів та налаштувань. Гнучкість FL Studio дозволяє швидко створювати та тестувати музичні ідеї, що важливо для

процесу розробки ігор, де часто потрібна швидка адаптація та внесення змін до аудіо контенту.

Wwise (WaveWorks Interactive Sound Engine) є вбудованою платформою для аудіодизайну, розробленою компанією Audiokinetic. Цей інструмент надає розробникам відеоігор потужні засоби створення, керування та імплементації аудіо в ігрове середовище [15]. Запущений у 2006 році Wwise став стандартом для реалізації звукових ефектів, музики та атмосферного звуку в ігровій індустрії. Wwise дозволяє розробникам легко інтегрувати аудіо в ігровий двигун, надаючи гнучкі засоби для створення акустичної атмосфери, музики та звукових ефектів. Платформа підтримує реалістичне 3D просторове звучання, що дозволяє створювати звукові ефекти, що відтворюються відповідно до розташування гравця в ігровому світі. Wwise пропонує механізм динамічного перехресного змішування, що автоматично відрегулює гучність та інтенсивність звукових ефектів залежно від дій гравця та ігрової ситуації. Програма підтримує інтеграцію з багатьма популярними ігровими двигунами, такими як Unity, Unreal Engine, CryEngine та іншими, а керування пам’яттю дозволяє ефективно використовувати пам’ять аудіо ресурсів, що важливо для оптимізації продуктивності гри.

Для таких великих програмних інструментів, також існують допоміжні платформи. Наприклад платформа Serum. Serum – найпопулярніший віртуальний синтезатор, який надає користувачам потужні інструменти звукового дизайну, включаючи можливість редагування хвильових форм.

Висновок. Ці платформи є одними з найпопулярніших інструментів для створення музики та звукових ефектів завдяки своєму інтуїтивно-зрозумілому інтерфейсу, потужними можливостями та гнучкості у використанні. У контексті розробки відеоігор вони надають композиторам та звуковим дизайнерам всі необхідні інструменти для створення високоякісного аудіо контенту, що значно збагачує ігровий досвід. Завдяки своїм широким можливостям, платформи залишаються незамінними інструментами в кишені професійних музикантів та продюсерів.

1.3.3 Інструменти для створення сценаріїв та діалогів

Створення якісних сценаріїв та діалогів є важливою складовою процесу розробки відеоігор, оскільки вони визначають сюжетну лінію, персонажів та динаміку гри. Інструменти для створення сценаріїв та діалогів надають розробникам можливість створювати захопливі та емоційно насичені ігрові світи, які залучають гравців. Розглянемо основні інструменти для створення сценаріїв.

Текстові редактори. Базовий інструмент для написання сценаріїв та діалогів – текстовий редактор. Він дозволяє розробникам створювати та редагувати текстові файли, що містять діалоги персонажів, опис сцен та інші елементи сюжету.

Спеціалізовані платформи для сценарного планування. Деякі інструменти надають спеціалізовані середовища для створення сценаріїв та сценарного планування. Вони дозволяють розробникам створювати структуровані сценарії, включаючи персонажів, діалоги, дії та сцени.

Візуальні редактори сценаріїв. Деякі редактори надають можливість візуалізувати створення сценаріїв. Найчастіше в них присутні інструменти які дозволяють розробникам візуально організовувати сцени, персонажів та діалоги за допомогою блоків або схем.

Інструменти для діалогового дерева. Ці інструменти дозволяють розробникам створювати діалогові дерева, що є візуальними структурами, які визначають логіку та варіанти діалогів між персонажами.

Мовні системи. Деякі інструменти мають мовні системи, які дозволяють розробникам створювати діалоги та сценарії різними мовами, що розширює аудиторію гри.

Для створення та реалізації сценаріїв використовується інструмент під назвою «Twine». Twine є безкоштовним інструментом для створення текстових ігор та інтерактивних історій, що дозволяє авторам створювати складні інтерактивні навігаційні структури без необхідності програмування.

Розроблений Грейс Лі та Денієлом Шифманом, Twine став популярним серед письменників, геймдизайнерів та художників, які шукають засіб для створення історій зі складною гілковою структурою [16]. Twine надає візуальний редактор, що дозволяє творцям створювати гілкові структури інтерактивних історій за допомогою перетягування та випадаючих меню. Він підтримує можливість створення інтерактивних історій, які можуть бути спільною творчістю кількох авторів, що дозволяє колективно розробляти та вдосконалювати історії. Хоча Twine не вимагає знань програмування для базового використання, він підтримує скрипти та макроси для створення більш складних інтерактивних елементів та логіки гри. Розробники можуть використовувати HTML, CSS та JavaScript для кастомізації та додавання додаткових функцій. Twine підтримує інтеграцію з іншими інструментами та платформами, такими як Unity та Unreal Engine, що дозволяє його використовувати як частину більшого ігрового проєкту. Це дозволяє розробникам створювати текстові сегменти та діалоги в Twine і потім інтегрувати їх у більш складні ігрові механіки. Завдяки швидкості та простоті використання, Twine є чудовим інструментом для прототипування наративних ідей та механік. Розробники можуть швидко створювати та тестувати різні варіанти сюжету та взаємодії.

Висновок. Twine є потужним інструментом для розробки інтерактивних історій та текстових ігор, що дає розробникам можливість створювати захопливі та складні сюжети без потреби у глибоких технічних знаннях. Його простота, гнучкість та можливості кастомізації роблять його важливим інструментом у сучасній ігровій індустрії.

Ще одним професійним інструментом для створення інтерактивних історій та сюжетних ліній є «Articy: Draft». Articy: Draft є потужним інструментом для розробки інтерактивного контенту, створеного компанією Nevigo. Цей інструмент широко використовується в ігровій індустрії для створення сценаріїв, діалогів, квестів та нелінійних сюжетів, надаючи розробникам можливість організовувати та керувати складними структурами.

Він пропонує візуальний інтерфейс для створення та редагування сценаріїв [17]. Розробники можуть створювати гілкові структури, організовувати сцени, діалоги та інші елементи за допомогою зручних блоків та діаграм. Також інструмент дозволяє створювати та керувати базами даних персонажів, об’єктів, локацій та інших елементів гри. Це допомагає розробникам відстежувати зв’язки між різними елементами гри та гарантувати, що вони узгоджені. Articy: Draft пропонує інтеграцію з популярними ігровими двигунами, такими як Unity та Unreal Engine. Це дозволяє експортувати створений контент прямо в ігровий проєкт, спрощуючи процес реалізації сюжету. Співпраця в реальному часі дозволяє кільком користувачам одночасно працювати над одним проєктом. Це особливо корисно для великих студій, де різні члени команди можуть працювати над різними аспектами сюжетних ліній, а система автоматичного збереження допомагає розробникам відстежувати зміни в проєкті та повертатися до попередніх версій у разі потреби. Плагіни та розширення які підтримує інструмент, надають можливість додавати нові функції та інтегрувати Articy: Draft з іншими інструментами та платформами, підвищуючи його гнучкість та функціональність.

1.4 Висновки до розділу 1

Розробка відеоігор є складним і багатогранним процесом, який вимагає використання різноманітних інструментів і технологій. Від вибору ігрового двигуна та жанру до використання спеціалізованих інструментів для створення графіки, звуків і сценаріїв. Розуміння специфіки ігрових двигунів, вимог різних жанрів і можливостей інструментів допомагає розробникам створювати якісні, захопливі та конкурентоспроможні ігри. Успішна інтеграція всіх цих аспектів є ключем до створення успішного ігрового проєкту, який знайде своє місце на ринку та завоює прихильність гравців.

2 ПРОЄКТУВАННЯ ГРИ

Проєктування гри складається з кількох основних етапів, кожен з яких є важливим для створення успішного та якісного продукту. Від самого етапу проєктування залежить хід всього проєкту, тому розробники приділяють йому значну кількість часу, розмірковуючи над кожною деталлю. Саме проєктування може тривати від декількох місяців, до року в залежності від складності та масштабу проєкту, розміру та досвіду команди, фінансування та використовуваних технологій. Коли гру спроєктовано, команда переходить до наступного етапу – розробки гри.

2.1 Концепція та ідея гри

Створення концепції та ідеї гри вважається основним етапом при проєктуванні гри. Це досить важкий етап у якому дуже важливо правильно сформулювати мету та тему проєкту, тому що саме від теми залежить те, як буде виглядати кінцевий продукт. Наступним кроком є створення короткого опису гри, у якому відображаються основні сюжетні лінії та механіки гри, які можуть бути розроблені на етапі реалізації.

Назва гри: «Лицарські хроніки».

Сетинг гри: Середньовічний фентезійний світ у 2D.

Тема гри: «Лицарські хроніки» – це захоплива рольова гра, в якій гравці постають у ролі молодого лицаря, який мріє стати легендою свого королівства. Гравець повинен подолати безліч випробувань, битися з численною кількістю ворогів, розгадувати стародавні таємниці та приймати важливі рішення, які вплинуть на долю світу. Джерелом натхнення для створення цієї гри були: захоплива гра з відкритим світом «Kingdom Come: Deliverance» та короткометражні відеоролики «Epic NPC Man» з YouTube

каналу «Viva La Dirt League». Ці джерела вдихнули життя в саму ідею створення гри та стали основою для реалізації проєкту.

Жанр гри належить до рольової гри (RPG) та позиціонує себе як пригодницький екшен. Враховуючи ці жанри можна зробити висновок, що гравець не просто спостерігає за перебігом подій у грі, він безпосередньо бере участь у захопливих подіях, досліджуючи світ та вирішуючи його долю. Основний концепт полягає саме у пригодах, аби зацікавити гравця та залучити його до гри.

Основна історія сюжету починається з того, що головний герой, сирота, який після знищення рідного селища виховується в монастирі. Дізнавшись про своє походження та справжню мету, він вирушає в подорож, щоб виконати своє покликання – стати лицарем і врятувати королівство від загрози, що насувається. По дорозі герой зустрічає союзників і ворогів, вивчає мистецтво бою і магії та розкриває таємниці своєї родини.

Основні механіки гри.

Система бою. Комбінація бою з мечами, сокирами, списами, луками, арбалетами та іншими видами зброї середньовіччя.

Використання магії. Вивчення магічних заклинань і комбінування їх з фізичними атаками.

Еволюція головного героя. Накопичення досвіду та розподіл очок рівнів для прокачування навичок та відкриття нових здібностей. Можливість вибору шляху розвитку персонажа за допомогою вибору класу, наприклад могутній воїн, чарівник, стрілець, некромант, грабіжник тощо.

Напіввідкритий світ. Наповнення світу численною кількістю локацій таких як міста, селища, лісові та горні масиви, підземелля та замки. Можливість вибору маршруту персонажа на карті світу.

Сюжетні рішення. Рішення, які приймає гравець, впливають на розвиток сюжету та стосунки з іншими персонажами.

Квести та завдання. Наявність основних сюжетних завдань, що розкривають головну сюжетну лінію та побічні квести, які дають змогу

накопичити рівень для прокачування персонажа, здобути легендарну зброю, або дізнатися більше про персонажів другого плану та лор гри.

2.2 Вибір технологій для реалізації проєкту

Вибір технологій для розробки ігор є надзвичайно важливим етапом, оскільки він впливає на різні аспекти проєкту. Це рішення впливає на продуктивність, якість, вартість, час розробки, масштабованість і майбутнє розробки ігор. Ретельний аналіз і врахування всіх цих факторів допоможе зробити правильний вибір і забезпечить успішну реалізацію проєкту.

Продуктивність та якість гри впливають на те, наскільки плавно та швидко гра працюватиме на різних пристроях. Це охоплює в себе вибір двигуна (наприклад, Unity, Unreal Engine) і мови програмування (наприклад, C++, C#). Технології для реалізації графіки та ефектів визначають графічні можливості, ефекти, фізику, анімацію та інші візуальні аспекти гри.

Важливо враховувати, на яких платформах буде доступна гра: ПК, консолі, мобільні пристрої, VR. Деякі двигуни та інструменти пропонують вбудовану підтримку кросплатформної розробки. Одним з таких ігрових двигунів є «Unity».

Вибір технологій безпосередньо впливає на вартість розробки, включаючи ліцензії на програмне забезпечення, засоби розробки та ресурси, необхідні для вивчення та впровадження цих технологій.

Час розробки проєкту залежить від знань та досвіду команди. Якщо технології нові або занадто складні, це може вплинути на ефективність роботи та збільшити час, необхідний на розробку гри. Але за наявності активної спільноти та гарної документації можуть допомогти швидше вирішити проблеми і знайти відповіді на запитання.

Розширення та оновлення гри відіграють ключову роль після релізу гри, тому обрана технологія повинна мати можливість для реалізації додаткових функцій та контенту.

Роблячи висновок з вимог, які були поставлені для реалізації проєкту, було обрано ігровий двигун «Unity», тому що саме він відповідає усім критеріям та вимогам.

«Unity» – є потужним, гнучким і доступним інструментом для розробки ігор, що робить його популярним вибором для розробників будь-якого рівня досвіду [3]. Поріг входження до вивчення «Unity» є значно нижчим ніж у такого самого потужного двигуна як «Unreal Engine». Кожен новачок, який має хоча б основну базу знань та тільки починає свій шлях у світ розробки ігор, може з легкістю почати вивчати «Unity». Завдяки широкій спільноті та підтримці розробників він має офіційну документацію, велику кількість навчальних матеріалів, форумів, курсів та відеоуроків. Це робить пошук відповідей на складні запитання значно простішим.

Завдяки тому, що «Unity» підтримує велику кількість платформ, включаючи Windows, macOS, Linux, iOS, Android, WebGL, PlayStation, Xbox, Nintendo Switch тощо, це дозволяє розробникам створити гру один раз, а потім легко експортувати її на різні платформи [3].

Для розробників, які не мають художнього мислення та не вміють малювати, але хочуть, щоб їх гра виглядала досить пристойно з погляду графіки та візуальних ефектів, «Unity» має свій магазин «Unity Asset Store» який пропонує велику кількість готових до використання моделей, текстур, скриптів та інших ресурсів, які можна легко інтегрувати в проєкти. Це дозволяє розробникам економити час і ресурси. Це справжнє відкриття для новачків, тому що багато з них не розуміють як саме повинна виглядати локація ігрового світу, та взагалі як створювати спрайти або моделі для гри. На цьому етапі їх шлях як розробника ігор може закінчитись, але з «Unity Asset Store» ця проблема вирішується декількома кліками.

Оскільки «Unity» має потужні інструменти для створення 2D ігор, то за допомогою них виконується основна задача сетингу гри, а саме світ у 2D вимірі. Ігри такого плану мають всього дві координати за якими переміщується персонаж.

C# – це сучасна об’єктно-орієнтована мова програмування, розроблена компанією Microsoft як частина платформи .NET. Вона була створена у 2002 році і з того часу стала однією з найпопулярніших мов програмування в світі [18]. Саме ця мова програмування використовується як основний інструмент для створення складних ігрових механік та систем у двигуні «Unity».

2.3 Проєктування основних алгоритмів гри за допомогою діаграм

Алгоритми являються ключовою складовою кожного проєкту. Вони використовуються абсолютно у кожній сфері роботи та в повсякденному житті.

Одним з найвідоміших алгоритмів є алгоритм приготування чаю. Він складається з дуже простих кроків, які можуть повторити навіть діти. Але це лише один з видів алгоритмів. Насправді існує два види алгоритмів:

– суворий алгоритм;

– не суворий алгоритм.

Алгоритм приготування чаю є не суворим, оскільки якщо взяти сто людей та запропонувати їм заварити чай, кожен з них зробить це по-різному. Хтось додасть холодної води, хтось покладе скибку лимона тощо. Тому цей алгоритм вважається не точним.

Сенс алгоритму полягає у суворому дотриманні всіх правил для виконання задачі. Це певний список, який описує порядок дій одна за одною. Програма яка слідує одному алгоритму повинна видавати один і той самий результат.

При проєктуванні проєкту з’явилась необхідність у створенні алгоритму дій гравця, які він буде виконувати одразу після відкриття гри, для аналізу, пошуку та виправлення всіх розбіжностей та додавання відсутніх кнопок взаємодії. На рисунку нижче (див. рис. 1) продемонстровано спроєктований алгоритм взаємодії гравця з грою.

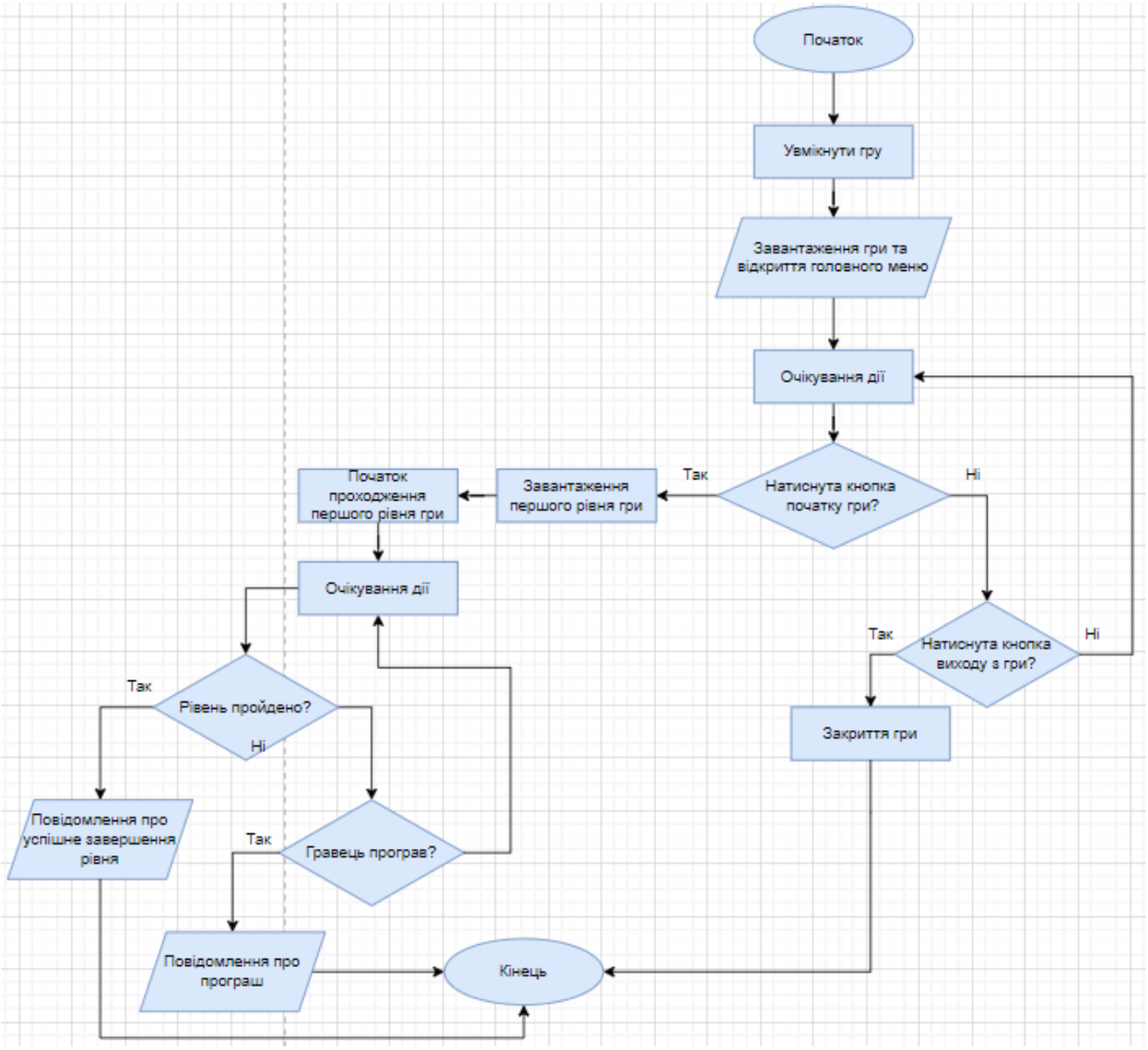

Рисунок 1 – Алгоритм взаємодії гравця з грою

Алгоритм взаємодії гравця з грою демонструє кожен крок гравця, який він може зробити. Після увімкнення гри перед гравцем з’являється головне меню з вибором, почати проходження рівня, або вийти з гри. За це відповідають дві кнопки: «Start» та «Exit». Кнопка «Start» відповідає за початок гри, а кнопка «Exit» відповідно за вихід з гри. Після початку гри гравець починає проходити рівень, а гра очікує поки гравець дійде до кінця рівня, або не програє. В залежності від результату на екрані перед гравцем з’явиться привітання з успішним проходженням рівня з пропозицією перейти до наступного рівня, або повідомлення про програш з можливістю перезавантажити рівень.

Крім цього гравець має можливість посеред проходження рівня поставити гру на паузу. Алгоритм взаємодії з меню паузи продемонстровано нижче (див. рис. 2).

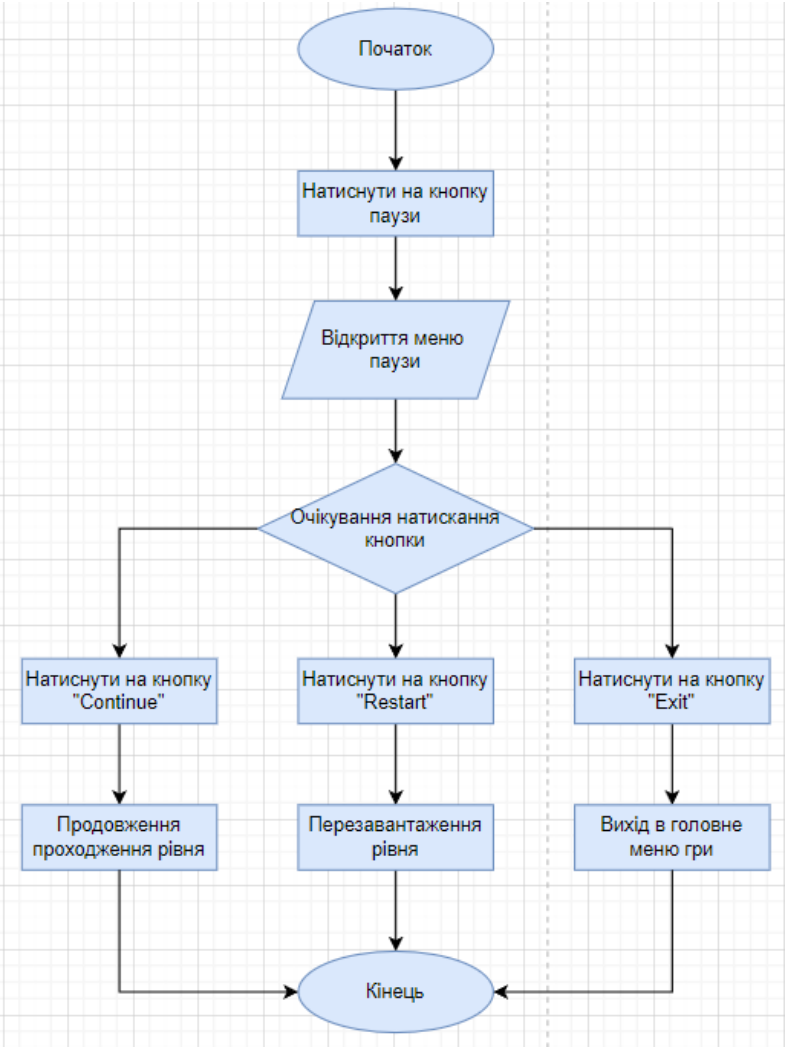

Рисунок 2 – Алгоритм взаємодії з меню паузи

2.4 Висновки до розділу 2

Проєктування гри передбачає ретельний підхід до кожного етапу, від концепції до вибору технологій і розробки алгоритмів. Вибір Unity як основного ігрового двигуна пояснюється його перевагами у продуктивності та гнучкості. Основна ідея гри та її механіки спроєктовані з акцентом на залучення гравців до активної участі в подіях та прийняття важливих рішень.

3 РОЗРОБКА ТА РЕАЛІЗАЦІЯ ГРИ

Розробка та реалізація є центральною фазою при створенні гри, під час якої фактичне створення гри здійснюється на основі концепцій, планів і прототипів, розроблених на попередніх фазах. Важливою частиною цього етапу є гнучкість і готовність до змін, оскільки в процесі розробки можуть виникати нові ідеї, проблеми та можливості покращити гру. Цей етап не вважається останнім, тому що після завершення реалізації починається етап QA-тестування. На цьому етапі відбувається пошук помилок та багів, які не були виявлені при розробці, перевірка основних механік та функцій гри, після чого обов’язково вносяться зміни для їх виправлення.

3.1 Створення базової локації

Створення базової локації це фундамент створення гри. З неї починається весь проєкт. Насамперед вона потрібна для того, щоб розробник міг протестувати початкові механіки переміщення персонажа та його взаємодію з навколишнім ігровим світом. Зазвичай базова локація створюється за допомогою простих, вбудованих в двигун фігур, або заздалегідь підготовлених спрайтів. Після чого над нею починають працювати Level-дизайнери, та перетворюють на перший рівень гри, або на навчальний рівень.

Початком створення базової локації є додавання у двигун «Unity» спеціальних префабів зі спрайтами. Префаб – збірні заготовки зі спрайтів, які мають відкориговані налаштування. Для створення базової локації було використано готові префаби з паку «2D Village Pack Free» [19] які є у вільному доступі в магазині «Unity Asset Store». Цей пак демонструє певний асортимент готових спрайтів та містить в собі сім різновидів будівель, хмари,

задній фон, гори та приклад земельного блоку. Окрім спрайтів структур та навколишнього середовища також було додано набір з головним персонажем та тренувальним манекеном [22], який демонструється як ворог головного героя (див. рис. 3).

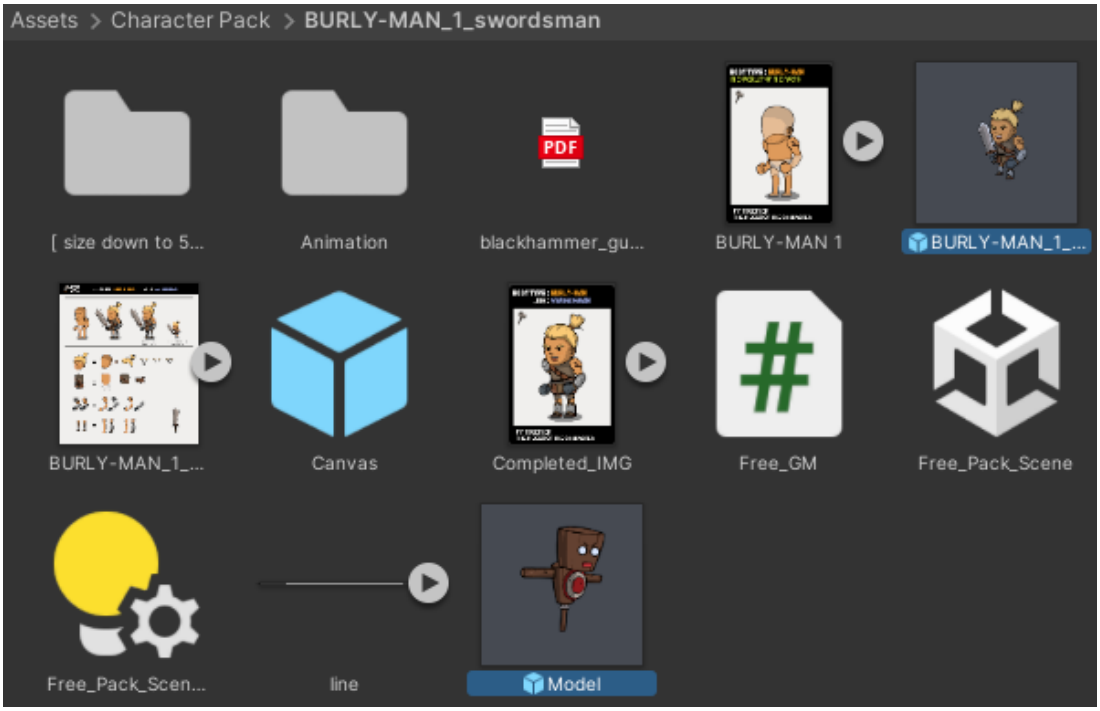

Рисунок 3 – Набір «Fantasy 2D Character: Free»

Після додавання усіх необхідних компонентів, спрайтів та префабів, починається етап створення самої локації. Цей етап можна розділити на два кроки.

Перший крок – додати на сцену усі об’єкти, які відображають задній фон та навколишній світ. До них входять усі спрайти будівель, дерев, хмар тощо. Це необхідно для того, щоб одразу додати локації певного шарму та краси, аби вона не виглядала порожньо.

Другий крок – розмістити на сцені об’єкти першого плану. Це об’єкти які безпосередньо беруть участь в ігровому процесі. Вони можуть бути звичайною поверхнею для переміщення персонажа, або кнопками, які персонаж повинен активувати, або інші предмети, з якими гравець може взаємодіяти.

Саме такий порядок дій дозволяє коректно спроєктувати та побудувати перший тестувальний рівень гри (див. рис. 4).

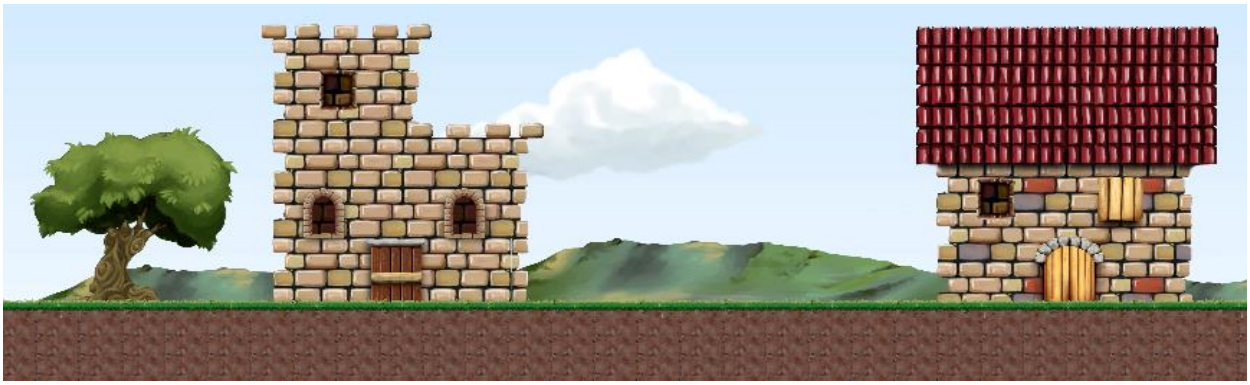

Рисунок 4 – Базова локація для початкового тестування

На базовому рівні представлено два будинки, задній фон, хмара, гори та земляне покриття з паку «2D Village Pack Free» [19] і дерево завантажене з сайту [20]. У майбутньому локація буде дороблена та розширена до розміру першого рівня.

3.2 Створення персонажа та реалізація скрипта переміщення

Коли базова локація була побудована, прийшов час для додавання головного героя на сцену та реалізація механіки його переміщення.

Переміщення головного героя у просторі можна реалізувати кількома різними способами, залежно від потреб гри та типу контролю. У «Unity» є спеціальний компонент «Rigidbody 2D», який наділяє об’єкт фізичними властивостями та дозволяє підкорятися законам фізики. Це надає можливість реалізувати реалістичне переміщення об’єкта з урахуванням сили гравітації та зіткнень з іншими об’єктами. «Rigidbody 2D» налічує як мінімум три основних параметри, які мають вплив на об’єкти. «Body type» визначає тип фізичного тіла і пливає на те, як це тіло взаємодіє з фізичним середовищем. В «Body type» існує всього три основні типи фізичних тіл.

Dynamic. Використовується для об’єктів, які повинні рухатися та взаємодіяти з іншими фізичними об’єктами. Він залежить від законів сили, гравітації та імпульсу та має масу і здатність до зіткнень.

Kinematic. Застосовується для об’єктів, які повинні рухатися, але не повинні взаємодіяти фізично з іншими об’єктами. Він не підпорядковується гравітації або іншим фізичним силам, а рух об’єкта налаштовується вручну, зміщенням трансформації, але об’єкти можуть взаємодіяти з динамічними об’єктами, штовхаючи їх.

Static. Найчастіше використовується для об’єктів, які не повинні рухатись. Такі об’єкти дуже потрібні при створенні твердих перешкод, наприклад для обмеження області дії гравця або для елементів оточення, таких як стіни, платформи тощо.

Перед тим, як програмувати персонажа, його спочатку потрібно додати на сцену та налаштувати компоненти його префабу. В першу чергу було додано компонент «Rigidbody 2D» для надання персонажу фізичних властивостей, маси та гравітації. Останнім за рахунком, але не за важливістю був доданий компонент «Capsule Collider 2D» (див. рис. 5).

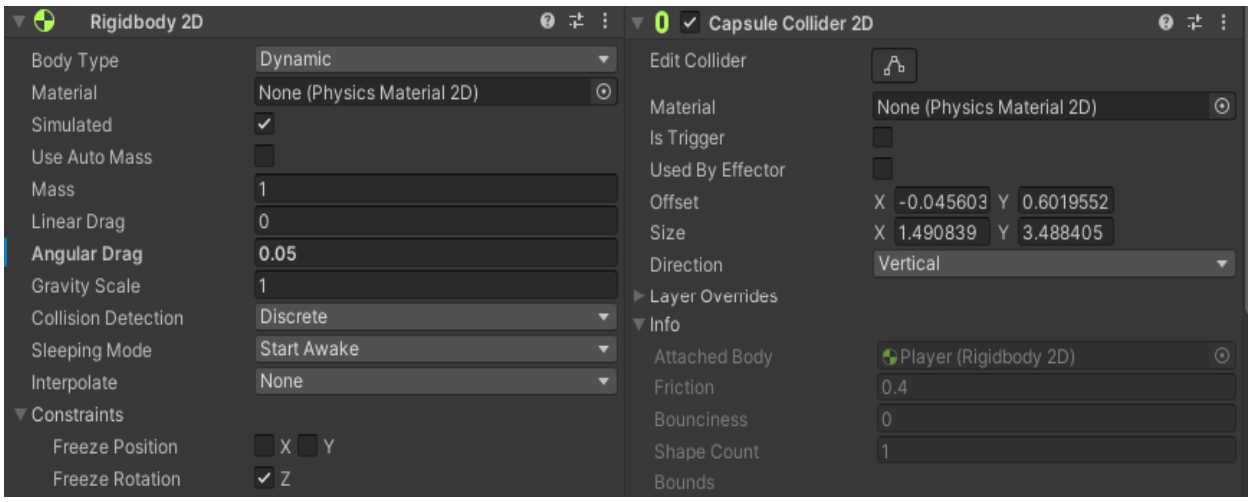

Рисунок 5 – Додані компоненти «Rigidbody 2D» та «Capsule Collider 2D»

«Capsule Collider 2D» – це компонент, який надає фізичні межі об’єктам для виявлення зіткнень та взаємодії між ними. Капсульний

колайдер має форму капсули, яка складається з двох півкіл, з'єднаних прямокутником, що охоплює об'єкти певної форми більш ефективно, ніж інші типи колайдерів.

Наступним кроком був аналіз складових префабу моделі. Префаб персонажа складається з окремих частин, які за необхідності, можна замінити на інші, таким чином створюючи ієрархію об'єктів моделі (див. рис. 6).

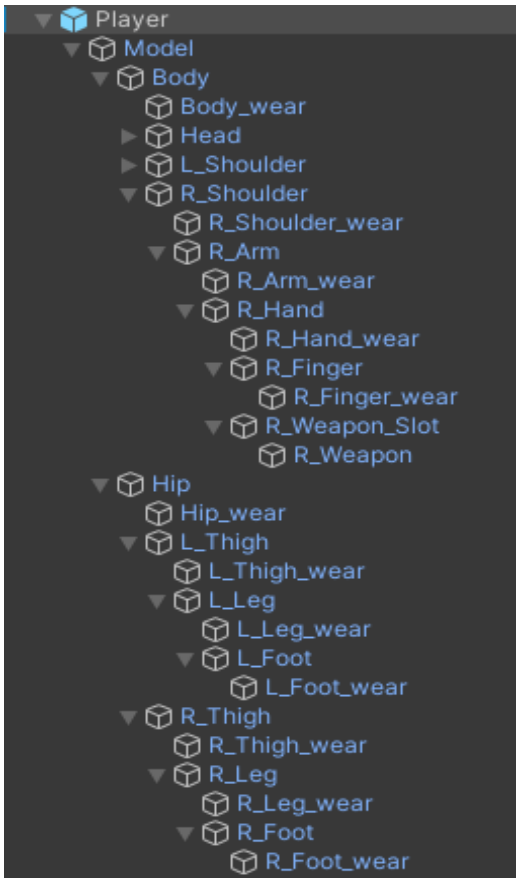

Рисунок 6 – Ієрархія частин моделі префабу

Кожен окремий об'єкт моделі має свою позицію на сцені, аби вона коректно відображалась з погляду гравця. З цього виходить, що всі об'єкти накладаються один поверх одного створюючи повноцінну картинку персонажа.

Для того, щоб усі елементи накладались один на одного правильно, потрібно відредагувати значення «Order in Layer». Цей параметр знаходиться у компоненті «Sprite Renderer» та відповідає за порядок відображення

елементів на сцені. З його допомогою можна з легкістю керувати тим, які спрайти будуть показані зверху, а які – під ними, коли вони перекриваються. Таким чином у префабі персонажу було змінено порядок накладання кожного елемента на сцені, тим самим зробивши правильну ієрархію накладання об’єктів (див. рис. 7). При зміні порядку накладання важливо не забувати робити діапазонні відступи між спрайтами. Це потрібно для того, аби була постійна можливість додати нові елементи до префабу персонажа не змінюючи або не перероблюючи весь порядок накладання знову.

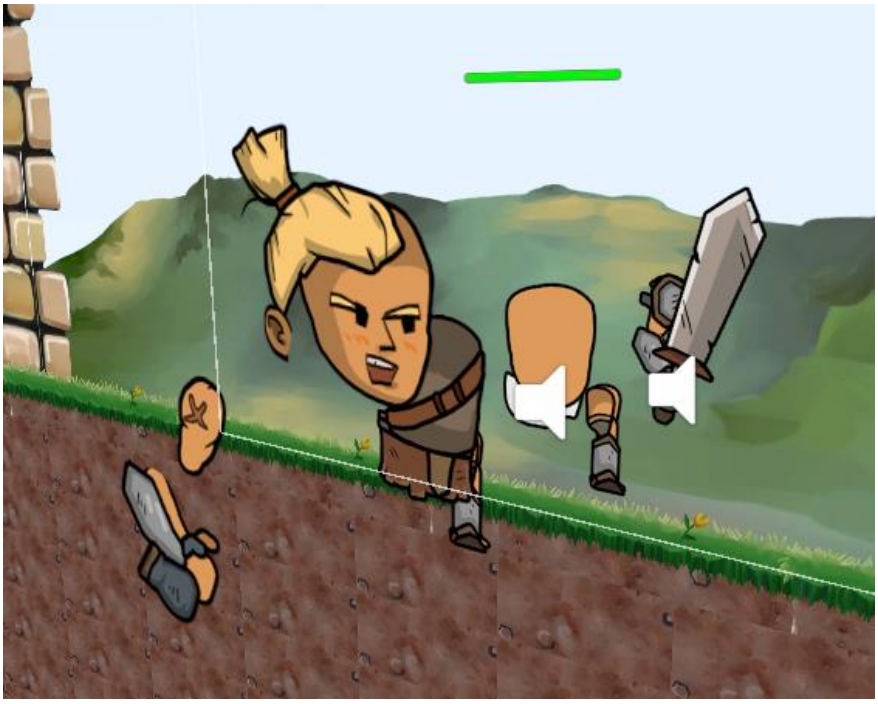

Рисунок 7 – Накладання спрайтів у 3-вимірному вигляді

Коли усі підготовчі дії виконано, порядок накладання елементів встановлено, а всі необхідні компоненти додано, настає час до написання алгоритму пересування персонажа.

Першим етапом створення скрипта пересування головного героя є створення нового файлу програми з розширенням мови програмування C#. Розробити програму можна у будь-якому текстовому редакторі який підтримує мову C#, але для більш зручної та комфортної реалізації у проєкті був використаний редактор «Visual Studio Code».

«Visual Studio Code» – це безкоштовний, легкий і потужний редактор вихідного коду, розроблений Microsoft. Він отримав велику популярність серед розробників завдяки своїй зручності, розширюваності та багатофункціональності [21].

В нього одразу вбудовано підтримку JavaScript, TypeScript та Node.js, а також він має розвинену екосистему розширень для інших мов програмування таких як C++, C#, Java, Python, PHP тощо.

Цей редактор було обрано з погляду зручності використання та великої кількості розширень, особливо для розробки скриптів на «Unity».

Оскільки мова програмування C# вважається об’єктно-орієнтованою мовою, написання програми починається зі створення базового класу. Завдяки редактору, після відкриття файлу, можна побачити одразу створений клас та дві базові функції цього класу «void Start» та «void Update».

«Void Start()» – це функція, яка викликається один раз перед першим кадром, коли активується об’єкт. У ній розміщується код, який відповідає за початкову ініціалізацію змінних, налаштування стану об’єктів тощо.

«Void Update» – функція, яка викликається один раз за один кадр. Вона містить в собі основні методи для реалізації логіки, яка має виконуватися постійно протягом усього часу існування об’єкта. Використовується для обробки введених користувачем даних, оновлення анімацій, перевірки умов та виконання інших повторюваних завдань.

Переміщення гравця в коді реалізовано за допомогою основного компонента його префабу, а саме «Rigidbody 2D». В наслідок того, що компонент «Rigidbody 2D» надає персонажу фізичні властивості, зазвичай використовується функція «FixedUpdate()». Тільки у цій функції мають виконуватись всі обчислення та оновлення пов’язані з фізикою, тому що вона викликається з фіксованим часовим інтервалом, що робить симуляцію більш стабільною, а переміщення плавними. Код переміщення персонажа ліворуч та праворуч виглядає наступним чином (див. рис. 8).

```
void Start(){
    _rb = GetComponent<Rigidbody2D>();
}
void FixedUpdate(){
   _horizontal = Input.GetAxis("Horizontal");
   _rb.velocity = new Vector2(_horizontal * speedX * speedMultiplier * Time.fixedDeltaTime, _rb.velocity.y);
}
```

Рисунок 8 – Реалізація переміщення персонажа ліворуч та праворуч

Після реалізації переміщення гравця, наступним кроком було створення можливості стрибка. Під час розробки алгоритму стрибка була виявлена негативна особливість, через яку персонаж міг стрибати навіть тоді, коли знаходився в повітрі. Цей баг був виправлений завдяки створенню тайлу землі тегу «Ground». В наслідок чого в програмі була створена окрема функція, яка відповідає за перевірку дотику колайдера персонажа з колайдером тайла землі.

Залежно від умови логічна змінна приймала одне з двох значень – правда або брехня. Реалізація алгоритму стрибка персонажа продемонстровано нижче (див. рис. 9).

```
void FixedUpdate(){
      …
      if (_isJump){
         _rb.AddForce(new Vector2(0f, 450f));
         _isGround = false;
         _isJump = false;
      }}
```

Рисунок 9 – Реалізація стрибка головного героя при натисканні на клавішу

Перевірка дотику між персонажем та землею (див. рис. 10).

```
void OnCollisionEnter2D(Collision2D other){
    if (other.gameObject.CompareTag("Ground")){
        _isGround = true;
    }
  }
```

Рисунок 10 – Перевірку дотику колайдерів персонажа і тайла землі

Останнім кроком реалізації повноцінного переміщення персонажа було створення поворотів моделі залежно від напрямку руху. Цю механіку було реалізовано простим віддзеркаленням моделі завдяки зміні у компоненті «Transform» значення «Scale x». Логіка алгоритму полягає у тому, що залежно від значення логічної змінної та напрямку руху в функції «FixedUpdate()» буде викликатися метод «Flip()»який і буде відповідати за віддзеркалення персонажа (див. рис. 11).

```
void FixedUpdate(){
    …
    if (_horizontal > 0f && !_isFacingRight){
        Flip();
    } else if(_horizontal < 0f && _isFacingRight){
        Flip();
    }}
void Flip(){
    _isFacingRight = !_isFacingRight;
    Vector3 playerScale = playerModelTransform.localScale;
    playerScale.x *= -1;
    playerModelTransform.localScale = playerScale;}
```

Рисунок 11 – Реалізація повороту персонажа та виклик метода Flip()

3.3 Налаштування камери персонажа

Налаштування камери, яка буде відстежувати позицію гравця та слідувати за ним у всіх напрямках, є однією з головних механік гри. Саме камера є невід'ємною частиною будь-якої гри, тому що з її допомогою гравець має можливість орієнтуватися в ігровому світі, розуміти що знаходиться навколо та утримувати фокус на головній дійовій особі та її взаємодіях. Під час налаштування камери дуже важливо дотримуватись плавності її руху, аби зробити гру більш гарною та кінематографічною, покращуючи візуальну привабливість. Роблячи короткий підсумок, можна сказати, що камера є досить важливим компонентом при створенні якісного і захопливого ігрового досвіду, покращуючи керування, орієнтацію та візуальну привабливість гри.

Для реалізації камери було використано ассет «Cinemachine» який вбудований в «Unity». «Cinemachine» – це потужний інструмент для керування камерами в Unity, який значно спрощує створення складних і динамічних систем камер для ігор та інших інтерактивних програм. Він надає гнучкі та інтуїтивно зрозумілі можливості для створення камер, які можуть стежити за персонажем, перемикатися між різними точками огляду тощо.

Для початку ассет камери було встановлено через «Package Manager», після чого на сцену був доданий компонент «Virtual Camera». Одразу, як тільки об'єкт з'явився на сцені, він додав новий компонент «CinemachineBrain» до об'єкта «Main Camera». Цей компонент необхідний для створення зв'язку між головною та віртуальною камерами. Головні зміни відбуваються в налаштуваннях об'єкта «Virtual Camera». Першим кроком є зв'язування віртуальної камери з об'єктом, за яким камера буде рухатись. Цим об'єктом є головний герой. Вслід за цим йде налаштування зони видимості камери, та зони її руху (див. рис. 12). Кожна зона розфарбована різним кольором. Синім кольором позначена «Soft Zone». Це головна зона, в межах якої знаходиться персонаж. В цій зоні камера буде рухатися повільніше і плавніше, створюючи м'який перехід.

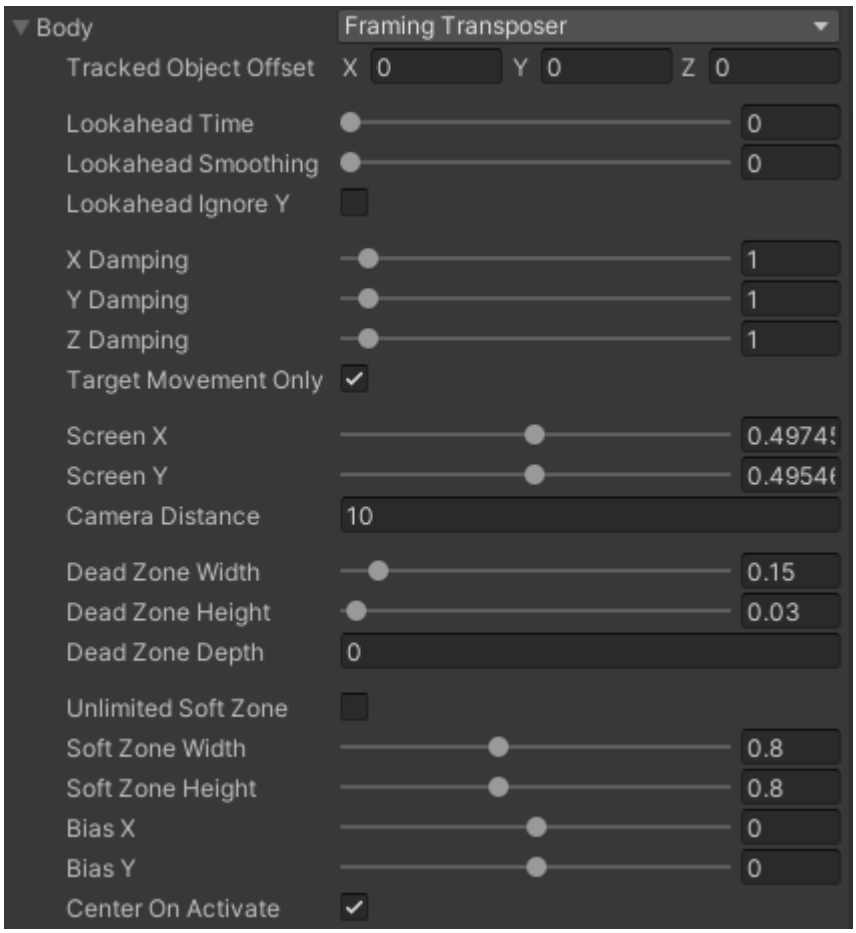

Рисунок 12 – Параметри налаштувань «CinemachineVirtualCamera»

Крім налаштування головної зони, також необхідно налаштувати зону, в якій персонаж має можливість вільно рухатись без зміщення камери. За цю зону відповідає параметр «Dead Zone». Він також допомагає зменшити дрібні рухи камери, які можуть виникати через невеликі переміщення головного героя.

Значення «Dead Zone Width» були змінені з одиниці на 0.15, а значення «Dead Zone Height» – з одинці на 0.03. Після ряду тестувань параметра було визначено, що саме ці значення добре підходять до персонажа відносно його розміру та швидкості руху. Адже від швидкості руху персонажа залежить і швидкість руху камери.

Якщо камера рухається занадто швидко, тоді ігровий досвід користувача може бути негативним. Кінцеві налаштування камери представлені на малюнку нижче (див. рис. 13).

Рисунок 13 – Візуалізовані налаштування камери розділені на зони різного кольору

3.4 Створення фінішу рівня

Рівень гри повинен мати логічне завершення, логічний кінець. Для цього було створено фініш. Фініш – це об’єкт, який позначає логічне завершення рівня, та є відправною точкою до наступного рівня. Фінішем може бути абсолютно будь-який предмет, чи то двері, прапор, будинок, край карти або просто порожня область в кінці рівня. Кожне досягнення фінішу, це не просто кінець рівня, це можливість отримати нову інформацію про персонажів, світ гри або відкриття сюжетних таємниць. Перед переходом на новий рівень після завершення попереднього, гравець може отримати яку-небудь нагороду, або ресурси, які йому вдалося знайти за час проходження рівня. Також кінець рівня може розблокувати нові рівні, локації чи здібності головного героя, які він зможе застосовувати у подальших рівнях.

Для створення фінішу було використано одну з будівель ассету «2D Village Pack Free». На об’єкт будівлі було додано компонент «Box Collider

2D», для того, щоб програмно відстежувати, коли головний герой перетне межу колайдера фінішу. Оскільки колайдер задає фізичні межі об'єкта, з цього виходить що персонаж просто зіштовхнеться з ним та не зможе пройти наскрізь. Така практика може погано вплинути на ігровий досвід, тому що, наприклад користувачу необхідно буде натиснути перемикач або потягнути важіль, який знаходиться в межах колайдера, щоб активувати фініш. Тому щоб виправити цю помилку, у налаштуваннях «Box Collider 2D» було встановлено параметр «Is Trigger» (див. рис. 14). Цей параметр дозволяє іншим об'єктам проходити крізь колайдер не зіштовхуючись з ним.

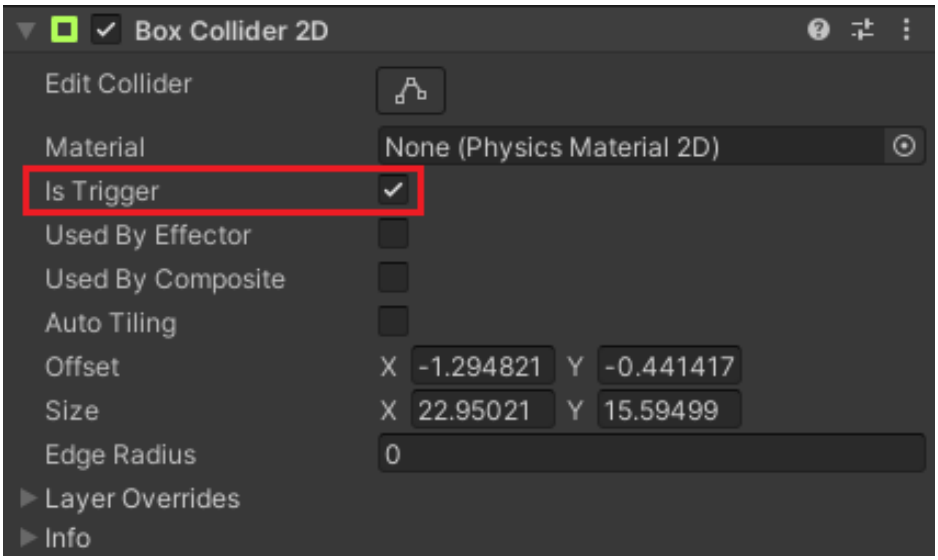

Рисунок 14 – Встановлення активної позначки для параметра «Is Trigger»

Після завершення усіх налаштувань для об'єкта, необхідно написати скрипт для роботи фінішу. Цей скрипт відповідає за зміну логічної змінної від якої залежить завершення рівня та вивід привітання про успішно пройдений рівень (див. рис. 15). Аби рівень завершився, гравець повинен підійти до фінішу та натиснути кнопку «F» на клавіатурі. Усі дії, які може виконувати персонаж, належать скрипту «PlayerControl», тому і відстеження натискання кнопки також знаходиться у цьому скрипті. Таке розбиття дій на різні скрипти допомагає оптимізувати програму, не навантажуючи її великою кількістю кода.

```
public void Activate(){
        _isActivated = true;
        messageUI.SetActive(false);
    }
public void FinishLevel(){
    if (_isActivated){
        levelCompleteCanvas.SetActive(true);
        gameObject.SetActive(false);
        Time.timeScale = 0;
    }
    else {
        messageUI.SetActive(true);
    }
```

Рисунок 15 – Алгоритм закінчення рівня та виводу повідомлення про виграш

Однак аби досягти фінішу було цікавіше, на рівень було додано важіль, який гравець повинен знайти та активувати. Тільки після активації важеля, у гравця з’явиться можливість активувати фініш. В іншому випадку, якщо гравець завчасно дійде до фінішу та натисне активуючу кнопку, поряд з будівлею з’явиться надпис-повідомлення про необхідність активувати важіль для подальшого проходження рівня. Така механіка додає до гри цікавості та інтриги, тому що гравець розуміє, що рівень пройдено не до кінця і на локації ще є місця, які він не дослідив.

Створення підказок на рівні, особливо на початку гри, дуже важливі, тому що коли користувач вперше завантажує гру та потрапляє на перший рівень, він не зовсім розуміє що саме йому необхідно робити, на які кнопки натискати та звідки почати свою подорож (див. рис. 16). Крім підказок, розробники часто використовують механіку створення навчального рівня, де користувача навчають, як грати в гру.

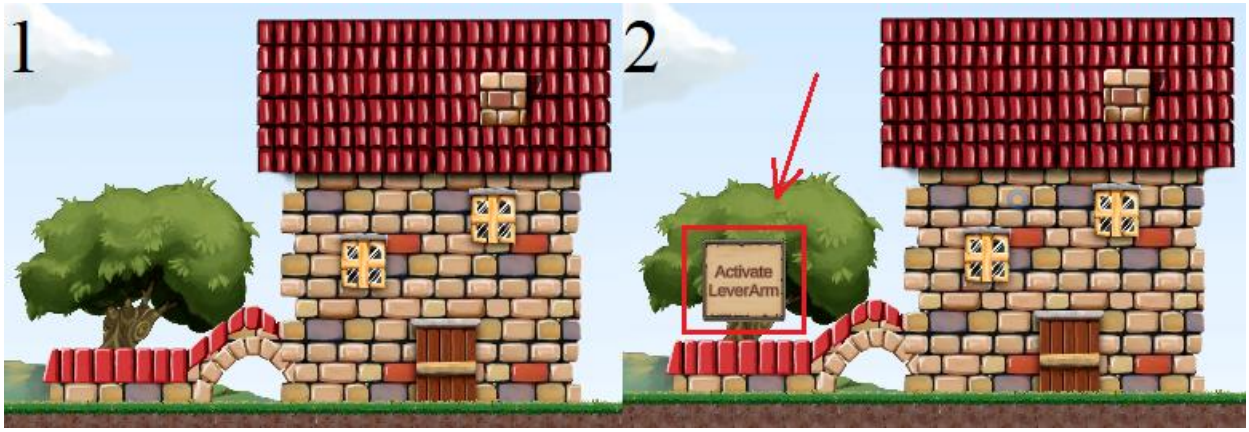

Рисунок 16 – Відображення підказки біля фінішу (1 – без підказки; 2 – з підказкою)

Алгоритм роботи важеля працює завдяки зверненню до об’єкта «Finish» та виклику його функції «Activate()» яка і відповідає за активацію фінішу та вимкнення підказки у разі її появи (див. рис. 17).

```
void Start(){
    _finish = GameObject.FindGameObjectWithTag("Finish").GetComponent<Finish>();
}
public void ActivateLeverArm(){
    animator.SetTrigger("activate");
    _finish.Activate();
}
```

Рисунок 17 – Програма активації фінішу

Також до важеля була розроблена та записана анімація, яка активується при натисканні на нього за допомогою скрипта.

3.5 Створення ворогів, здоров’я та бойової системи

Вороги є ключовими елементами ігрового процесу. Вони підтримують інтерес до гри, додають складність, сприяють розвитку навичок і

допомагають розвивати сюжет гри. Вороги створюють перешкоди, які гравець повинен подолати, що робить гру захопливою і складною. Зустрічаючись із ворогами, гравець відчуває напругу та адреналін, що підсилює емоційний досвід гри. З кожним новим рівнем вороги стають сильнішими, а рівні складнішими. Ускладнення гри напряму залежить від рівня майстерності гравця. Тому, щоб проаналізувати та правильно ускладнювати гру, потрібно створити графік кривої навчання (див. рис. 18).

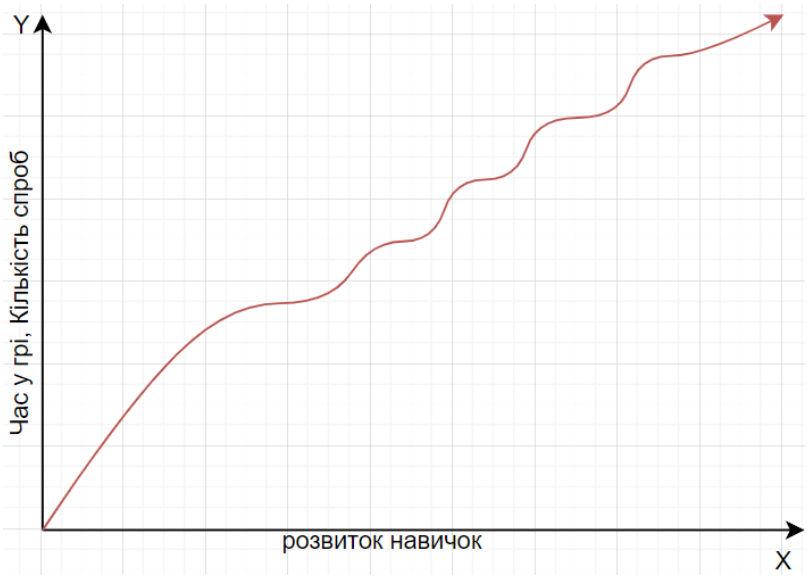

Рисунок 18 – Графік кривої навчання

Кривизна графіку обумовлена тим, що з часом проходження гри гравець відкриває нові навички та починає вчитися грати з ними.

Тому при додаванні до гри нових видів ворогів, зброї або навичок, слід коректно їх збалансувати, аби гра не була дуже простою чи дуже складною. Необхідно знайти золоту середину складності гри, щоб гравці розчарувалися і не пішли з гри.

Створення ворога не дуже відрізняється від створення головного героя. Перші кроки повністю збігаються з етапом редагування персонажа. Спочатку було додано основні компоненти «Rigidbody 2D» та «Capsule Collider 2D». У «Rigidbody 2D» був змінений параметр «Body type» на динамічний, а вага збільшена до 100 одиниць (див. рис. 19).

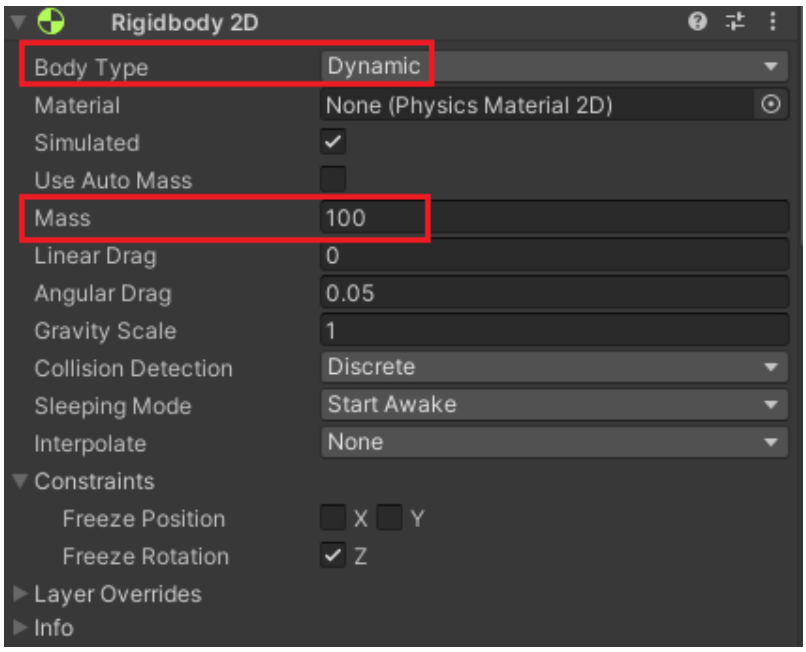

Рисунок 19 – Налаштування «Rigidbody 2D»

Наступним етапом було налагодження порядку відображення елементів на сцені (див. рис. 20). Цей етап займає більше часу ніж такі самі налаштування у префабі головного героя. Це пов’язано з тим, що ворог повинен знаходитись нижче за накладанням ніж персонаж, тому потрібно постійно порівнювати значення «Order in Layer» героя та ворога, аби не з’явилося візуальних багів. Цей етап вважається досить складним, тому що потрібно приділяти багато уваги до деталей, з яких складається об’єкт.

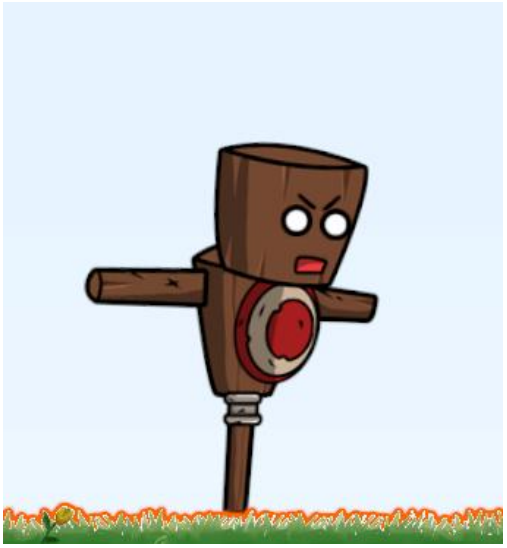

Рисунок 20 – Вигляд ворога після редагування накладання спрайтів

Далі настав етап опрацювання руху ворога. Якщо ворог буде просто стояти на місці, тоді гравцю не буде цікаво з ним битися. Основне завдання ворога створювати перешкоди на шляху гравця до фінішу. Тому динамічні вороги дуже важливі для ігор жанрів екшен. Основні механіки ворогів:

– наявність здоров’я;
– можливість заподіювання шкоди персонажу;
– механіка патрулювання місцевості;
– можливість переходу в агресивний стан при виявленні персонажа.

Усі ці механіки були розроблені та додані до ворога. Наявність здоров’я обов’язкова умова для ворогів, оскільки гравець повинен мати можливість подолати ворога. Також, щоб гравець відчував складність проходження рівнів, ворог так само має можливість дати відсіч гравцеві. Це відбувається, коли гравець потрапляє у поле зору ворога, тоді ворог переходить у режим переслідування, та намагається наздогнати гравця. Якщо гравець виходить з поля зору ворога на декілька секунд, ворог повертається до спокійного стану, та повертається у зону патрулювання. Якщо на сцену додано декілька ворогів одного префабу, тоді кожного з них можна налаштувати окремо, не змінюючи основний префаб. Для зручності налаштування дистанцій у скрипті було додано візуалізацію маршрутів (див. рис. 21). Детально ознайомитись зі скриптом можна у додатку А.3.

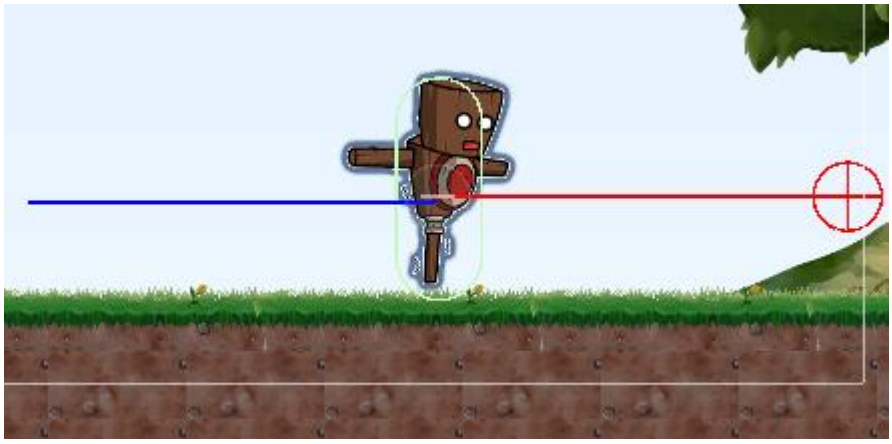

Рисунок 21 – Візуалізація патрулювання та поля зору ворога, де синім кольором позначена дистанція патрулювання, а червоним – поле зору

Здоров’я ворога було розроблено з використанням компонента UI, «Canvas». «Canvas» – компонент, який використовується для створення та керування елементами інтерфейсу користувача (UI). Це основна структура для всіх елементів інтерфейсу користувача в грі. Для створення смуги здоров’я використовувався елемент «Slider». Він дуже зручний у використанні, тому що він одразу має рухому смугу, яку можна перефарбувати у потрібний колір. Перш ніж налаштовувати смугу здоров’я, необхідно трохи змінити налаштування «Canvas». Для цього у компоненті є параметр «Renderer mode» який відповідає за розташування «Canvas» у просторі сцени. Зазвичай UI компоненти знаходяться поверх усієї гри, тому вони ніяк не впливають на ігровий процес. Але у випадку смуги здоров’я, параметр «Renderer mode» слід замінити з «Screen Space – Overlay» на «World Space». Ця дія приведе до того, що «Canvas» буде визначати своє положення не за глобальними координатами, а за координатами гри. Після чого «Canvas» необхідно додати до префабу ворога, щоб він рухався одночасно з ним. Така сам процедура відбувається і для персонажа. Єдина відмінність полягає у кольорах смуги здоров’я (див. рис. 22).

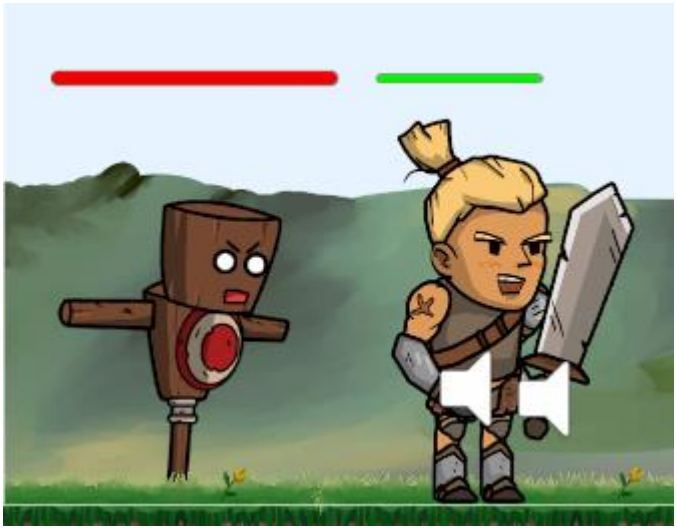

Рисунок 22 – Відображення «Canvas» зі смугою здоров’я персонажа та ворога

Останній етап створення ворога – це реалізація системи бою головного героя з супротивником. Основна механіка гри, це бій, тому дуже важливо

правильно її реалізувати. Є два основних випадки, коли відбувається алгоритм бою. Перший випадок – це коли ворог наздоганяє головного героя та стикається з його колайдером. В цей момент у персонажа віднімається фіксована частина здоров'я. Другий випадок настає тоді, коли вже гравець, за допомогою своєї зброї б'є по колайдеру ворога. Одразу після цього у ворога спрацьовує анімація отримання шкоди. В цей момент виникає проблема. Вона полягає у тому, що меч персонажа не має фізичної оболонки. Тому знайшовши меч в ієрархії префабу головного героя, до нього було додано компонент «Box Collider 2D» який і буде перевірятись дотик меча до ворога. Але якщо дотик буде перевірятися кожен кадр гри, тоді гравець може просто дуже близько підійти до ворога, щоб колайдери ворога та меча доторкнулись. В такому випадку ворог може померти миттєво. Це дуже серйозний недолік, який суттєво впливає на ігровий досвід. Тому, щоб уникнути такого багу, було створено скрипт, який спрацьовує тільки тоді, коли гравець натискає на ліву кнопку миші. Це дозволяє уникнути перевірки дотику на кожному кадрі, а робити її тільки у момент, коли гравець б'є ворога (див. рис. 23).

```
private void OnTriggerEnter2D(Collider2D other)
    {
        EnemyHealth enemyHealth = other.GetComponent<EnemyHealth>();
        if (enemyHealth != null && _attackController.IsAttack){
            enemyHealth.ReduceHealth(damage);
            enemyHitSound.Play();
        }
    }
```

Рисунок 23 – Частина коду, яка відповідає за перевірку дотику до колайдера та віднімання здоров'я ворога

3.6 Створення меню паузи, головного меню та повідомлень

Наявність головного меню, меню паузи та повідомлень про виграш і програш є ключовими елементами інтерфейсу користувача. Вони

забезпечують зручність, організованість і задоволення від гри.

Головне меню це як обкладинка книги. Перше, що бачить гравець, коли вперше завантажує гру, це головне меню. Дивлячись на нього, можна одразу зрозуміти який вигляд має і сама гра. Тому головне меню повинно виглядати якомога красивіше та привабливіше.

Головними критеріями якості головного меню є:

– зручність використання;
– інтуїтивно зрозумілий інтерфейс;
– гарне оформлення;
– наявність тільки важливих компонентів.

Якщо дотримуватись цих критеріїв, тоді головне меню буде виглядати дуже привабливо та технічно правильно, що додасть пару балів до оцінки якості меню користувачем.

Етап розробки головного меню починається зі створення нової сцени до якої одразу додається компонент «Canvas».

Створення нової сцени необхідно для того, щоб відділити головне меню від основного контенту гри. У головному меню не потрібні усі елементи та префаби гри, тому нова, чиста сцена є першочерговим етапом створення головного меню.

Одразу після створення «Canvas», до нього був доданий компонент «Panel», на якому і будуть розміщені усі кнопки. До компонента «Panel» можна додати задній фон, який буде відображатись одразу після входу в гру. Задній фон – це звичайна картинка. Для створення всіх меню був використаний ассет «Fantasy Wooden GUI : Free» [23]. Як тільки був створений задній фон меню, необхідно створити кнопки, з якими користувач зможе взаємодіяти.

Кнопкам обов’язково необхідно налаштувати текстове поле, змінивши його розмір, колір та шрифт. Також до кнопок було додано фон, щоб вони виглядали красиво та естетично (див. рис. 24).

Рисунок 24 – Вигляд головного меню гри очима гравця

Щоб створити алгоритм, спочатку необхідно додати скрипт до компонента «Canvas». Алгоритм роботи кнопок дуже простий. При натисканні кнопки старт, для користувача завантажиться перший рівень та почнеться гра. У кожної сцени є свій порядок, тому при написанні програми потрібно вказати номер сцени, яку необхідно завантажити. Порядок сцен корегується у налаштуваннях проєкту (див. рис. 25)

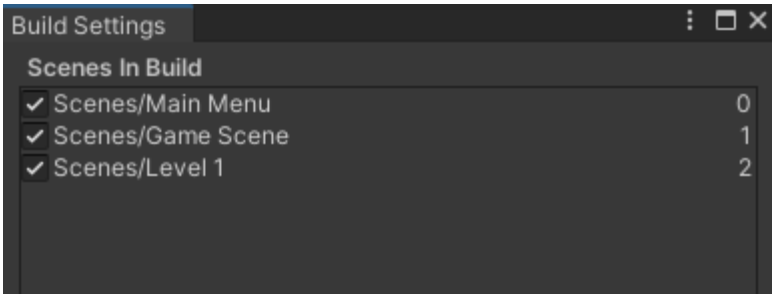

Рисунок 25 – Порядок завантаження сцен

Після корегування списку, можна переходити до створення алгоритму (див. рис. 26)

```
public void StartHandler(){
        SceneManager.LoadScene(1);
        Time.timeScale = 1;
    }
    public void ExitHandler(){
        Application.Quit();
    }
```

Рисунок 26 – Частина коду яка відповідає за натискання кнопок

Наступним за рахунком було створено меню паузи. Воно знаходиться безпосередньо на рівнях гри, і відповідає за можливість зупинки гри під час проходження. Оскільки меню не належить до частини геймплею, у компоненті «Canvas» параметр «Renderer Mode» встановлено як «Screen Space – Overlay». Це потрібно для того, щоб у меню паузи була прив’язка до глобальних координат, а не до координат гри та воно завжди відображалось на екрані гравця.

Після переходу з головного меню до рівня гри, гравець одразу бачить локацію та кнопку паузи зліва зверху. Одразу після натискання на кнопку гра зупиняється, а всі об’єкти завмирають. Перед гравцем з’являється анімоване меню паузи у якому є три кнопки. Кнопка «Continue», «Restart» та «Exit». Кнопка «Continue» повертає користувача до гри знімаючи паузу та розморожуючи усі об’єкти. «Restart», за допомогою коду, перезавантажує рівень починаючи його з початку. Кнопка «Exit» повертає гравця у головне меню, де гравець може вийти з гри або почати нову гру (див. рис. 27).

Рисунок 27 – Кнопка паузи зліва зверху та меню паузи з кнопками

Після початку гри у гравця є декілька можливих варіантів розвитку гри. Користувач може активувати важіль, дійти до кінця рівня, активувати фініш

та пройти рівень, або він може програти, загинувши від супротивника. Саме для таких випадків було розроблено два меню-повідомлення. Залежно від результату під час проходження рівня перед гравцем може з’явитися повідомлення про виграш або програш. Такі повідомлення є досить важливими для орієнтування гравців. Вони повідомляють його про результат гри та надають можливість перейти на наступний рівень або навпаки перезавантажити рівень. Ці два меню дуже схожі між собою, в них є кнопки для взаємодії гравця з грою, але в меню програшу є червоний надпис, який і повідомляє гравця про програш (див. рис. 28, 29).

Рисунок 28 – Меню перемоги гравця з трьома кнопками

Рисунок 29 – Меню програшу гравця з надписом та двома кнопками

3.7 Звуки та анімація

Музика та анімація відіграють важливу роль у створенні захопливого ігрового досвіду. Вони забезпечують емоційний зв’язок, занурення в ігровий світ і візуальне задоволення.

Завдяки музичному супроводу у грі створюється певна атмосфера, що відповідає подіям у грі. Гарний саундтрек може повністю змінити враження про гру, та захопити гравців ще більше. За допомогою музики можна керувати динамічністю подій у грі. Чим швидше музика, тим динамічніше відбуваються події у грі, або навпаки спокійна музика спонукає гравця до розслаблення.

При додаванні музичного супроводу використовувався сайт з безкоштовними мелодіями для ігор «OpenGameArt.org» [24]. Щоб зробити музичний супровід, до кожного об’єкту, який повинен мати звук, був доданий компонент «Audio Source». Цей компонент відповідає за налаштування музичної доріжки, зміну гучності, зацикленні тощо. Після чого використання звуку додається до скрипта, який відповідає за певні дії (див. рис. 30).

```
void Update()
   {
      if (Input.GetKeyDown(KeyCode.W) && _isGround){
         _isJump = true;
         jumpSound.Play();
      }
```

Рисунок 30 – Приклад виклику звуку стрибка персонажа

За такою аналогією були додані звуки ударів в повітрі та по ворогах, а також звук попадання по персонажу. Також у грі присутня фонова музика, яка грає в головному меню так на рівнях гри.

Анімації у грі використовуються для пожвавлення ігрового процесу. Вони додають до гри життя, створюють відчуття реалізму та допомагають зануритись ігровий процес. Для створення анімацій в «Unity» є вбудований редактор анімацій та аніматор. Аніматор використовується для відстеження порядку програвання анімацій залежно від дій гравця (див. рис. 31).

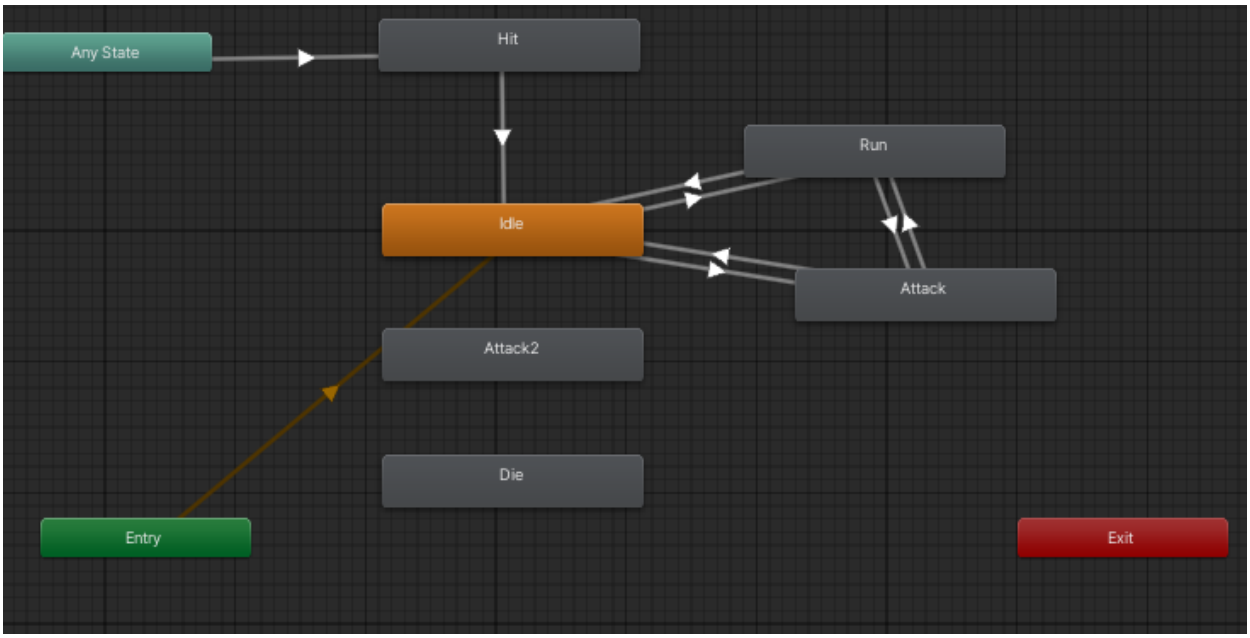

Рисунок 31 – Порядок виконання анімацій головного героя

Використовуючи вбудований редактор анімацій були створені анімації появи меню та анімація активування важеля (див. рис. 32).

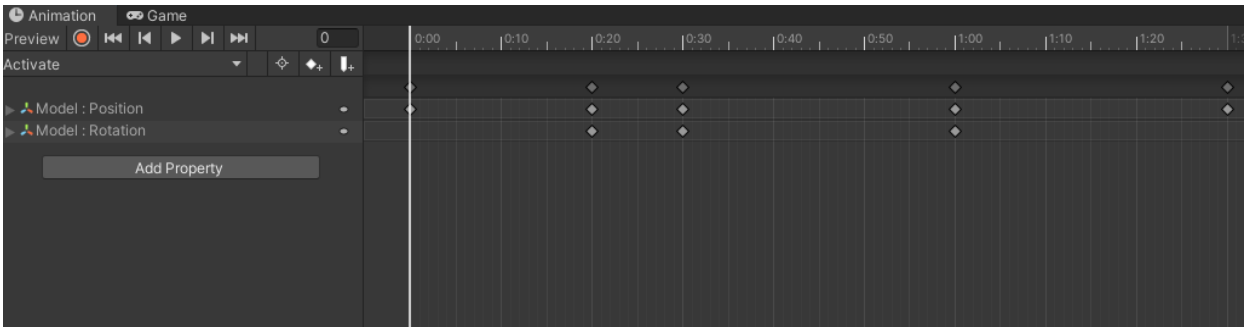

Рисунок 32 – Створена анімація активації важеля засобами «Unity»

ВИСНОВКИ

У процесі створення гри важливо враховувати різні аспекти, які разом складають повний ігровий досвід. Кожен етап розробки гри відіграє ключову роль у формуванні загальної якості та привабливості гри.

Створення базової локації є основою для всієї гри. Він визначає, як буде виглядати і відчуватися ігровий світ. Від добре продуманої локації залежить, наскільки захопливим буде дослідження гри. Локація повинна бути детальною, мати цікаві елементи та об’єкти, які взаємодіють з гравцем, а також забезпечувати логічний і приємний шлях для гравця.

Персонаж є головною складовою гри, з якою гравець постійно взаємодіє. Створення цікавого та добре анімованого персонажа забезпечує гравцеві емоційний зв’язок із грою. Реалізація сценарію руху повинна бути плавною і точною, щоб керування персонажем було інтуїтивно зрозумілим і приємним. Це впливає на задоволення гравця від гри та може визначити її успіх.

Камера визначає, як гравець бачить і сприймає ігровий світ. Правильні налаштування камери мають вирішальне значення для комфортної та приємної гри. Камера повинна плавно слідувати за персонажем, забезпечуючи хороший огляд і не створюючи незручностей під час гри.

Завершення рівня є важливим елементом, який допомагає гравцеві відчути досягнення та прогрес у грі. Він повинен бути чітко позначений і мати відповідні візуальні та звукові ефекти, які підкреслюють успіх гравця. Це стимулює гравця продовжувати гру та досягати нових цілей.

Вороги та бойова система додають грі динаміки та викликів. Вороги повинні бути різноманітними та цікавими, з різними рівнями складності. Система здоров’я дозволяє гравцеві стежити за своїм станом, а бойова система повинна бути інтуїтивно зрозумілою і захопливою. Разом ці елементи створюють напругу та адреналін, що робить гру цікавою та викликає бажання вдосконалювати свої навички.

Меню паузи, головне меню та сповіщення про програші та виграші важливі для забезпечення зручності та організації ігрового процесу. Головне меню допомагає гравцеві легко орієнтуватися в налаштуваннях і параметрах гри, меню паузи дозволяє зробити перерву і налаштувати параметри гри в процесі, а повідомлення про програш і виграш забезпечують гравця зворотним зв’язком і мотивацією.

Звуки та анімація додають грі глибини та реалістичності. Вони забезпечують емоційний зв’язок між гравцем і ігровим світом, роблячи події в грі більш захопливими та інтерактивними. Музика створює атмосферу, підсилює емоції та додає грі динаміки, а анімація робить персонажів і предмети більш живими та реалістичними.

Кожен з описаних елементів є важливою складовою у створенні успішної та захопливої гри. Базова локація, персонаж, камера, фініш рівня, вороги, меню та звуки разом створюють повноцінний ігровий досвід, який залишає гравця задоволеним і мотивованим продовжувати грати. Важливо ретельно опрацювати кожен з цих аспектів, щоб досягти високої якості гри та привернути увагу гравців.

ПЕРЕЛІК ПОСИЛАНЬ

1. Source2. *Valve Developer Community.* URL: https://developer.valvesoftware.com/wiki/Source_2?uselang=uk (дата звернення: 11.04.2024).
2. RAGE. *LinkedIn.* URL: https://www.linkedin.com/pulse/rage-swasthik-k-ln5ef (дата звернення: 11.04.2024).
3. What is Unity. *Zenva.* URL: https://gamedevacademy.org/what-is-unity/ (дата звернення: 13.04.2024).
4. Unreal Engine 4. *ArtCraft.* URL: https://blog.artcraft.net.ua/unreal-engine-4-dlya-kino (дата звернення: 14.04.2024).
5. Godot Engine. *GitHub.* URL: https://github.com/godotengine/godot (дата звернення: 14.04.2024).
6. Construct 3. *Common sense education.* URL: https://www.commonsense.org/education/reviews/construct-3 (дата звернення: 14.04.2024).
7. Ігрові жанри. *QATestLab.* URL: https://training.qatestlab.com/blog/technical-articles/games-genres/ (дата звернення: 16.04.2024).
8. Adobe Photoshop. *Wikipedia.* URL: https://uk.wikipedia.org/wiki/Adobe_Photoshop (дата звернення: 17.04.2024).
9. About GIMP. *GIMP.* URL: https://www.gimp.org/about/introduction.html (дата звернення: 17.04.2024).
10. The Software. *Blender.* URL: https://www.blender.org/about/ (дата звернення: 17.04.2024).
11. What is Autodesk Maya. *Vanas.* URL: https://www.vanas.ca/en/blog/what-is-autodesk-maya (дата звернення: 17.04.2024).
12. What is 3ds Max. *Autodesk.* URL: https://www.autodesk.com/eu/products/3ds-max/overview (дата звернення: 17.04.2024).
13. Ableton Live 12. *Amazon.* URL: https://www.amazon.com/Ableton-Live-Standard/dp/B0CTK2BYJT?th=1 (дата звернення: 20.04.2024).

14. FL Studio Overview. *FL Studio.* URL: https://www.image-line.com/fl-studio/ (дата звернення: 20.04.2024).
15. Wwise Documentation. *Audiokinetic.* URL: https://www.audiokinetic.com/en/library/edge/?source=Help&id=welcome_to_wwise (дата звернення: 20.04.2024).
16. Twine. *Wikipedia.* URL: https://en.wikipedia.org/wiki/Twine_(software) (дата звернення: 21.04.2024).
17. Articy: Draft 3. *Humble.* URL: https://www.humblebundle.com/store/articydraft-3 (дата звернення: 21.04.2024).
18. C#. *Microsoft.* URL: https://tinyurl.com/ycyxfmvb (дата звернення: 22.04.2024).
19. 2D Village Pack Free. *Unity AssetStore.* URL: https://assetstore.unity.com/packages/2d/environments/2d-village-pack-free-150601 (дата звернення: 01.05.2024).
20. Green tree illustration. *PNGEGG.* URL: https://www.pngegg.com/en/png-nxqwe/download (дата звернення: 15.05.2024).
21. Visual Studio Code Docs. *Visual Studio Code.* URL: https://code.visualstudio.com/docs (дата звернення: 22.04.2024).
22. Fantasy 2D Character: Free. *Unity AssetStore.* URL: https://assetstore.unity.com/packages/2d/characters/fantasy-2d-character-free-110506 (дата звернення: 04.05.2024).
23. Fantasy Wooden GUI: Free. *Unity AssetStore.* URL: https://assetstore.unity.com/packages/2d/gui/fantasy-wooden-gui-free-103811 (дата звернення: 13.05.2024).
24. Music. *OpenGameArt.* URL: https://opengameart.org/art-search-advanced?keys=&field_art_type_tid%5B%5D=12&sort_by=count&sort_order=DESC (дата звернення: 18.05.2024).

ДОДАТОК А

Лістинг скриптів

А.1 Лістинг скрипта PlayerControll

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

public class PlayerControll : MonoBehaviour
{
    [SerializeField] private float speedX = 7f;
    [SerializeField] private Animator animator;
    [SerializeField] private Transform playerModelTransform;
    [SerializeField] private AudioSource jumpSound;
    private float _horizontal = 0f;
    private bool _isFacingRight = true;
    private bool _isGround = false;
    private bool _isJump = false;
    private bool _isFinish = false;
    private bool _isLeverArm = false;
    const float speedMultiplier = 50f;
    private Finish _finish;
    private LeverArm _leverArm;
    private Rigidbody2D _rb;
    void Start(){
        _rb = GetComponent<Rigidbody2D>();
        _finish =
GameObject.FindGameObjectWithTag("Finish").GetComponent<Finish>();
```

```
        _leverArm = FindObjectOfType<LeverArm>();
    }
    void Update(){
        if (Input.GetKeyDown(KeyCode.W) && _isGround){
            _isJump = true;
            jumpSound.Play();
        }
        if (Input.GetKeyDown(KeyCode.F)){
            if (_isFinish){
                _finish.FinishLevel();
            }
            if (_isLeverArm){
                Debug.Log("Worked");
                _leverArm.ActivateLeverArm();
            }
        }
    }
    void FixedUpdate(){
        _horizontal = Input.GetAxis("Horizontal");
        animator.SetFloat("speedX", Mathf.Abs(_horizontal));
        _rb.velocity = new Vector2(_horizontal * speedX * speedMultiplier *
Time.fixedDeltaTime, _rb.velocity.y);
        if (_isJump){
            _rb.AddForce(new Vector2(0f, 450f));
            _isGround = false;
            _isJump = false;
        }
        if (_horizontal > 0f && !_isFacingRight){
            Flip();
        } else if(_horizontal < 0f && _isFacingRight){
```

```
            Flip();
        }
    }
    void Flip(){
        _isFacingRight = !_isFacingRight;
        Vector3 playerScale = playerModelTransform.localScale;
        playerScale.x *= -1;
        playerModelTransform.localScale = playerScale;
    }
    void OnCollisionEnter2D(Collision2D other){
        if (other.gameObject.CompareTag("Ground")){
            _isGround = true;
        }
    }
    private void OnTriggerEnter2D(Collider2D other){
        LeverArm leverArmTemp = other.GetComponent<LeverArm>();
        if (other.CompareTag("Finish")){
            Debug.Log("Worked");
            _isFinish = true;
        }
        if (leverArmTemp != null){
            Debug.Log("Worked");
            _isLeverArm = true;
        }
    }
    private void OnTriggerExit2D(Collider2D other){
        LeverArm leverArmTemp = other.GetComponent<LeverArm>();
        if (other.CompareTag("Finish")){
            Debug.Log("Not Worked");
            _isFinish = false;
```

```
        }
        if (leverArmTemp != null){
            Debug.Log("Not Worked");
            _isLeverArm = false;
        }
    }
}
```

А.2 Лістинг скрипта Weapon

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

    public class Weapon : MonoBehaviour{
        [SerializeField] private AudioSource enemyHitSound;
        [SerializeField] private float damage = 20f;
        private AttackController _attackController;
        private void Start() {
        _attackController = transform.root.GetComponent<AttackController>();
    }
    private void OnTriggerEnter2D(Collider2D other){
        EnemyHealth enemyHealth = other.GetComponent<EnemyHealth>();
        if (enemyHealth != null && _attackController.IsAttack){
            enemyHealth.ReduceHealth(damage);
            enemyHitSound.Play();
        }
    }
}
```

А.3 Лістинг скрипта EnemyVision

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

    public class EnemyVision : MonoBehaviour{
        [SerializeField] private GameObject currentHitObject;
        [SerializeField] private float circleRadius;
        [SerializeField] private float maxDistance;
        [SerializeField] private LayerMask layerMask;

        private EnemyController _enemyController;
        private Vector2 _origin;
        private Vector2 _direction;
        private float _currentHitDistance;
        private void Start() {
        _enemyController = GetComponent<EnemyController>();
    }
    private void Update(){
        _origin = transform.position;

        if (_enemyController.IsFacingRight){
            _direction = Vector2.right;
        } else {
            _direction = Vector2.left;
        }
        RaycastHit2D hit = Physics2D.CircleCast(_origin, circleRadius, _direction,
maxDistance, layerMask);
        if (hit){
```

```
            currentHitObject = hit.transform.gameObject;
            _currentHitDistance = hit.distance;
            if (currentHitObject.CompareTag("Player")){
                _enemyController.StartChasingPlayer();
            }
        } else {
            currentHitObject = null;
            _currentHitDistance = maxDistance;
        }
    }
    private void OnDrawGizmos(){
        Gizmos.color = Color.red;
        Gizmos.DrawLine(_origin, _origin + _direction * _currentHitDistance);
        Gizmos.DrawWireSphere(_origin + _direction * _currentHitDistance,
circleRadius);
    }
}
```