

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

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

Кафедра програмних засобів
(повне найменування кафедри)

Пояснювальна записка

до дипломного проєкту (роботи)

бакалавр
(ступінь вищої освіти)

на тему ПРОГРАМНА РЕАЛІЗАЦІЯ ВІДЕОГРИ
В ЖАНРІ МЕТРОЇДВАНІЯ З ВИКОРИСТАННЯМ РУШІЯ UNITY
SOFTWARE IMPLEMENTATION OF A METROIDVANIA VIDEO GAME
USING THE UNITY ENGINE

Виконав: студент(ка) 4 курсу, групи КНТ-227
Спеціальності 122 Комп'ютерні науки та
технології
(код і найменування спеціальності)

Освітня програма (спеціалізація)
Комп'ютерні науки та технології

Кузнецов В.О.
(прізвище та ініціали)

Керівник Зеленьова І.Я.
(прізвище та ініціали)

Рецензент Терещенко Е.В.
(прізвище та ініціали)

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

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

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.

С.О. Субботін
 “ ” 20 року

З А В Д А Н Н Я

НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

Кузнецова Віталія Олександровича

(прізвище, ім'я, по батькові)

1. Тема проєкту (роботи) Програмна реалізація відеогри в жанрі метроїдванія з використанням рушія Unity

Software Implementation of a Metroidvania Video Game Using the Unity Engine

керівник проєкту (роботи) Зеленьова Ірина Яківна, к.т.н., доцент,
 (прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом закладу вищої освіти від “30” березня 2021 року № 103

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

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

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Огляд аналогів та теоретичні положення розробки відеоігор. 2. Аналіз інструментів для розробки. 3. Розробка програмної частини системи. Висновки. Перелік джерел посилань. Додатки.

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

6. Консультанти розділів проєкту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-3 Основна частина	Зеленьова І.Я. доцент		
Нормоконтроль	Андреев М.О., асистент		

7. Дата видачі завдання “ 15 ” березня 2021 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Постановка завдання роботи	1 тиждень	Завдання, ТЗ
2	Огляд аналогів та теоретичні положення розробки відеоігор	2-3 тижні	Розділ 1
3	Аналіз інструментів для розробки	4 тиждень	Розділ 2
4	Розробка програмної частини системи	5-8 тижні	Розділ 3
5	Оформлення пояснювальної записки та документів до неї. Нормоконтроль та рецензування	9 тиждень	Додатки
6	Захист роботи	10 тиждень	

Студент(ка)

_____ Кузнецов В.О.
(підпис) (прізвище та ініціали)

Керівник проєкту (роботи)

_____ Зеленцова І.Я.
(підпис) (прізвище та ініціали)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи бакалавра:
76 с., 4 табл., 32 рис., 2 дод., 25 джерел.

ІНТЕРАКТИВНА ГРА, КОМП'ЮТЕРНА ПРОГРАМА,
МЕТРОЇДВАННЯ, ПЛАТФОРМЕР, UNITY ENGINE.

Об'єкт дослідження – індустрія відеоігор.

Предмет дослідження – методи реалізації відеогри в жанрі метроїдванія.

Мета роботи – створення та тестування гри в жанрі метроїдванія на базі рушія Unity.

Матеріали, методи та технічні засоби: об'єктно-орієнтоване програмування, ігровий рушій Unity 2D Engine, графічний редактор Aseprite, ICP Microsoft Visual Studio Code, мова програмування C#, персональний комп'ютер з процесором Intel Core i7 4770k, 8 Гб оперативної пам'яті, 21'' монітором, клавіатурою, веб-камерою та маніпулятором типу «миша» під керуванням операційної системи Microsoft Windows 7.

Результати. Досліджено теоретичні положення розробки відеоігор та створено кросплатформенну відеогру в жанрі метроїдванія.

Висновки. В процесі створення дипломного проекту мною було отримано багато достатньо цінного досвіду в усіх галузях розробки відеоігор.

Галузь використання – індустрія розваг.

ABSTRACT

Explanatory note to the final qualifying work of the bachelor: 76 pages, 32 figures, 4 tables, 2 appendixes, 25 sources.

COMPUTER GAME, INTERACTIVE GAME, METROIDVANIA, PLATFORMER, UNITY ENGINE

Object of study is a videogame industry.

Subject of study are methods of realization of a metroidvania videogame.

The purpose of work is to create and test metroidvania videogame based on Unity Engine.

Materials, methods and tools: object-orientated programming, game engine Unity 2D Engine, Aseprite graphic editor, integrated development environment Microsoft Visual Studio Code, personal computer with Intel Core i7 4770k processor, 8 Gb of RAM, 21'' monitor, keyboard and manipulator type «mouse», which operates under the operating system Microsoft Windows 7.

Results. The theoretical provisions of video game development have been studied and a cross-platform video game in the genre of metroidvania has been created.

Conclusions. In the process of creating a thesis project, I gained a lot of valuable experience in all areas of video game development.

Field of use – entertainment industry.

ЗМІСТ

С.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ.....	8
ВСТУП.....	9
1 ОГЛЯД АНАЛОГІВ ТА ТЕОРЕТИЧНІ ПОЛОЖЕННЯ РОЗРОБКИ	
ВІДЕОІГОР	11
1.1 Аналогічні проекти в відеоігровій індустрії	11
1.1.1 Castlevania: Symphony of the Night	11
1.1.2 Dark Souls	12
1.1.3 Hollow Knight.....	14
1.1.4 Blasphemous	15
1.2 Основні теоретичні положення розробки відеоігор	16
1.3 Висновки за розділом 1	20
2 АНАЛІЗ ІНСТРУМЕНТІВ ДЛЯ РОЗРОБКИ	22
2.1 Програмні засоби для розробки відеоігор	22
2.1.1 Ігровий рушій	22
2.1.2 Інтегроване середовище розробки	25
2.1.3 Графічний редактор	26
2.1.4 Інші програми та плагіни	27
2.2 Рекомендовані апаратні засоби.....	27
2.3 Постановка задачі розробки та дослідження.....	29
2.4 Висновки за розділом 2	29
3 РОЗРОБКА ПРОГРАМНОЇ ЧАСТИНИ СИСТЕМИ.....	31
3.1 Загальна концепція відеогри	31
3.1.1 Начерки сюжету гри	33
3.1.2 Візуальний стиль	34
3.1.3 Основні механіки	35
3.2 Написання скриптів для реалізації ігрових механік	36
3.3 Опрацювання графічної частини	43
3.4 Левел-дизайн (побудова локацій).....	50

3.5 Тестування готової гри	51
ВИСНОВКИ.....	56
ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ.....	57
ДОДАТОК А Текст програми.....	60
ДОДАТОК Б Слайди презентації	71

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

IDE — Integrated Development Environment;

ESRB — Entertainment Software Rating Board;

HP – Health Point;

SSD — Solid-state Drive;

RAM — Random Access Memory;

ПЗ — програмне забезпечення;

ЕОМ — електронно-обчислювальна машина;

ПК — персональний комп'ютер.

ВСТУП

Перші ЕОМ були створені в середині ХХ століття, та вже мали основні властивості сучасної обчислювальної техніки, але вони відносилися до спеціалізованих пристроїв, що можуть виконувати лише обчислювальну функцію. Згодом, відповідно до вимог суспільства того часу, з'явилися перші електронні та відеоігри. Це було зумовлено тим, що у ті роки комп'ютерні технології не були доступними простому населенню, та, як вже зазначалось раніше, виконували лише функцію обчислення. Надалі, починаючи з 80-х років минулого століття, коли персональні комп'ютери стали доступні цивільному населенню, відеоігри, що відтепер виходять на ПК, стали класифікувати як комп'ютерні ігри.

Науково-технічний прогрес не стоїть на місці, та з плином часу розробники відео- та комп'ютерних ігор почали стикатися з вдосконаленням технологій, збільшенням потужності комплектуючих ПК, що призвело до підвищення очікувань та збільшення середнього віку гравців. Також, до підвищення середнього віку покупців відеоігор призвело введення в обіг у 1994 році вікових рейтингів ESRB [1], запроваджених компанією Interactive Digital Software Association (сучасна назва — Entertainment Software Association). У книзі «Blood, sweat and pixels» [2] за авторством відеоігрового журналіста Джейсона Шраєра головний виконавчий директор компанії Obsidian Entertainment Фергюс Уркарт відзначає, що «розробляти ігри – це приблизно як знімати кіно, але щоразу перед зйомками нового фільму вам потрібно збирати цілком нову камеру».

Перед написанням дипломної роботи було вирішено, що об'єктом розробки та дослідження буде гра в жанрі платформер-метроїдванія.

За різними аспектами відеоігри можна класифікувати по-різному, але найважливішим з них є жанр [3], перелік основних жанрів наведено в таблиці 1.

Таблиця 1 — Основні жанри відеоігор

Назва жанру	Короткий опис	Назви представників жанру
Шутер	Основою ігрового процесу є стрільба по вороже налаштованих цілях	Tom Clancy's Rainbow Six, ARMA, Counter Strike, DOOM
Файтинг	Основою ігрового процесу є рукопашний поєдинок між двома суперниками	Mortal Kombat, Tekken, Guilty Gear
Платформер	Основою ігрового процесу є стрибки по різноманітних платформах (від цього й походить назва) та подолання перешкод	Super Mario, Metroid, Hollow Knight, Ori and the blind forest
Стратегія	Жанр ігор, в якому від гравця вимагають проявляти навички стратегічного планування, менеджменту ресурсів, тощо	Sid Meyer's Civilization, Warhammer 40000 Dawn of War, Total War
Симулятор	Основою жанру є якомога точніше відтворення фізичних законів реального світу, властивостей реальних об'єктів, процесів, тощо	Microsoft Flight Simulator, FIFA, The Sims
Рольова гра (RPG)	Основна частина ігрового процесу полягає в керуванні персонажем, чи групою персонажів, які досліджують ігровий світ	Baldur's Gate, Dragon Age: Origins, Star Wars Knights of the Old Republic
Пригодницька гра (Квест)	Ігри цього жанру можуть варіюватися від детективних головоломок до інтерактивної літератури	The life and suffering of Sir Brante, Machinarium, Siberia, Myst

1 ОГЛЯД АНАЛОГІВ ТА ТЕОРЕТИЧНІ ПОЛОЖЕННЯ РОЗРОБКИ ВІДЕОІГОР

1.1 Аналогічні проекти в відеоігровій індустрії

Перед оглядом аналогів потрібно дати визначення терміну «метроїдванія» [4] та визначити, сукупність яких ознак вирізняє метроїдванії на фоні інших ігор. Метроїдванія - концепція жанру комп'ютерних ігор, піджанр пригодницького бойовика, з набором ігрових механік и ігровим процесом, схожим на той, що був у франшизах Metroid [5] и Castlevania [6] (починаючи з Castlevania: Symphony of the Night). Основними ознаками метроїдваній є великий світ зі зв'язаними між собою локаціями, однак доступ до деяких частин перекритий різними бар'єрами, серед яких можуть бути: недолік інформації, відсутність у інвентарі персонажа потрібного ключа або тимчасова неспроможність досягти окремих місць без спеціальних вмінь; і заради подолання цих перешкод гравець має досліджувати світ. Також необхідно зазначити, що більшість метроїдваній мають похмуру атмосферу та серйозний, неоднозначний сюжет. Також, за рідкісним виключенням, метроїдванії являють собою двовимірні платформери. Далі піде перелік проектів в відеоігровій індустрії, які є представниками визначеного жанру.

1.1.1 Castlevania: Symphony of the Night

Castlevania: Symphony of the Night [7] прийнято вважати одним з основоположників жанру метроїдванія, наряду з Super Metroid. Своєю знаковістю гра зобов'язана своєму провідному геймдизайнеру Кодзі Ігарасі, який, задля збільшення тривалості та більшої зручності гри для новачків, вирішив об'єднати ігровий процес класичних платформерів з рольовими елементами у вигляді характеристик, які можна покращувати, у головного героя, та з елементами іншої популярної у той час гри – The Legend of Zelda, зокрема звідти був взятий відкритий світ з можливістю його дослідження.

За своєю стилістикою світ Castlevania являє собою продовження всесвітньо відомого роману «Дракула» за авторством Брема Стокера. Протягом більшої частини ігрового процесу під контролем гравця буде дампір (наполовину вампір) Алукард, котрий намагатиметься дістатися до винуватця усіх подій, самого графа Дракули, водночас досліджуючи внутрішні приміщення замку Кастльванія (рисунок 1.1). Кожній з локацій гри притаманні різні типи ворогів, кожний з яких також відсилає нас до різних фольклорних істот як-от зомбі, мумії, перевертні та інші. Також, як і у більшості подібних ігор, у кожній локації є свій бос. У відеоігровій індустрії боси – це тип супротивників, що відрізняються, як правило, своєю силою, кількістю HP, або якимись іншими особливостями. Зазвичай босів розташовують наприкінці рівнів, або перед важливими в структурі гри предметами.



Рисунок 1.1 – Карта приміщень замку

1.1.2 Dark Souls

Dark Souls [8] стала одним з найвідоміших виключень з правил, та однією з небагатьох ігор, які перенесли концепцію метроїдванії у тривимірний

простір. Створена фактично с з нуля японським геймдизайнером Хідетака Міядзакі, перша частина серії Dark Souls відповідала майже усім вимогам, щоб називатися класичною метроїдванією: в ній був відкритий світ, що поділявся на 34 локації, прохід до деяких головному герою перегороджували боси. Окремо варто виділити складність гри, яка стала еталоном рівня складності в іграх, а напис «YOU DIED» став найбільш впізнаваною ознакою серії Souls (рисунок 1.2). From Software, компанія-розробник гри, змогла домогтися цього ефекту завдяки контрасту між їх проектом, і тим, що випускали інші розробники у той час.



Рисунок 1.2 – Напис «YOU DIED», який бачить гравець коли кількість його НР доходить до нуля

В пізніх 2000-х та ранніх 2010-х більшість розробників намагалась зробити максимально прості ігри, які не вимагають від гравця майже ніяких зусиль, та від цього страждала й комплексність ігор. Хідетака Міядзакі вирішив не йти на поводу видавців, та зробив, на перший погляд, максимально недружню до гравця атмосферу, як в плані ігрового процесу, так і в плані історії світу гри, яку автори подавали гравцеві через описи предметів та діалоги

з персонажами, але ніяк не шляхом видовищних відеороликів. Зокрема, саме вихід Dark Souls у 2011 році дав імпульс для популяризації складних, недружніх до гравця ігор, а також розробників, у тому числі й незалежних, для створення своїх метроїдваній, бо тепер середньостатистичному гравцю хотілося не просто якось заповнити свій вільний час, але й подолати рівних йому, а часом і сильніших за нього, суперників та дослідити можливо й не дуже великий, але по-справжньому відкритий та деталізований світ.

1.1.3 Hollow Knight

Саме на хвилі популярності ігор, схожих на Dark Souls, і вийшла гра Hollow Knight [9] (рисунок 1.3). З точки зору сукупності ігрових механік перед нами знаходиться справжня класична метроїдванія, яка, однак, перейняла метод подачі сюжету від Dark Souls.



Рисунок 1.3 – Графічне оформлення Hollow Knight

Сам сюжет оповідає нам історію самотнього жука, який носить титул Порожній Лицар, та подорожує королівством Халлоунест у пошуках свого

короля, а також способу побороти інфекцію, котра зводить з розуму інших комах. Серед інших подібних проєктів Hollow Knight виділяє унікальний графічний стиль атмосферний музичний супровід, та унікальну можливість відчувати себе дослідником невідомого досі світу.

1.1.4 Blasphemous

З усіх приведених ігор Blasphemous [10] є найбільш схожою на Dark Souls у візуальному та сюжетному плані. The Game Kitchen, студія-розробник гри, черпали натхнення в історії свого рідного міста – Севільї, що була центром діяльності іспанської інквізиції. Саме цьому неймовірно похмурому та жорстокому, але безумовно колоритному періоду в історії Blasphemous зобов'язана своєю насиченою атмосферою. Майже все в цій грі тим чи іншим образом нагадує гравцеві або про біблійні історії, або про легенди тих часів, або навіть про класичні витвори мистецтва (рисунок 1.4).



Рисунок 1.4 – Зовнішній вигляд одного з босів гри, при створенні якого автори спиралися на скульптуру «П'єта» за авторством Мікеланджело Буонарроті

1.2 Основні теоретичні положення розробки відеоігор

Розробка відеогри [11] – комплексний процес, в якому зазвичай приймає участь достатньо великий колектив, однак, є й невеликі студії, а над деякими проектами може працювати лише одна людина. В великих ігрових студіях можна виділити 13 посад, які займають працівники студії, але кількість цих самих посад, може варіюватися в залежності від потреб окремих проектів, або від кількості працівників в компанії (таблиця 1.1).

Варто розуміти, що розробка відеоігор – комплексний та структурований процес, у якому можна виділити чотири основних етапи: проектування, виробництво, випуск і продажі та підтримка. Кожен з них зараз розглянемо окремо.

Проектування, також відоме як пре-продакшн – починається з подання ідеї кимось з працівників, зазвичай цей працівник і стає керівником проекту. Далі, шляхом мозкового штурму, колектив вирішує які у гри будуть ключові особливості, створюються начерки основного сюжету, тощо. Виходячи з цієї інформації про майбутню гру, що є у студії, обирається жанр майбутньої гри. Варто зазначити, що жанр, або навіть концепція гри в цілому може змінюватися в процесі розробки. Геймдизайнер вирішує, як приблизно виглядатимуть персонажі та ігровий світ, та, спираючись на це, створює концепт-арти. Зазвичай, в цей час гра вже отримує попередню назву, яка, однак, може бути змінена в процесі розробки. Після цього, якщо мова йде про велику студію, збирається команда програмістів, художників, сценаристів, композиторів, менеджерів, тощо. Колективом визначаються строки виконання кожного з видів роботи, приблизна дата виходу гри та кількість коштів, що знадобляться для цього. Частіше за все, на цьому етапі в команди розробників вже є концепт-документ, дизайн-документ та прототип. В концепт-документі обґрунтовується чому буде вигідно створити цю гру. Дизайн-документ (у спільноті розробників прийнято вживати скорочення «диздок») описує гру в цілому: її початковий сюжет, ігровий процес, та містить в собі перші концепт-арти. Прототипом

прийнято називати початкову просту версію гри, яка відображає задуманий розробниками ігровий процес.

Таблиця 1.1 – Посади в компаніях, що спеціалізуються на розробці відеоігор

Назва посади	Обов'язки працівника
Геймдизайнер	Придумує основну концепцію гри, проектує основні ігрові механіки, образи персонажів. Геймдизайнерів може бути кілька, з-поміж яких один – провідний (головний), а інші відповідають за окремі аспекти гри
Менеджер	Координує роботу людей, залучених до розробки. Планує витрати, розподіляє обов'язки працівників
Програміст	Працює з програмною частиною гри: пише і коригує код
Художник	Відповідає за графічний аспект гри
Звукорежисер	Відповідає за обробку звукової частини гри
Композитор	Пише музику для гри, котра виконується найнятим музичним колективом
Звукоінженер	Знаходить/записує/синтезує звукові ефекти
Сценарист	Відповідає за текстовий аспект гри: основний сценарій, діалоги, описи предметів
Актор	Приймає участь в зйомці відеороликів та захопленні рухів для персонажів
Актор озвучування	Озвучує мову персонажів, закадровий голос
Дизайнер рівнів	Проектує локації гри, розставляє готові об'єкти та персонажів, створює сюжетні події
Піар-менеджер	Організовує презентації, керує рекламною кампанією
Тестер	Грає в різні версії гри на стадії розробки, документує та по можливості виправляє помилки

Виробництво є найбільш комплексним та обширним етапом, який, в свою чергу, поділяється ще на 4 стадії:

— зважаючи те, що будь-яка відеогра це, в першу чергу, комп'ютерна програма, варто розуміти, що її робота, ігровий процес, технічні можливості та контент в цілому забезпечуються програмним кодом. Здебільшого сучасні ігри базуються на вже готових ігрових рушіях, функціонал яких програмісти просто розширюють для реалізації свої задумів, але деякі

студії заради унікальних особливостей своїх ігор створюють свої власні рушії для своїх же потреб. Якщо розглядати вже існуючі рушії, варто зазначити, що вони відрізняються за типом ліцензування: в той час, як деякі рушії є безкоштовними, інші потребують покупки ліцензії, яка, в свою чергу, відрізняється в залежності від умов використання, тобто невеликій незалежній студії купувати ліцензію на вже готовий рушій може бути значно вигідніше, ніж для великих компаній. Також варто відзначити, що ігрові рушії поділяються за своєю специфікою, та у той час, поки одні рушії є універсальними, інші були створені для розробки ігор конкретного жанру. Цей же принцип працює і у випадку з ігровими платформами. Зазвичай, ігрові рушії обробляють код, написаний на загально прийнятих мовах програмування, як-от C#, C++ або JavaScript, але існують спеціальні програми, для роботи з якими не потребуються навички в програмуванні. Прикладами таких програм є GameMaker Studio, або RPG Maker;

— відеогра є комплексним об'єктом, та тому її розробка передбачає створення не тільки програмного коду, але й графіки, звукового оформлення та, за рідкісним винятком, написання сюжету. При створенні графіки художники спираються на створенні під час проектування концепт-арти, на основі яких у програмах для роботи з графікою створюються двовимірні або тривимірні моделі декорацій, персонажів та предметів. Щоб зробити моделям анімації, їх експортують до інших програм, де вже створюються набори рухів, які відтворюватимуться в залежності від дій гравця. У двовимірній графіці анімації являють собою набір спрайтів, де кожне зображення є окремим кадром. У наш час все частіше для створення реалістичних анімацій використовують технологію захоплення руху. Не менш важливою частиною графічного оформлення є візуальні ефекти, які задають стиль гри та роблять її більш видовищною. Деякі з них створені для надання зображенню реалістичності, як-от гра світлотіні, симуляція погодних умов, тощо. У той же час є й ефекти, які, навпаки, додають зображенню ефект коміксу, або кіноплівки. Інколи ігри містять відеовставки, так звані кат-сцени, які можуть бути як створені у

програмах для анімації, так і зняті з живими акторами і декораціями; При створенні звукового оформлення записують як музику та голоси акторів озвучування, так й звукові ефекти кроків, пострілів, ударів, тощо. Деякі з цих звуків можуть обиратися з безкоштовних звукових бібліотек, а деякі записуються окремо. Також варто зазначити, що є цілий ряд композиторів, які спеціалізуються на створенні музики для відеоігор. Якщо розробники вирішили, що гра буде сюжетно-орієнтованою, то до роботи приступають сценаристи, які пишуть сюжет, діалоги між персонажами, описи предметів та інший внутрішньоігровий текст;

— майже весь ігровий процес визначається ігровою механікою. Базисом ігрової механіки є персонажі та ігровий світ, з яким ці персонажі взаємодіють тим чи іншим образом. Також частиною ігрової механіки є управління, за допомогою якого гравець може керувати ігровим персонажем. Також на етапі розробки створюється інтерфейс користувача, який повідомляє гравцеві важливу інформацію про персонажа та оточуючий його світ. Деякі об'єкти у світі гри взаємодіють між собою без участі гравця, і за цей тип взаємодій відповідає фізичний рушій. Наприклад, до такого типу взаємодії відноситься сила гравітації, інерція, поведінка рідин, тощо. Зазвичай ігровий світ поділяється на окремі локації. Це робиться задля того, щоб знизити навантаження на апаратну частину комп'ютера. Але сучасні ігри розробляють так, щоб цей поділ не був помітним для гравця задля створення ілюзії відкритого світу. Дизайнер рівнів на цьому етапі розташовує об'єкти в ігровому світі. Грамотне komponування рівнів та об'єктів забезпечує гравця інтересом до гри;

— після завершення усіх попередніх етапів починається остаточне доробка гри, або полішинг. Версія гри, що не готова до кінця, але вже містить всі робочі механіки, називається альфа-версією. Вона може містити значні помилки, виявленням, та по можливості, виправленням яких займаються тестери. Частіше за все саме на цьому етапі розробники записують трейлери своїх ігор, даючи уявлення про свою гру потенційним покупцям. Пізніше,

розробники випускають бета-версію, до тестування якої вже можуть залучатися й прості гравці. На обох стадіях можуть вноситися зміни ігрового балансу, графічного та звукового оформлення, тощо.

Випуск і продажі зазвичай контролюються видавцем гри, який виплачує розробникові його частку від отриманих коштів. Однак це не стосується компаній, які є одночасно і розробниками і видавцями. У таких ситуаціях прибуток розподіляється згідно з правилами, встановленими в компанії. До обов'язків видавця також входять зв'язки з користувачами та піар-кампанія. Також видавець займається локалізацією гри, тобто перекладом тексту та голосів на інші мови а також адаптує її до законів країни, де гра видаватиметься. Закінчена та локалізована гра записується на фізичні носії, або розповсюджується по інтернету завдяки сервісам цифрової дистрибуції, як-от Steam, GOG або Origin.

Підтримка гри являє собою виправлення певних помилок, які не були помітні до випуску, наприклад, деякі гравці можуть бути незадоволені складністю гри. У цьому випадку розробник видасть патч, який виправляє певні помилки та якимось чином покращує гру. Розробка патчів не принесе компанії прибутку, але збільшить довіру цільової аудиторії до розробників. Крім того, для підвищення інтересу до гри розробники іноді випускають DLC (downloadable content), деякі з яких розширюють гру за змістом, а інші - суто декоративні.

1.3 Висновки за розділом 1

В першому розділі було проаналізовано найвідоміші ігри в жанрі метроїдванія та були розглянуті основні теоретичні положення, що стосуються розробки відеоігор. Перш за все, варто відмітити, що більшість представників жанру мають схожу похмуру атмосферу та не завжди очевидний сюжет, а також комплексний та ретельно вивірений ігровий процес. Також було встановлено, що розробка відеогри являє собою не просто малювання моделей

або спрайтів та написання коду, але сукупність процесів, що може зайняти велику кількість часу та сил. Крім того, була підтверджена актуальність цієї роботи, бо в наш час відеоігри стали невід’ємною частиною індустрії розваг та культури в цілому.

В результаті було сформульовано які саме риси має мати гра, щоб її можна було віднести до жанру метроїдванія, та які задачі треба виконати під час розробки проекту.

Мета цієї дипломної роботи полягає в створенні та випробовуванні власної гри в жанрі метроїдванія на базі ігрового рушія Unity Engine. Також, шляхом аналізу існуючих проектів в цьому жанрі, що був виконаний в підрозділі 1.1, було доведено доцільність створення подібного проекту.

2 АНАЛІЗ ІНСТРУМЕНТІВ ДЛЯ РОЗРОБКИ

2.1 Програмні засоби для розробки відеоігор

Програмними засобами або програмним забезпеченням називають сукупність програм і програмних документів, що необхідні для експлуатації цих програм, або, якщо казати простіше, будь-яка комп'ютерна програма є програмним забезпеченням. За своїми властивостями програмне забезпечення поділяють на:

- системне, обслуговує власні потреби комп'ютера;
- інструментальне, автоматизує процес розробки нових програм та їх супровід за допомогою мови програмування;
- прикладне, яке призначено для розв'язання тих чи інших задач користувача.

За своє суттю, всі інструменти для створення комп'ютерних програм відносяться саме до прикладного програмного забезпечення. В основному при розробці ігор використовуються:

- ігровий рушій [12];
- інтегроване середовище розробки [13];
- графічний редактор [14].

Також варто зазначити, що ПЗ можна розділити на корисне та шкідливе. Їх різниця полягає у тому, що корисне забезпечення відповідає вимогам користувача та виконує поставлені ним задачі, у той час як шкідливе створюється з метою завдання шкоди комп'ютеру або збору конфіденційної інформації.

2.1.1 Ігровий рушій

Перш за все, основним програмним засобом для розробки гри є ігровий рушій. Його основна функція полягає у забезпеченні функціональності гри шляхом роботи з її технічним аспектом та систематизації її внутрішньої

структури. Як і сама відеогра, «движок» є комплексним програмним забезпеченням, що складається з декількох елементів, серед яких можна виділити чотири основних: рушій рендерингу (графічний рушій, або «інструмент візуалізації»), фізичний рушій, звуковий рушій та ігровий штучний інтелект. Як вже було зазначено у попередньому розділі, при створенні ігор більшість компаній замість розробки рушія з нуля користуються вже готовими, що дозволяє заощадити час та ресурси. Сам термін «ігровий рушій» з'явився у середині 1990-х років, зокрема у зв'язку з появою 3D-ігор. Першою компанією, чий рушій почали використовувати поза нею, стала id Software з її легендарними іграми DOOM (рисунок 2.1) та Quake. Їх програмна частина виявилася настільки досконалою для того часу, що розробники з інших студій ліцензували окремі її компоненти та на цій базі за допомогою створеного власними силами контенту розробляли рівні, або навіть цілі ігри. Наприклад, рушій гри Quake послужив основою для понад десятиох проектів, та став серйозною віхою у розвитку ігрової індустрії.



Рисунок 2.1 – Скріншот з гри DOOM (1993)

Варто зазначити, що з плином часу змінився й підхід до розробки ігор: якщо раніше через примітивність графіки та музики у студії могло взагалі не бути художника, то тепер у команді розробників зазвичай є однакова кількість людей, що відповідають за графіку, та людей відповідальних за код. На даний момент ігрові рушії – мало не найскладніші у створенні програмні засоби, які зазвичай містять в собі багато різноманітних компонентів, що можна індивідуально налаштовувати в залежності від потреб гри. Для дипломного проекту виходячи з цілої низки причин було обрано рушій Unity [15]. Першою та найголовнішою перевагою цього програмного забезпечення є кросплатформенність. Взаємодіяти з програмою-редактором можна на Windows, macOS та Linux, у той час як білди ігор, створених за її допомогою, можуть працювати на 25 платформах. Друга перевага Unity – це те, що мовою програмування для написання внутрішньоігрової логіки є C#, одна з найвідоміших та, на мій погляд найзручніших об'єктно-орієнтованих мов програмування. Ще одною суб'єктивною перевагою Unity є зручність інтерфейсу, який складається з різних вікон, кожне з яких можна розташовувати на свій розсуд (рисунк 2.2).

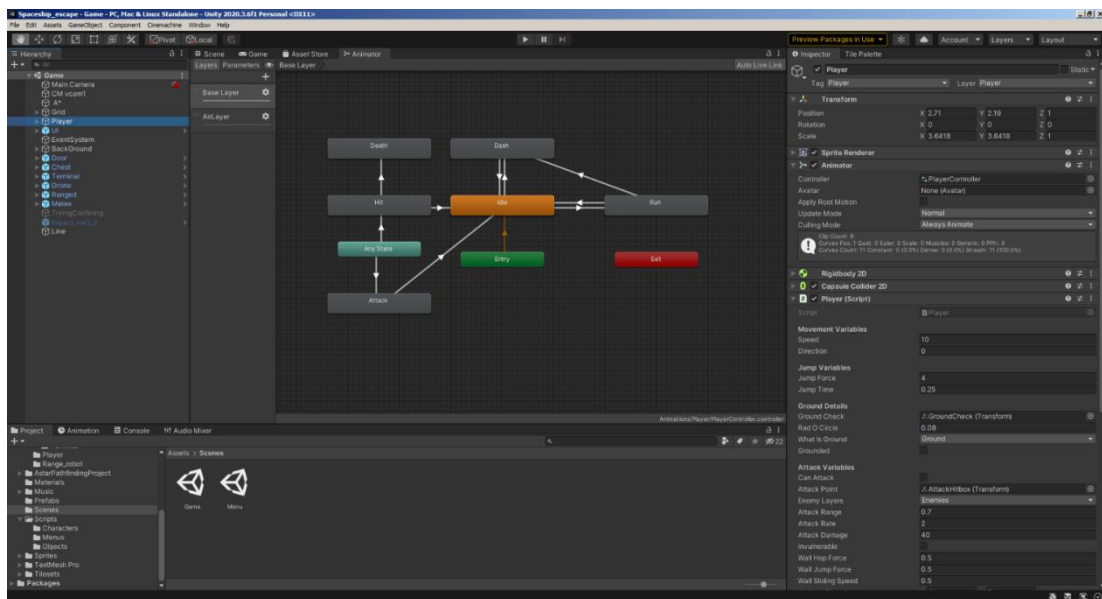


Рисунок 2.2 – Інтерфейс редактора Unity

2.1.2 Інтегроване середовище розробки

Інтегроване середовище розробки, або IDE, являє собою систему програмних рішень для написання коду при створенні програмного забезпечення. Основними інструментами в IDE є редактор початкового коду, та набору інструментів для автоматизації побудови та налагодження програм. Варто зазначити, що деякі програми такого типу можуть мати різний функціонал: у деяких середовищ може не бути компілятора або інтерпретатора, або навіть обох з них, у той час як інші містять обидва ці елементи. Головною задачею ІСР можна назвати підвищення ефективності роботи програміста шляхом надання йому єдиної програми, що матиме весь необхідний функціонал, замість цілого комплексу інструментів, кожен з яких відповідатиме за різні задачі. При виборі IDE для дипломного проекту дуже важливим критерієм була наявність у ній інструментів для написання об'єктно-орієнтованого програмного забезпечення. Не менш важливими були ціна самої ІСР та можливість працювати з системою керування версіями Git. Виходячи з цих умов було вирішено працювати з Microsoft Visual Studio Code [16] (рисунок 2.3).

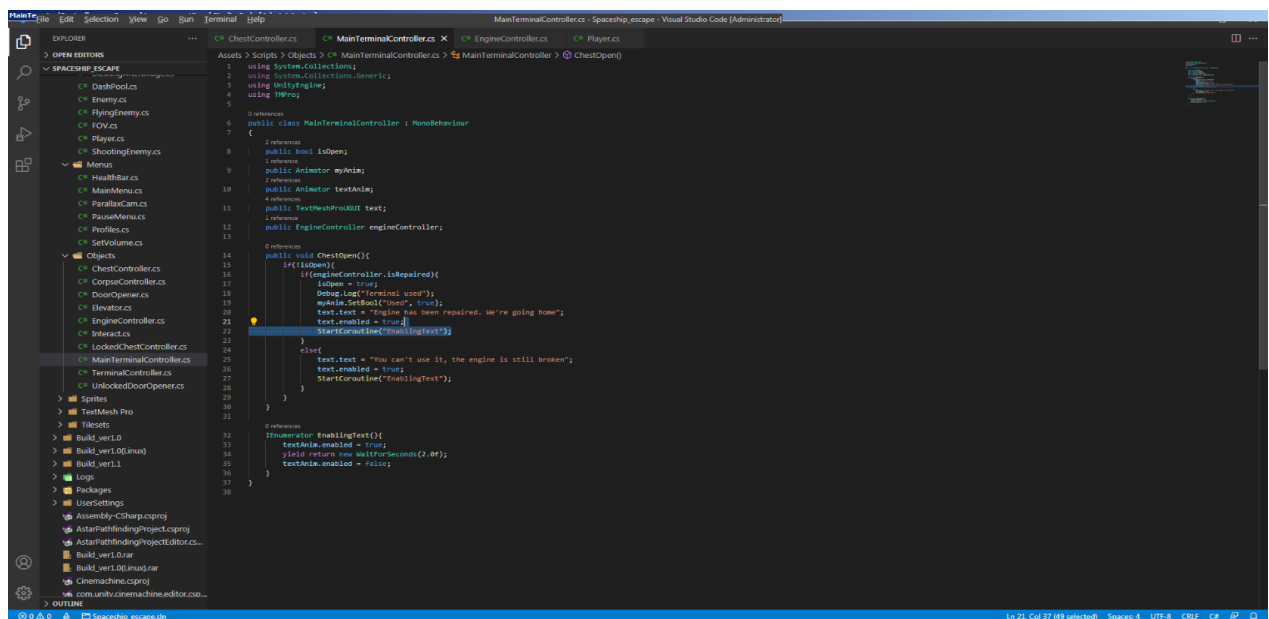


Рисунок 2.3 – Інтерфейс Microsoft Visual Studio Code

2.1.3 Графічний редактор

Графічним редактором є комп'ютерна програма (або пакет програмного забезпечення) за допомогою якої можна створювати або редагувати зображення та зберігати їх на носіях інформації. Використовувати графічні редактори можна як для редагування готових фотографій, так і для створення власних зображень з нуля. Усього існує чотири типи редакторів:

- растрові графічні редактори існують для роботи з растровою графікою, яка характеризує зображення як масив пікселів (кольорових точок);
- векторні графічні редактори працюють з геометричними примітивами, як-от точки, лінії, полігони, тощо, на відміну від растрових редакторів, що працюють з масивом пікселів;
- гібридні графічні редактори забезпечують можливість роботи одночасно і з векторною і з растровою графікою та перетворювати зображення одного типу в інший завдяки сукупності спільних інструментів;
- тривимірні графічні редактори є прикладними програмами, що дозволяють користувачеві створювати та обробляти 3D-зображення.

При розробці цього проекту використовувався вузькоспеціалізований растровий графічний редактор Aseprite [17] для роботи з піксель-артом (рисунок 2.4).

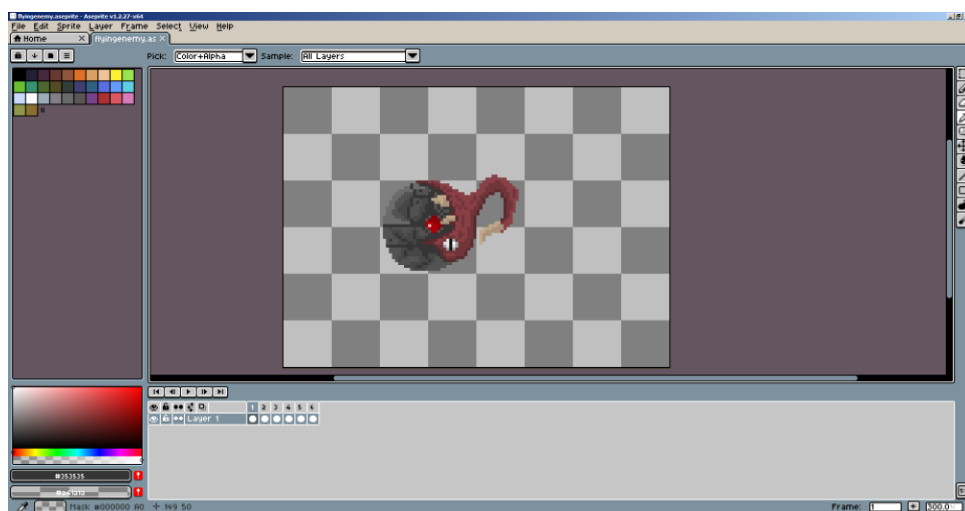


Рисунок 2.4 – Інтерфейс графічного редактора Aseprite

2.1.4 Інші програми та плагіни

Окрім основних програм, для розробки цієї гри використовувались два плагіни (розширення) для Unity: Cinemachine та A* Pathfinding Project, що доступні в Unity Asset Store.

Основною функцією плагіну, або, якщо точніше, розширення, Cinemachine є взяття під контроль усієї логіки поведінки камер, що дозволяє дуже сильно зекономити час та сконцентруватися на інших аспектах гри.

A* Pathfinding Project, у свою чергу, є плагіном, що відповідає за знаходження шляху за допомогою алгоритму A* (A star).

2.2 Рекомендовані апаратні засоби

Перш за все, потрібно дати визначення терміну «апаратний засіб», або «апаратне забезпечення». Апаратним забезпеченням, або «залізом» називають електронно-механічну складову ЕОМ, як-от: компоненти живлення, електронні схеми, пристрої вводу-виводу, пасивні компоненти, тощо. Якщо мова йде про персональний комп'ютер, то до необхідних для його роботи апаратних засобів відносяться: блок живлення, материнська плата, центральний процесор, оперативна пам'ять (RAM), карти розширення (відеокарти, аудіокарти, мережеві адаптери), жорсткий диск (або інколи SSD), прилади вводу (зазвичай це клавіатура та комп'ютерна миша) та виводу (монітор).

Апаратним забезпеченням для розробки відеоігор, по суті, є будь-який комп'ютер, який містить усю апаратну складову, та на якому встановлені усі вищезазначені типи програм, але, так чи інакше, кожна з них має свої системні вимоги, що необхідні для роботи. Найменш вибагливим з цього списку виявився Aseprite, для роботи з яким всього лише потрібно 128 мегабайтів оперативної пам'яті та 40 мегабайтів вільного місця на жорсткому диску. Другою за вибагливістю програмою виявилася Microsoft Visual Studio Code, для якої потрібно близько 200 мегабайтів вільної пам'яті на диску, CPU з тактовою

частотою не менше ніж 1.6 гігагерц та один гігабайт оперативної пам'яті. У свою чергу, Unity Editor не має якихось детальних специфікацій, окрім 64-розрядної операційної системи, наявності центрального процесора з архітектурою x64 та підтримкою набору інструкцій SSE2 та GPU з підтримкою DirectX версії 10 та вище. Але, завдяки пошукам на офіційному форумі Unity мені вдалося знайти пост від користувача з нікнеймом Joe-Censored [18], у якому він зазначає, що мінімальні вимоги для запуску редактора є дуже низькими, але для комфортної роботи з проектами потрібен приблизно такий набір комплектуючих:

- процесор компанії Intel i7 четвертого чи вище покоління, або його еквівалент від компанії AMD;
- не менше, ніж 16 гігабайтів оперативної пам'яті;
- SSD об'ємом не менш, ніж 512 гігабайтів (але, варто відмітити, що жорсткого диска з досить високими показниками запису та зчитування даних буде достатньо);
- відеокарта на базі графічного процесора GeForce 1060 або краще, або її еквівалент від AMD (варто зазначити, що з поставленими задачами в процесі роботи дуже добре справлялася відеокарта на базі GeForce 1050ti);
- монітор з розподільною здатністю 1920x1080, чи вище.

Для роботи над дипломним проектом мною була використана наступна конфігурація ПК:

- процесор Intel Core i7 4770k;
- 16 гігабайтів RAM;
- відеокарта ASUS GeForce 1050ti;
- SSD від компанії Kingston об'ємом 480 гігабайтів.

Внаслідок цього при роботі з необхідним ПЗ проблем у розробці гри не виникало.

2.3 Постановка задачі розробки та дослідження

Завданням цієї бакалаврської дипломної роботи є розробка відеогри в жанрі метроїдванія, тому можна виокремити такі задачі.

1. Створення концепції гри.
2. Написання скриптів керування персонажем.
3. Створення спрайтів для гравця.
4. Створення тайлів (плиток) для платформ та декорацій.
5. Реалізація функціоналу меню.
6. Створення інтерфейсу.
7. Вибір шрифту для текстової частини гри.
8. Створення спрайтів для ворогів та інтерактивних об'єктів.
9. Написання скриптів для ворогів.
10. Написання скриптів для інтерактивних об'єктів.
11. Створення префабів (готових ресурсів для багаторазового використання) ворогів та інтерактивних об'єктів.
12. Побудова локації за допомогою готових тайлів.
13. Пошук на безкоштовних сховищах аудіо-зразків звуків та музики, що підходять для атмосфери гри.
14. Тестування гри та виправлення помилок.
15. Компіляція проекту.

2.4 Висновки за розділом 2

Виходячи з того, що розробка відеогри є комплексним процесом, для цього потрібен ціла низка програм, а саме зазначені вище IDE, графічний редактор та ігровий рушій. При виборі цих програм варто орієнтуватися на їх переваги та недоліки, а також зіставити системні вимоги тієї чи іншої програми з потужністю свого ПК. Також варто зазначити, що окрім повноцінних окремих програм при розробці відеоігор відносно часто використовують

різноманітні плагіни та розширення, що якимось чином розширюють функціонал окремих програм, або підвищують ефективність їх роботи.

3 РОЗРОБКА ПРОГРАМНОЇ ЧАСТИНИ СИСТЕМИ

3.1 Загальна концепція відеогри

Розробка будь-якого проекту починається з подання кимось з працівників ідеї, яка згодом трансформується в концепт [19] та дизайн-документ [20]. При написанні обох з цих документів, необхідно розуміти, в чому полягає різниця між ними, та для чого обидва з них потрібні.

У підході до створення концепт-документа дуже поширеним міркуванням є думка про те, що його потрібно написати якомога раніше, одразу, як ідея з'явилася в когось з команди, але насправді його складають тільки після написання диздока, або навіть після створення прототипу гри. Метою створення концепт-документа є обґрунтувати перед інвесторами доцільність створення подібного проекту та описати основну концепцію майбутнього проекту. Об'єм концепт-документа ніким не регламентується, однак в середньому складає 7-8 листів А4 з одностороннім заповненням.

В свою чергу, дизайн-документ являє собою план роботи над проектом з максимально детальним описом гри, що знаходиться на стадії розробки. Створюється диздок спільними зусиллями команд програмістів, дизайнерів та художників. Готовий дизайн-документ може містити зображення, діаграми, текст, концепт-арти та будь-яку іншу інформацію, яка може виявитися необхідною для обґрунтування тих чи інших проектних рішень. Через те, що від тих чи інших концепцій можуть відмовитися під час розробки диздок часто називають «живим» документом, бо протягом усього періоду розробки він може доповнюватися та вдосконалюватися. Саме через «живість» цього типу документів, концепт-документ більш резонно писати вже після того, як був створений прототип гри та диздок прийняв більш-менш стабільний вигляд.

У обох цих документах, крім усього іншого, також зазначається цільова аудиторія даного проекту. За рідким виключенням у ігор, що мають один жанр, ЦА перетинається, але треба мати на увазі, що навіть в рамках одного жанру різні проекти можуть мати графічне та музичне оформлення, що кардинально

відрізняються один від одного. Крім того, варто мати на увазі й системи вікових обмежень, такі як вищезазначена ESRB (рисунок 3.1) або PEGI. Для визначення рекомендованого віку гравців розробники присилають компіляцію з коротких відео, що демонструють сцени з гри, що в теорії можуть когось вразити, образити чиїсь почуття, тощо. Для прикладу можна взяти наведену в пункті 1.1.3 гру Hollow Knight, яка, не дивлячись на свою складність, не містить сцен жорстокості, фізичного насильства, лихослів'я або вживання заборонених речовин, і тому, згідно з критеріями ESRB, має віковий рейтинг E10+, тобто для всіх, хто старше десяти років. Після проведення порівняльного аналізу було виявлено, що більшість метроїдваній за рідким виключенням мають віковий рейтинг PEGI 12, тобто вони призначені для гравців віком 12 років та старше. Віковий рейтинг розробленої для дипломного проекту гри відчувається приблизно як рейтинг «Т» за системою ESRB.



Рисунок 3.1 – Позначення вікових рейтингів системи ESRB

3.1.1 Начерки сюжету гри

На етапі проектування однією з найважливіших речей є вибір сетингу майбутньої гри. Слово «сетинг» [21] переводиться з англійської як «обстановка», або «антураж», та являє собою сукупність елементів що визначають світ в якому відбуваються події того чи іншого твору та регулюють правила, за якими цей світ існує. Поява такого терміну як «сетинг» була обумовлена тим, що трактування слова «жанр» є занадто розпливчатим та його не завжди достатньо щоб описати той чи інший світ. За приклад можна взяти три вигаданих світу: “Dungeons & Dragons”, “Warcraft” та “Might and Magic”. Кожен з цих трьох ігрових світів є частиною жанру фентезі та підкорюються усім його правилам, однак ці світи мають різну історію світу та, при більш детальному огляді, мають свої специфічні правила та ознаки. При розробці дипломного проекту мною був обраний фантастичний сетинг, що поєднує в собі риси світів “Warhammer 40000” та “Dead Space”. Сюжет у грі досить простий, але зі своїми нюансами:

Головний герой – безіменний інженер, що приходить до тьми на капітанському мості вантажного космічного корабля, що перевозить промислових роботів для виконання шахтарської діяльності на віддалених планетах. З якоїсь поки що невідомої причини, весь екіпаж корабля кудись зник, а усі роботи виявилися покриті невідомою органічною субстанцією, що послужило причиною їх агресії по відношенню до гравця. Першочерговою метою головного героя стає ремонт двигуна судна та повернення до рубки управління для того, щоб дістатися назад, на Землю з метою дослідження невідомих організмів. Однак, дістатися до двигуна буде непросто через оскаженілих роботів та пошкодження внутрішніх приміщень. Для подолання перешкод гравцеві доведеться знайти інструменти, якими користувались зниклі члени екіпажу: реактивні чоботи для ривка, що дозволить навіть перелітати великі відстані та ухилятися від ударів супротивників; та магнітні рукавиці, що дозволять гравцеві чіплятися за стіни та здійснювати стрибки від стін.

Коли початкова версія сюжету була готова, прийшов час визначитися з назвою проекту, бо від неї безпосередньо залежить сприйняття гри її цільовою аудиторією. Вона повна легко запам'ятовуватися, бути місткою та відображати основну концепцію гри. Після тривалих роздумів на вибір залишилося 3 варіанти:

- втеча на Землю (Escape to Earth);
- спасіння інженера (Salvation of an engineer);
- важкий день на зміні (Hard day on the shift).

Після порівняльного аналізу та проведення опитування серед знайомих я прийшов до думки, що найбільш доцільною буде назва «Втеча на Землю». З трьох наведених саме ця назва якомога точніше відображає головну мету персонажа.

3.1.2 Візуальний стиль

Тепер, коли основна сюжетна канва була визначена, настав час вирішувати як саме виглядатиме гра. Для підкреслення нереальності усього, що відбувається у проекті, мною було прийняте рішення виконувати графіку у грі в форматі піксель-арту (рисунок 3.2).



Рисунок 3.2 – Візуальне оформлення проекту

З точки зору візуального дизайну гра, на перший погляд, нічим не виділяється, однак безумовною перевагою цього проекту є те, що усі спрайти та тайли були створені з нуля. Також варто зазначити, що не дивлячись на те, що в Unity Editor 2D можливо налаштувати реалістичне високоякісне освітлення, його використання в проекті, на мій погляд, було б помилкою, бо тоді була б порушена візуальна структура проекту. Таким чином, вдалося реалізувати досить приємну графіку, не прибігаючи до підвищення системних вимог.

3.1.3 Основні механіки

Після того, як були визначені сюжетний та графічний напрямки роботи, настав час визначитися з ігровими механіками [22], що будуть задіяні в проекті. Перш за все варто зазначити, що за своєю класифікацією ігрові механіки поділяються на головні, основні та другорядні (таблиця 3.1).

Таблиця 3.1 – Класифікація ігрових механік

Тип механік	Головні механіки	Основні механіки	Другорядні механіки
Короткий опис	Головні, або соге-механіки являють собою дії, які гравець виконує протягом всієї гри знову й знову (стрілянина в шутерах, стрибки в платформерах, тощо)	Основні механіки гравець за своїм бажанням може використовувати від самого початку, або отримує цю можливість на ранніх етапах гри	Другорядні механіки стають доступними гравцеві після виконання якихось умов та можуть використовуватися в окремих специфічних ситуаціях

Тепер, після того як було дано визначення, треба вирішити, які ігрові механіки будуть використані в проекті. Перш за все, будуть реалізовані ключові для платформерів механіки, такі як переміщення головного персонажа в горизонтальній площині та стрибки. Основною механікою представленої у дипломному проекті гри стане бойова система, через те, що окрім подорожей

по пошкодженому судну, гравцеві необхідно буде прокладати собі шлях скрізь оскаженілі механізми. У свою чергу, другорядними механіками стануть зазначені в пункті 2.1.1 ривок та чіпляння за стіни.

3.2 Написання скриптів для реалізації ігрових механік

Після того, як було вирішено якої концепції варто притримуватися при створенні проекту, настав час для програмної реалізації основних механік гри. У всіх файлах з кодом, що були задіяні в проекті використовується стандартний для рушія Unity простір імен UnityEngine, але в деяких з файлів є й додаткові, їх ми розглянемо далі. За своєю ієрархією скрипти були розподілені на три папки:

- папка “Characters” містить в собі скрипти, що відповідають за поведінку гравця та трьох типів супротивників;
- в папці “Menus” знаходяться скрипти, якими контролюються, відповідно, ігрові меню, елементи інтерфейсу та паралакс-ефекту для фонових зображень;
- в свою чергу, папка “Objects” містить скрипти для об’єктів, з якими може взаємодіяти гравець.

Перш за все розглянемо вміст папки “Characters”. Вона містить такі файли:

- Character.cs;
- PlayerController.cs;
- DroneController.cs;
- MeleeController.cs;
- RangeController.cs;
- Projectile.cs;
- FOV.cs.

При написанні скриптів для персонажів на мій погляд було доцільно застосувати знання з ООП, набуті під час навчання. Розглянемо структуру файлів у даній теці. Базовим класом для гравця та NPC є клас Character, що

міститься у відповідному файлі. Цей клас відповідає за усі процеси, що можуть відбуватися як з гравцем, так і з іншими персонажами. Далі буде приведено список методів, що містить базовий клас:

- `Start ()`; Викликається перед першим фреймом. У даному випадку в ньому відбувається ініціалізація компоненту `Animator`, що відповідає за відтворення анімацій, а також компоненту `Rigidbody2D`, що поміщає об'єкт під контроль фізичного рушія. Кожен з підкласів перезаписує його під свої потреби;

- `Update ()`; Викликається один раз за кожен кадр для його прорисовки. У випадку з цим проектом в методі `Update` відбувається перевірка чи стоїть персонаж на твердій поверхні та чи стоїть гра на паузі;

- `FixedUpdate ()`; За замовченням викликається 50 разів за секунду незалежно від частоти кадрів для фізичних розрахунків. У класі `Character` відповідає за прорахунок переміщення персонажів шляхом виклику функції `HandleMovement ()`;

- `Move ()`; В цьому методі йде перевірка чи може персонаж рухатися, та якщо може його фізичній оболонці (`Rigidbody`) надається прискорення по осі `Ox`;

- `Jump ()`; У даному методі відбувається перевірка чи може персонаж здійснити звичайний стрибок, або стрибок від стіни;

- `Attack ()`; Даний метод опрацьовує атаку одним персонажем іншого, шляхом перевірки так званих «хітбоксів»;

- `OnDrawGizmosSelected ()`; Відповідає за візуальне відображення так званих гізмо (умовних позначок, будь-то лінії, сфери чи інші геометричні фігури) коли об'єкт, до якого ці гізмо приєднані, вибраний;

- `TakeDamage ()`; Відповідає за отримання персонажами пошкоджень та втрату їх `HP`. Після отримання персонажем пошкоджень запускає корутину `GetInvulnerable`. Якщо після отримання пошкоджень здоров'я персонажа менше або дорівнює нулю, починає виконуватися метод `Die()`;

- `GetInvulnerable ()`; Корутина, протягом якої персонаж стає несприйнятливим до будь-яких пошкоджень;
- `Die ()`; Метод, який запускається коли здоров'я персонажа знижується до нуля;
- `HandleMovement ()`; та `HandleJumping ()`; Викликають `Move` та `Jump` відповідно. На відміну від оригінальних функцій, їх можуть перезаписувати підкласи;
- `TurnAround (float horizontal)`; Метод, який відповідає за розворот персонажа в горизонтальній площині.

Першим зі списку породжених класів буде клас `PlayerController`. Методи, які містяться в ньому відповідають за управління головним героєм. Окрім використання стандартного простору імен `UnityEngine`, в даному скрипті використовується `TMPPro`, який відповідає за відображення тексту. В даному скрипті використовувалися такі методи:

- `Start ()`; В даному підкласі метод `Start` задає гравцеві максимальне значення здоров'я, максимальну кількість «хілок» (засобів для лікування) та ініціалізує використання шкали здоров'я;
- `Update ()`; В класі `PlayerController` в методі `Update` виконується зчитування натиснутих клавіш та зіставлення їх з відповідними функціями. Таким чином, при натисненні на клавіші `A` та `D` персонаж рухатиметься відповідно вліво та вправо, при натисненні на `Space` – стрибати (при затисненні `Space` персонаж стрибатиме вище), якщо персонаж вже отримав відповідне вміння, то за натисненням лівого `Shift` персонаж виконуватиме ривок у напрямку руху, а після натиснення `R` персонаж відновлює собі фіксований об'єм здоров'я;
- `Dash ()`; Запускає корутину `Dashing` та відтворює відповідну анімацію.
- `Dashing ()`; Корутина, на час дії якої персонажу надається прискорення та він стає несприйнятливим до ворожих атак;

- `CheckIfWallsliding ()`; Перевіряє, чи може персонаж сковзати по стіні;
- `HandleMovement ()`; В цьому підкласі в методі `HandleMovement` відбувається перевірка чи зісковзує гравець по стіні та якщо зісковзує, то швидкість ковзання зменшується;
- `SwordAttackSlash ()`; Корутина, що запускається на час атаки;
- `OnDrawGizmos ()`; Візуалізує гізмо;
- `HandleLayers ()`; Відповідає за переключення шарів під час стрибка в інструменті `Animator` в рушії;
- `Heal ()`; Метод, завдяки якому реалізується механіка лікування персонажа;
- `TakeDamage ()`; У підкласі `CharacterController` цей метод обчислює втрату HP та відповідає за зменшення шкали здоров'я.

Далі в ієрархії знаходяться скрипти `MeleeController.cs` та `RangeController.cs` (Додаток А), що відповідають за поведінку супротивників з атаками ближнього та дальнього бою відповідно. Ці два класи мають спільний алгоритм роботи, але замість простої атаки робот-стрілець активує префаб снаряду, що летить у заданому напрямку з визначеною швидкістю. Розглянемо методи, наведені в цих скриптах більш детально:

- `Start ()`; Задається максимальне значення здоров'я та встановлюється швидкість персонажа;
- `Update ()`; В даному методі знаходиться «мозок» персонажа, який кожен кадр проводить обчислення подальшої поведінки персонажа за алгоритмом, наведеним на рисунку 3.3;
- `OnDrawGizmos ()`; Відображає гізмо в редакторі Unity;
- `EnemyAttack ()`; Корутина, що запускається під час атаки.

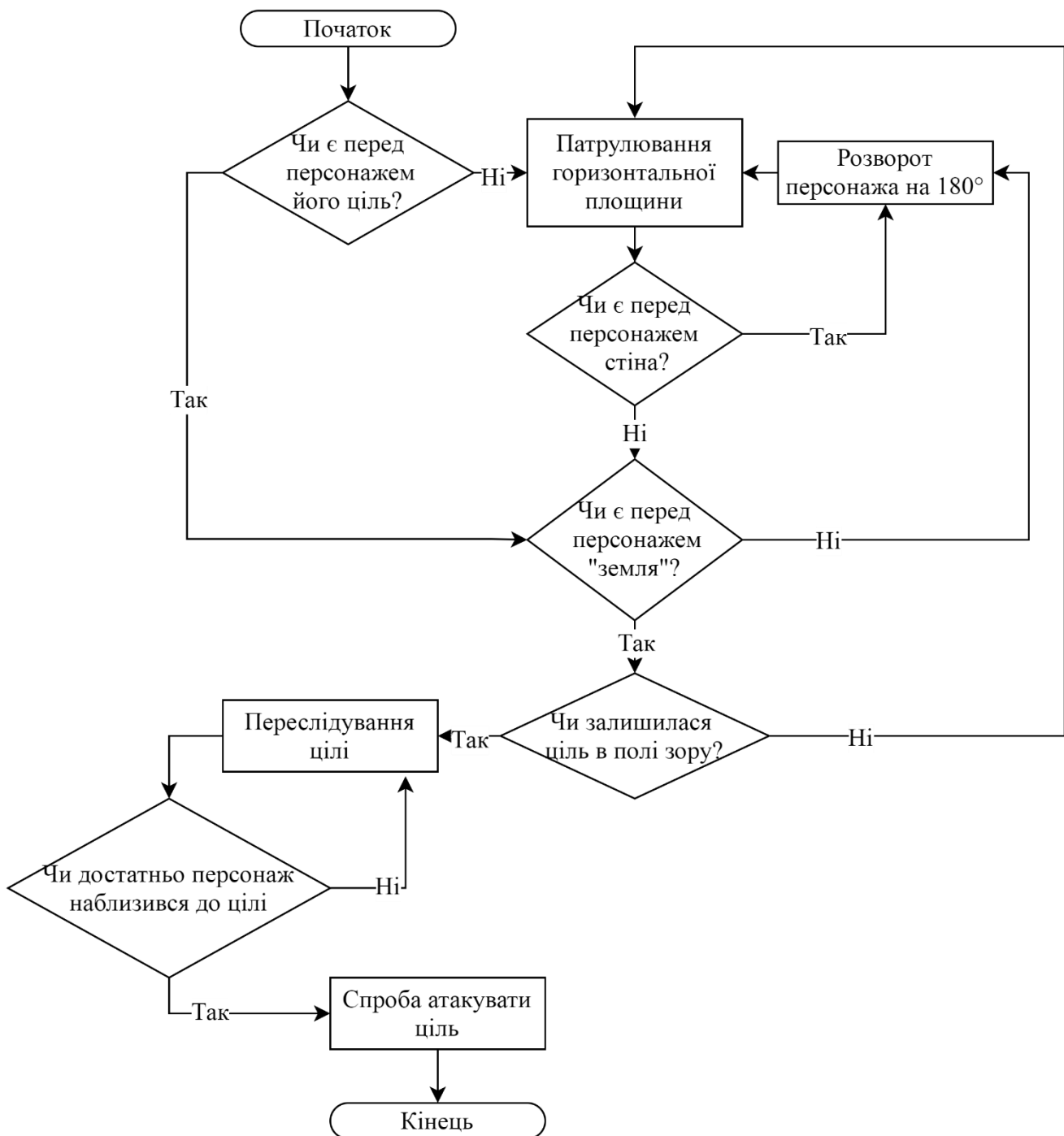


Рисунок 3.3 – Блок-схема, що демонструє алгоритм

Крім того, варто зазначити файл `Projectile.cs`, що відповідає за поведінку снаряду, який випускає персонаж, що керується скриптом `RangeController`.

Перелік методів, що використовується в даному класі:

- `Start ()`; Перед першим фреймом призначає префабу швидкість польоту в напрямку гравця;

- `Awake ()`; Викликається до початку будь-яких функцій або після ініціалізації префабу. В даній ситуації дозволяє снаряду ігнорувати радіус взаємодії з інтерактивними предметами, який був реалізований за допомогою `Circle Collider 2D` (на самій реалізації детально зупинимося пізніше);

- `OnTriggerEnter2D (Collider2D hitInfo)`; По своїй суті є аналогом методу `Attack` з класу `Character`, дозволяє снаряду наносити пошкодження гравцеві.

Останнім підкласом у цьому списку буде `DroneController` (Додаток Б), що відповідає за літаючих супротивників. Окрім всього іншого, цей клас використовує простір імен `Pathfinding`, який був доданий до проекту плагіном `A* Pathfinding`:

- `Start ()`; Робить усе те ж саме, що й у інших підкласах класу `Character`, за винятком того, що тут за допомогою метода `InvokeRepeating` в ньому викликається функція `UpdatePath` що повторюватиметься кожні пів секунди;

- `UpdatePath ()`; Відповідає за побудову шляху до цілі за алгоритмом `A*`;

- `OnPathComplete ()`; Перевіряє, чи правильно був побудований шлях;

- `Update ()`; В даному випадку, в методі `Update` міститься код, що відповідає за атаку цілі при її досягненні;

- `FixedUpdate ()`; У цьому скрипті `FixedUpdate` реалізує проходження персонажем шляху при умові, що ціль знаходиться у його полі зору, що було реалізоване у скрипті `FOV` завдяки методам `OnTriggerEnter2D` та `OnTriggerExit2D`;

- `OnDrawGizmos ()`; Відповідає за візуалізацію гізмо, як і у інших скриптах.

Тепер, коли були розглянуті усі скрипти з папки `Characters`, треба коротко розглянути вміст папок `Menus` та `Objects`. У ситуації з цими скриптами

немає сенсу зупинятися детально на кожному методі, бо здебільшого використовується один скрипт, що модифікується в залежності від об'єкту. Перш за все, подивимося на теку `Objects`, в якій знаходяться скрипти, що відповідають за взаємодію персонажа з об'єктами:

- `Interact.cs` – перевіряє, чи є персонаж у радіусі для взаємодії;
- `ChestController.cs` – відповідає за взаємодію персонажа зі скринями, які містять важливі для сюжету здібності;
- `TerminalController.cs` – цим скриптом керуються термінали, якими відкриваються двері;
- `DoorOpener.cs` – скрипт, що відповідає за відкриття дверей;
- `Elevator.cs` – відповідає за керування ліфтом;
- `MainTerminalController.cs` та `EngineController.cs` – два скрипти, що відповідають за сюжетні об'єкти, а саме головний термінал та двигун.

Останньою папкою, що містить важливі для гри скрипти це `Menus`. Всього в ній знаходиться 5 скриптів:

- `MainMenu` – відповідає за кнопки у головному меню;
- `SetVolume` – скрипт, що дозволяє регулювати рівень гучності у грі;
- `PauseMenu` – завдяки цьому скрипту при натисненні клавіші `Escape` ігровий процес зупиняється та відкривається меню, що за функціоналом майже ідентичне головному меню;
- `HealthBar` – цей скрипт відповідає за відображення здоров'я персонажа на екрані;
- `ParallaxCam` – не є якимось з елементів меню, скоріше декоративним; Завдяки цьому скрипту зображення на задньому плані рухаються з різною швидкістю (чим далі зображення, тим менша його швидкість) під час руху персонажа, а отже й камери.

3.3 Опрацювання графічної частини

Після того, як були написані та перевірені усі скрипти, настав час створювати графічну частину проекту. Для цього дипломного проекту мною був обраний напрям графічного дизайну, відомий як піксель-арт. Його легко розпізнати серед простих растрових зображень завдяки обмеженій палітрі кольорів та невеликим розмірам самих зображень. Здебільшого візуальний стиль цього проекту поєднує в собі елементи класичних фантастичних фільмів як-от «Чужий» та зазначеної в пункті 1.1.4 гри Blasphemous. Першою поставленою задачею було створення тайлів для твердих блоків та предметів оточення (рисунок 3.4). Тайлами в сфері розробки відеоігор прийнято називати текстури, що по суті являють собою плитки однакових розмірів, які при цьому з'єднуються між собою без швів. Для дипломного проекту мною були створені квадратні тайли розміром 16 на 16 пікселів.

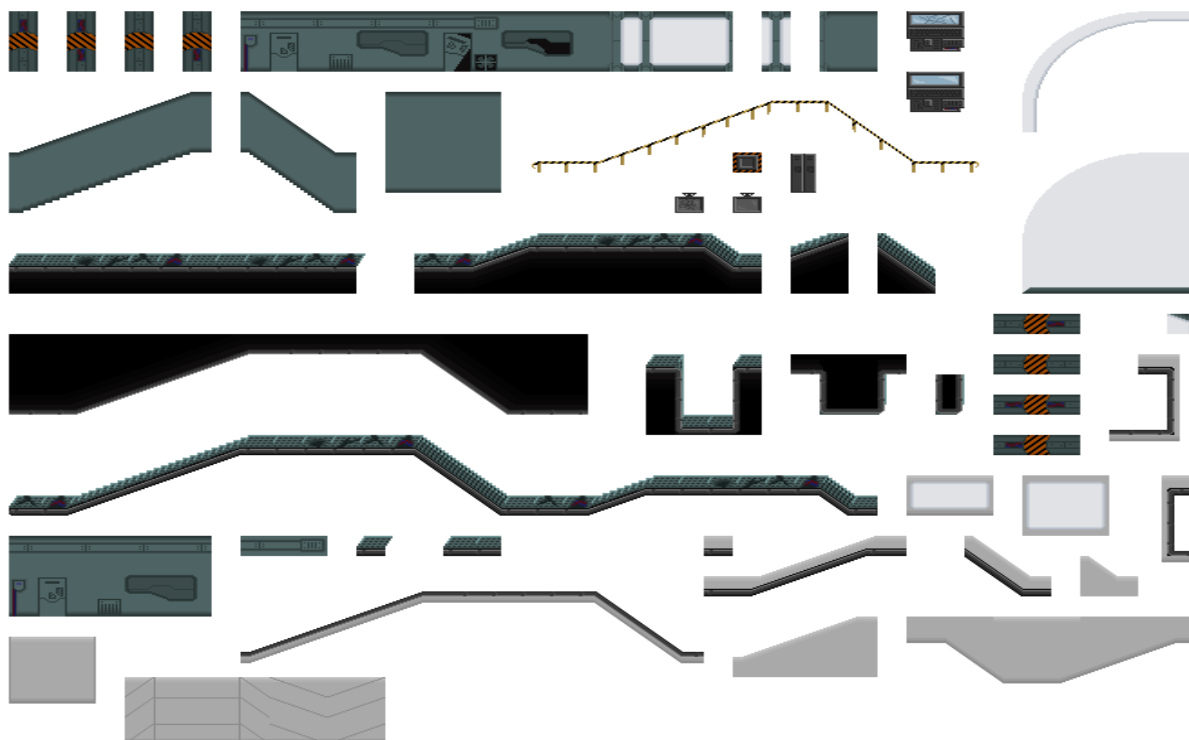


Рисунок 3.4 — Тайли, що використовуються в дипломному проєкті

Після того, як були готові тайли, прийшов час створювати спрайти для персонажів. Першим на черзі був головний герой, він же безіменний інженер. При створенні його образу я здебільшого спирався на головних героїв серій ігор «Dead Space» та «DOOM» з тієї причини, що вони теж опинилися на напівзруйнованому кораблі у достатньо ворожому середовищі. У результаті вийшов достатньо цікавий та детально пропрацьований з візуальної точки зору персонаж, спрайти для анімацій якого можна побачити на рисунку 3.5.



Рисунок 3.5 — Лист спрайтів головного персонажа

Коли усі спрайти для головного героя були готові я взявся за створення ворогів. Першим типом ворогів, що спав мені на думку, був літаючий робот. Мною було прийняте рішення зробити його та усі подальші ворожі механізми вкритими невідомою органічною субстанцією згідно з сюжетом гри. Орієнтирами в цьому напрямку роботи послужили головний персонаж гри «Carrion», що нещодавно вийшла, та демонічні механізми з сетингу «Warhammer 40000». Таким чином був створений незвичайний симбіоз літаючого дрона-дослідника та хижої безформної істоти (рисунок 3.6).



Рисунок 3.6 — Лист спрайтів для літаючого супротивника

Далі за планом було створення наземних роботів, що атакуватимуть головного героя поблизу та здалека. За задумкою гри на космічному кораблі з ГГ окрім літаючих дронів-дослідників також перевозилися промислові роботи

для виконання робіт у шахтах. Через це за своїм призначенням ці роботи поділялися на два типи: роботи для видобутку твердих порід та роботи для видобутку твердого палива. З метою економії, компанія-виробник вирішила стандартизувати виробництво даних типів механізмів, у результаті чого вони відрізняються лише інструментом, яким вони обладнані: у першого типу це промисловий лазер, а у другого – циркулярна пила (рисунок 3.7). Цим зумовлений майже однаковий зовнішній вигляд цих двох типів ворогів.

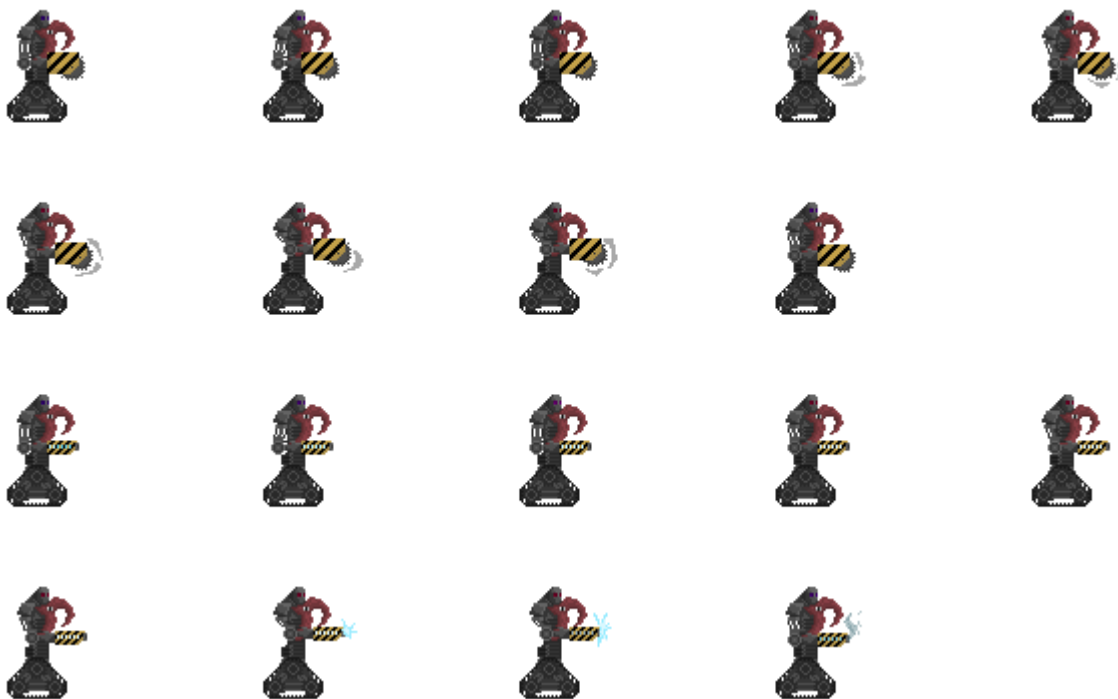


Рисунок 3.7 — Приклад спрайтів анімацій двох типів промислових роботів

Окрім тайлів та спрайтів персонажів для гри були потрібні спрайти інтерактивних об'єктів. Через те, що гра має сетинг космічної фантастики, об'єкти в ній мають бути відповідними. До переліку таких об'єктів входять:

- ліфт;
- пульт управління ліфтом;
- скрині, що містять ключові предмети;
- термінали для відкриття дверей;

- двері;
- двигун корабля, який треба відремонтувати згідно з сюжетом гри.

Після того, як всі об'єкти, персонажі та елементи декору були зроблені, настав час додавати всі ці спрайти у редактор рушія. Першим на черзі стало зображення, що містить в собі всі тайли для створення оточення. Для подальшого створення локацій у грі нам знадобиться так звана «палітра тайлів», яку можна створити у відповідному вікні в Unity Editor (рисунок 3.8) після «розрізання» початкового зображення на невеликі квадрати розміром 16 на 16 пікселів у вікні Sprite Editor (редактор спрайтів).

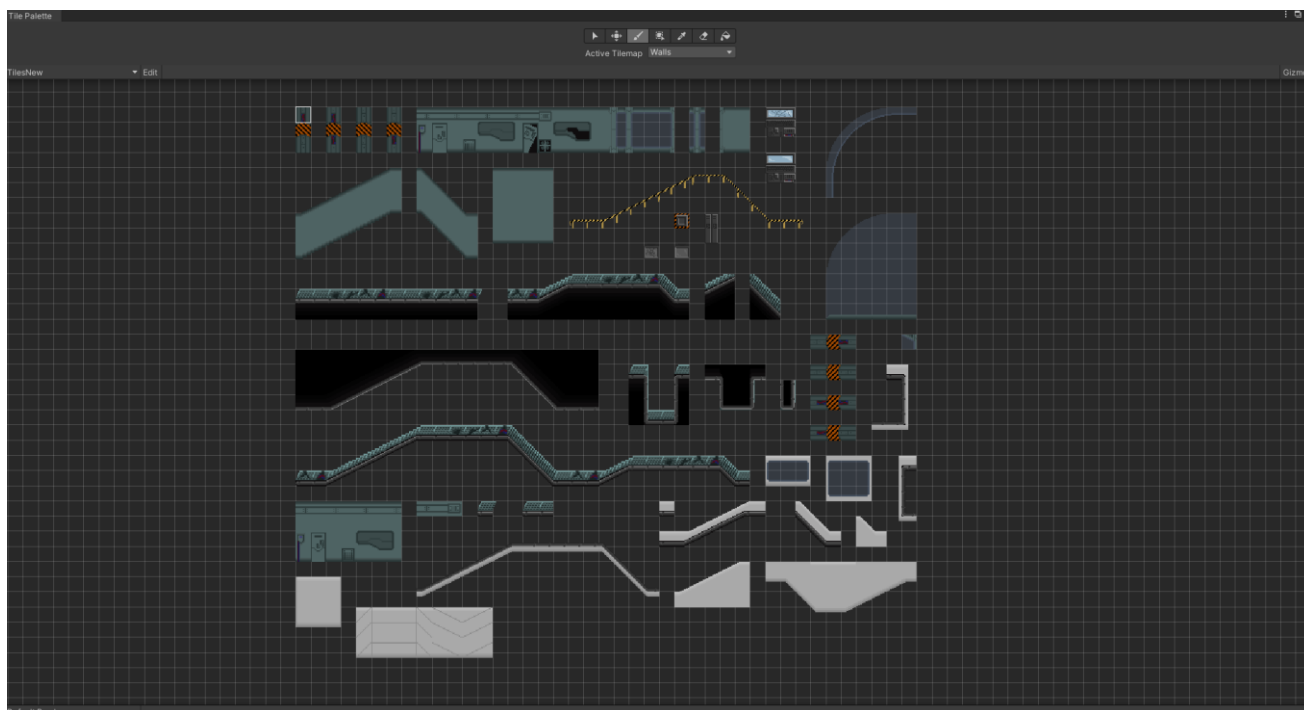


Рисунок 3.8 — Палітра тайлів в Unity Editor

Далі, після створення плиток, потрібно зайнятися створенням файлів анімацій для усіх анімованих об'єктів. Для цього у тому ж вікні Sprite Editor нам потрібно «розрізати» вихідне зображення відповідно з розмірами та кількістю кадрів в самій анімації. Вже після цього, шляхом перетягування потрібної кількості кадрів до вікна ієрархії в редакторі, формуються безпосередньо анімації. Для того, щоб заставити персонажа рухатися

відповідно заданим умовам, потрібно приєднати до одного й того ж об'єкта його скрипт та відповідний контролер анімації. Тепер треба роз'яснити як же працює цей самий контролер анімації. По своїй суті, інструмент анімації в Unity являє собою так звану *finite-state machine*, або машину зі скінченною кількістю станів. Її сенс полягає у тому, щоб при виконанні певних умов змінювався стан об'єкта. У випадку з інструментом Animator (рисунок 3.9), при зміні значення змінної буде програватись відповідна анімація.

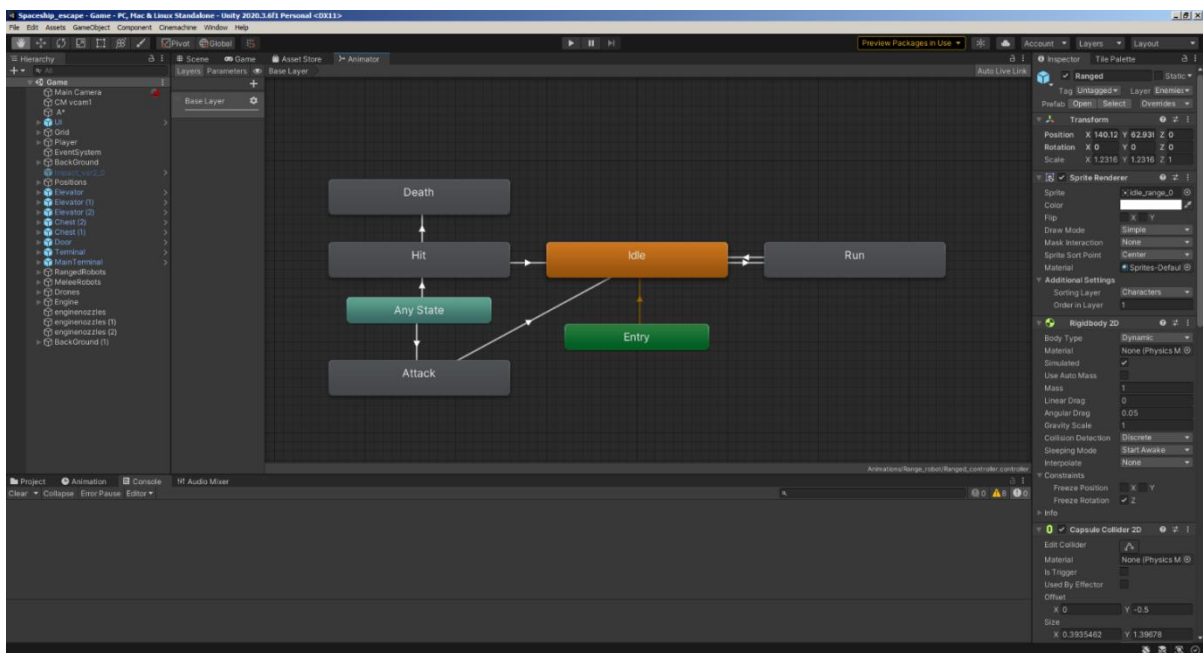


Рисунок 3.9 — Вікно Animator в редакторі Unity

Окрім звичайних спрайтів та анімацій було дуже важливо створити графічний інтерфейс користувача, в тому числі від нього залежить сприйняття гри. Перше, що ми бачимо при запуску проекту – це головне меню (рисунок 3.10). Воно містить назву гри, задній план у вигляді зоряного неба та три функціональні кнопки:

- Play – після натиснення відбувається перехід на основну сцену гри;
- Options – при натисненні відкривається вікно налаштувань гри, у якому можна змінити рівень гучності музики;
- Quit – вихід з гри.



Рисунок 3.10 — Зовнішній вигляд головного меню гри

Наступною після реалізації меню важливою задачею було створення користувацького інтерфейсу, що відображатиметься під час ігрового процесу. Він має відображати основну інформацію про гравця, у даному разі це здоров'я гравця та кількість лікувальних засобів (рисунок 3.11). За задумкою, кількість пошкоджень, що може витримати персонаж, характеризується зарядом батареї до повної розрядки костюма, саме тому шкала здоров'я виконана у вигляді акумулятора. В цей самий час, лікувальні засоби виглядають як аптечки у більшості старих відеоігор: стилізований квадрат з червоним хрестом.



Рисунок 3.11 — Зовнішній вигляд графічного інтерфейсу користувача

Окрім цього, було створене меню паузи (рисунок 3.12), що відкривається по натисненню клавіши Escape функціонал якого майже ідентичний головному меню, окрім: кнопки Menu, що повертає користувача назад в головне меню; та підказок в плані клавіш керування персонажем.



Рисунок 3.12 — Меню паузи в грі

3.4 Левел-дизайн (побудова локацій)

Сам процес побудови локацій в редакторі рушія не є чимось складним. По своїй суті все, що потрібно робити – це буквально малювати рівень окремими плитками, але є нюанс. За словами колишнього головного левел-дизайнера компанії Eidos-Montréal Дена Тейлора при побудові рівнів важливо притримуватись десяти «золотих правил» левел-дизайну [23], [24], згідно з якими він:

- підсилює гравця;
- дозволяє гравцеві контролювати складність;
- є ефективним;
- створює емоції;

- диктується механіками гри;
- не покладається на слова, щоб розповісти історію;
- показує гравцеві ЩО треба робити, але не ЯК це треба робити;
- постійно вчить гравця чомусь;
- дивує гравця;
- є цікавим для дослідження.

У дипломної роботі згідно з переліченими вище правилами мною була побудована локація, що являє собою напівзруйнований космічний корабель.

3.5 Тестування готової гри

Після того, як технічна частина проекту була зроблена, починається етап тестування [25]. Сам процес тестування зазвичай складається з дев'яти етапів, але в даному випадку нас цікавить тільки два з них: функціональне тестування та тестування зручності використання (юзабіліті-тестування). Їх докорінна відмінність полягає в тому, що функціональні тести перевіряють реалізацію функціональних задач та її відповідність вхідним вимогам, у той час як юзабіліті-тестування виконується з метою дослідження зручності користування пристроєм або проектом.

Перш за все мною було проведене функціональне тестування, що виконувалось в три етапи.

Перевірка запуску гри.

Перш за все було важливо перевірити чи запускається проект на комп'ютерах під керуванням операційних систем Windows 7, Windows 10 та Ubuntu. Очікуваним результатом був стабільний запуск на усіх перелічених системах. Отриманий результат повністю збігається з очікуваним, а отже, мета цього тесту досягнута.

Перевірка працездатності елементів меню.

Метою цього етапу стала перевірка того, як працюють елементи головного меню та меню паузи. По натисненню на кнопку «Options» і в меню

паузи і в головному меню відкривається вікно налаштувань, де можна відрегулювати гучність звукових ефектів та повернутися назад до основної частини меню (рисунок 3.13).



Рисунок 3.13 — Екран налаштувань гри

При натисненні кнопки «Play» у головному меню запуситься гра, приблизно за те саме відповідає кнопка «Resume» у меню паузи, а по натисненню «Menu» у тому ж меню паузи – відкриється головне меню. Кнопка «Quit» відповідає за вихід з гри, та вона працює без нарікань. Виходячи з цього, цей тест можна вважати успішним, бо отриманий результат збігається з очікуваним.

Перевірка працездатності ігрових механік.

Цей етап в тестуванні можна вважати найдовшим. Перш за все мною було перевірено базову механіку переміщення рівнем, а саме біг та стрибки. Під час тестування стрибків мною була виявлена проблема: коли персонаж в польоті торкався стіни, він «прилипав» до поверхні. Ця помилка була вирішена створенням в рушії для твердої поверхні фізичного матеріалу, сила тертя якого

дорівнює нулю. Далі була протестована механіка отримання пошкоджень головним героєм та іншими персонажами (рисунок 3.14).



Рисунок 3.14 — Демонстрація отримання персонажем пошкоджень

При тестуванні цієї механіки проблем виявлено не було. Також не було виявлено несправностей при тестуванні додаткових механік, а саме стрибків від стін та ривків. Так як усі несправності, що виникли, були виправлені, а всі інші механіки працюють без нарікань та загалом відповідають очікуванням, цей тест також можна вважати успішним.

Після того, як було завершено функціональне тестування, настав час для юзабіліті-тестування, яке проводиться шляхом залучання користувачів у якості тестувальників, та основною метою якого є отримати відгук від потенційної цільової аудиторії. Алгоритм виконання цього тесту складається з чотирьох етапів:

- 1) постановка задач тестування;
- 2) відбір тестувальників;
- 3) проведення тестування;

4) підведення підсумків тестування.

Перш за все потрібно правильно визначити задачі тестування. Метою цього тесту є визначити, наскільки користувачеві зручно користуватись проектом. У разі з відеогрою, потрібно визначити наскільки гра збалансована та наскільки в неї цікаво грати. Згідно з цим мною був побудований наступний список задач:

- 1) запустити гру;
- 2) встановити комфортний рівень гучності;
- 3) почати нову гру;
- 4) зібрати усі здібності героя;
- 5) успішно пройти гру.

Тепер, після того як були визначені задачі для тестування, настав час відбору команди тестувальників. Як було зазначено в пункті 3.1, віковий рейтинг гри знаходиться приблизно на позначці «від 13 та старше», але варто мати на увазі, що більшість підлітків цього віку не цікавлять проекти подібного жанру, у той час як справжніми його поціновувачами є люди віком приблизно від 16 до 40 років. Згідно з цим мною був сформований такий склад команди тестувальників:

- Наливайко Артур Юрійович, 21 рік, рід занять – студент;
- Бобришев Дмитро Вікторович, 21 рік, рід занять – студент;
- Гаврилов Валентин Сергійович, 29 років, рід занять – робітник;
- Тітов Іван Миколайович, 16 років, рід занять – школяр.

Після формування команди, тестувальники почали перевірку, результати якої можна побачити в таблиці 3.2.

Під час тестування деякі з його учасників скаржилися на складність окремих сегментів з платформингом та на неочевидність деяких елементів управління персонажем, у результаті чого в меню паузи з'явився список дій та відповідних цим діям клавіш.

Таблиця 3.2 — Результати юзабіліті-тестування

ПІБ тестувальн ика	Задача 1	Задача 2	Задача 3	Задача 4	Задача 5	Затрачени й час, хв.	Процент виконаних задач
Наливайко А.Ю	Так	Так	Так	Так	Ні	54	80%
Бобришев Д.В	Так	Так	Так	Так	Так	43	100%
Гаврилов В.С	Так	Так	Так	Так	Так	22	100%
Тітов І.М	Так	Так	Так	Ні	Ні	58	60%

ВИСНОВКИ

У наш час відеоігри вже стали не тільки невід'ємною частиною розважальної індустрії, але й, згідно з розповсюдженою думкою, повноцінним видом мистецтва. Незважаючи на поширену думку про те, що в Україні галузь розробки відеоігор не сильно розвинена, саме з-під пера наших співвітчизників вийшли такі шедеври світового масштабу, як серії ігор «S.T.A.L.K.E.R.», «Metro» та «Козаки».

Для реалізації поставленої задачі я проаналізував фахові літературу та інтернет-ресурси, що пов'язані з розробкою ПЗ в цілому та відеоігор зокрема, також були проаналізовані схожі проекти. Крім того, базуючись на цих знаннях, було обрано жанр розроблюваної гри, створена загальна концепція гри та були досліджені методи використання відповідних програм.

Під час створення з нуля власної гри я отримав дуже цінний досвід в багатьох сферах, пов'язаних з розробкою та реалізацією ігор, як-от: створення ігрових концепцій, опрацювання візуального дизайну, створення алгоритмів та написання коду, навички роботи з ігровим рушієм.

В результаті, завдяки навичкам в сфері програмування, піксель-арту та розробки ігрових концепцій, набутих під час навчання в університеті мною була створена відеогра в жанрі метроїдванія. При розробці мною використовувалися ICP Microsoft Visual Studio Code, піксельний графічний редактор Aseprite та ігровий рушій Unity 2D.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. ESRB Game Ratings [Electronic resource]. – Access mode: <https://www.esrb.org> (дата звертання 10.04)
2. Schreier, J. Blood, Sweat, and Pixels: The Triumphant, Turbulent Stories Behind How Video Games Are Made / J. Schreier – New York: Harper, 2017. – 304 p.
3. Жанри відеоігор [Електронний ресурс] – Режим доступу: https://uk.wikipedia.org/wiki/Жанри_відеоігор (дата звертання 10.04)
4. Metroidvania [Electronic resource]. – Access mode: <https://en.wikipedia.org/wiki/Metroidvania> (дата звертання 10.04)
5. Metroid [Electronic resource]. – Access mode: <https://en.wikipedia.org/wiki/Metroid> (дата звертання 10.04)
6. Castlevania [Electronic resource]. – Access mode: <https://en.wikipedia.org/wiki/Castlevania> (дата звертання 10.04)
7. Castlevania: Symphony of the Night [Electronic resource]. – Access mode: https://en.wikipedia.org/wiki/Castlevania:_Symphony_of_the_Night (дата звертання 10.04)
8. Dark Souls [Electronic resource]. – Access mode: https://en.wikipedia.org/wiki/Dark_Souls (дата звертання 10.04)
9. Hollow Knight [Electronic resource]. – Access mode: https://en.wikipedia.org/wiki/Hollow_Knight (дата звертання 11.04)
10. Blasphemous (video game) [Electronic resource]. – Access mode: [https://en.wikipedia.org/wiki/Blasphemous_\(video_game\)](https://en.wikipedia.org/wiki/Blasphemous_(video_game)) (дата звертання 11.04)
11. Розробка відеогри [Електронний ресурс]. – Режим доступу: https://uk.wikipedia.org/wiki/Розробка_відеогри (дата звертання 12.04)
12. Ігровий рушій [Електронний ресурс]. – Режим доступу: https://uk.wikipedia.org/wiki/Ігровий_рушій (дата звертання 12.04)

13. Інтегроване середовище розробки [Електронний ресурс]. – Режим доступу: https://uk.wikipedia.org/wiki/Інтегроване_середовище_розробки (дата звертання 12.04)
14. Графічний редактор [Електронний ресурс]. – Режим доступу: https://uk.wikipedia.org/wiki/Графічний_редактор (дата звертання 12.04)
15. Unity (game engine) [Electronic resource]. – Access mode: [https://en.wikipedia.org/wiki/Unity_\(game_engine\)](https://en.wikipedia.org/wiki/Unity_(game_engine)) (дата звертання 13.04)
16. Visual Studio Code [Electronic resource]. – Access mode: https://en.wikipedia.org/wiki/Visual_Studio_Code (дата звертання 13.04)
17. Aseprite [Electronic resource]. – Access mode: <https://www.aseprite.org> (дата звертання 13.04)
18. Computer System Requirements to Develop with Unity – Unity Forum [Electronic resource]. – Access mode: <https://forum.unity.com/threads/computer-system-requirements-to-develop-with-unity.647854/> (дата звертання 15.04)
19. Концепт-документ: анатомія, специфіка і розбір тестового кейса – UE4 Daily [Електронний ресурс] – Режим доступу: <https://ue4daily.com/blog/concept-doc> (дата звертання 16.04)
20. Дизайн-документ [Електронний ресурс] – Режим доступу: <https://ru.wikipedia.org/wiki/Дизайн-документ> (дата звертання 16.04)
21. Сетинг [Електронний ресурс]. – Режим доступу: <https://uk.wikipedia.org/wiki/Сетинг> (дата звертання 16.04)
22. Ігрова механіка [Електронний ресурс] – Режим доступу: https://ru.wikipedia.org/wiki/Ігровая_механика (дата звертання 17.04)
23. Gamasutra: Dan Taylor's Blog – Ten Principles of Good Level Design (Part 1) [Electronic resource]. – Access mode: https://www.gamasutra.com/blogs/DanTaylor/20130929/196791/Ten_Principles_of_Good_Level_Design_Part_1.php (дата звертання 17.04)
24. Gamasutra: Dan Taylor's Blog – Ten Principles of Good Level Design (Part 1) [Electronic resource]. – Access mode:

https://www.gamasutra.com/blogs/DanTaylor/20131006/197209/Ten_Principles_of_Good_Level_Design_Part_2.php (дата звертання 17.04)

25. Тестування програмного забезпечення [Електронний ресурс]. – Режим доступу:
https://uk.wikipedia.org/wiki/Тестування_програмного_забезпечення (дата звертання 20.04)

ДОДАТОК А

Текст програми

A.1 Текст скрипта RangeController.cs

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

[RequireComponent(typeof(Rigidbody2D))]
[RequireComponent(typeof(Animator))]

public class RangeController : Character
{
    public Transform target;
    public Transform fieldOfView;
    public Transform groundDetection;
    public Transform enemyDetection;
    public Transform enemyDetectionFromBack;
    RaycastHit2D findCharacters;
    RaycastHit2D detectEnemy;
    RaycastHit2D detectEnemyFromBack;
    RaycastHit2D WallInfo;
    public Transform firepoint;
    public float rayDistance;
    public float chaseSpeed;
    public float nextWaypointDistance = 3;
    public float FOVDistance = 2f;
    public float rangeOfView;
    public float bckDetection;
    int currentWayPoint = 0;
    public int maxEnemyHealth;
    protected bool readyToAttack;
```

```

bool enemyFound = false;
bool enemyInRange = false;
private bool movingRight = true;
public GameObject impactFX;
public GameObject arrowPrefab;

public override void Start()
{
    Physics2D.IgnoreLayerCollision(6,9);
    rb2d = GetComponent<Rigidbody2D>();
    myAnim = GetComponent<Animator>();
    currentHealth = maxEnemyHealth;
    readyToAttack = true;
    facingRight = false;
    speed = chaseSpeed;
}

public override void Update() {
    base.Update();
    myAnim.SetFloat("Speed", chaseSpeed);
    if(movingRight){
        findCharacters = Physics2D.Raycast(fieldOfView.position, Vector2.right,
FOVDistance, enemyLayers);
        WallInfo = Physics2D.Raycast(groundDetection.position, Vector2.right,
rayDistance, whatIsGround);
        detectEnemy = Physics2D.Raycast(enemyDetection.position, Vector2.right,
rangeOfView, enemyLayers);
        detectEnemyFromBack =
Physics2D.Raycast(enemyDetectionFromBack.position, Vector2.left, bckDetection,
enemyLayers);
    }
}

```

```

else
{
    findCharacters = Physics2D.Raycast(fieldOfView.position, Vector2.left,
FOVDistance, enemyLayers);
    WallInfo = Physics2D.Raycast(groundDetection.position, Vector2.left,
rayDistance, whatIsGround);
    detectEnemy = Physics2D.Raycast(enemyDetection.position, Vector2.left,
rangeOfView, enemyLayers);
    detectEnemyFromBack =
Physics2D.Raycast(enemyDetectionFromBack.position, Vector2.right, bckDetection,
enemyLayers);
}
RaycastHit2D GroundInfo = Physics2D.Raycast(groundDetection.position,
Vector2.down, rayDistance, whatIsGround);
if(findCharacters.collider == true){
    enemyInRange = true;
}
else
{
    enemyInRange = false;
}
if(detectEnemy.collider == true){
    enemyFound = true;
}
else
{
    enemyFound = false;
}
if(detectEnemyFromBack.collider == true)
{

```

```

        if(movingRight || !movingRight){
            movingRight = !movingRight;
            transform.Rotate(0f, -180f, 1f);
        }
    }
    if(enemyFound == true)
    {
        transform.position = Vector2.MoveTowards(transform.position,
target.position, chaseSpeed * Time.deltaTime);
    }
    if(enemyFound == false)
    {
        transform.Translate(Vector2.right * chaseSpeed * Time.deltaTime);
        if(GroundInfo.collider == false || WallInfo.collider == true){
            if(movingRight || !movingRight){
                movingRight = !movingRight;
                transform.Rotate(0f, -180f, 1f);
            }
        }
    }
    if(enemyInRange){
        chaseSpeed = 0f;
        if(Time.time >= nextAttackTime && readyToAttack){
            StartCoroutine("EnemyAttack");
            myAnim.SetTrigger("Attack");
            nextAttackTime = Time.time + 1f / attackRate;
        }
    }
    else
    {

```

```

        chaseSpeed = 0.5f;
    }
}

private void OnDrawGizmos() {
    Gizmos.DrawWireSphere(attackPoint.position, attackRange);
    Gizmos.DrawLine(groundDetection.position, new
Vector3(groundDetection.position.x, groundDetection.position.y - rayDistance));
    Gizmos.DrawLine(groundDetection.position, new
Vector3(groundDetection.position.x + rayDistance, groundDetection.position.y));
    Gizmos.DrawLine(fieldOfView.position, new Vector3(fieldOfView.position.x +
FOVDistance, fieldOfView.position.y));
    Gizmos.DrawLine(enemyDetection.position, new
Vector3(enemyDetection.position.x + rangeOfView, enemyDetection.position.y));
    Gizmos.DrawLine(enemyDetectionFromBack.position, new
Vector3(enemyDetectionFromBack.position.x - bckDetection,
enemyDetectionFromBack.position.y));
    Gizmos.DrawLine(firepoint.position, new Vector3(firepoint.position.x +
FOVDistance, firepoint.position.y));
}

IEnumerator EnemyAttack(){
    chaseSpeed = 0f;
    yield return new WaitForSeconds(0.75f);
    chaseSpeed = 0.5f;
}

void Shoot(){
    Instantiate(arrowPrefab, firepoint.position, firepoint.rotation);
}
}

```

A.2 Текст скрипта DroneController.cs

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

```

using UnityEngine;
using Pathfinding;

[RequireComponent(typeof(Rigidbody2D))]
[RequireComponent(typeof(Animator))]
public class DroneController : Character
{
    public Transform target;
    public Transform fieldOfView;
    public Transform groundDetection;
    RaycastHit2D WallInfo;
    RaycastHit2D GroundInfo;
    Seeker seeker;
    Path path;

    public float chaseSpeed = 200f;
    public float nextWaypointDistance = 3;
    public float FOVDistance = 2f;
    public float FOVRangeDistance = 3f;
    public float rayDistance;

    public int maxEnemyHealth = 100;
    int currentWayPoint = 0;

    protected bool readyToAttack;
    public bool enemyFound = false;
    public bool enemyIsDead = false;
    private bool movingRight = true;
    bool reachedEndOfPath = false;

```

```
[SerializeField] private bool patrolling = true;
```

```
public override void Start()
```

```
{
    Physics2D.IgnoreLayerCollision(6,8);
    Physics2D.IgnoreLayerCollision(6,9);
    seeker = GetComponent<Seeker>();
    rb2d = GetComponent<Rigidbody2D>();
    myAnim = GetComponent<Animator>();
    InvokeRepeating("UpdatePath", 0f, .5f);
    currentHealth = maxEnemyHealth;
    readyToAttack = true;
    facingRight = false;
    speed = chaseSpeed;

}
```

```
void UpdatePath() {
    if (seeker.IsDone()){
        seeker.StartPath(rb2d.position, target.position, OnPathComplete);
    }
}
```

```
void OnPathComplete(Path a){
    if(!a.error){
        path = a;
        currentWayPoint = 0;
    }
}

public override void Update()
```

```

{
    base.Update();
    if(Time.time >= nextAttackTime && reachedEndOfPath == true &&
readyToAttack == true && enemyFound){
        myAnim.SetTrigger("Attack");
        nextAttackTime = Time.time + 1f / attackRate;
    }
    if(currentHealth == 0f){
        enemyIsDead = true;
    }
    WallInfo = Physics2D.Raycast(groundDetection.position, Vector2.right,
rayDistance, whatIsGround);
    GroundInfo = Physics2D.Raycast(groundDetection.position, Vector2.down,
rayDistance, whatIsGround);
    if(GroundInfo.collider == true){
        grounded = true;
    }
}

public override void FixedUpdate()
{
    if (path == null){
        return;
    }
    if (currentWayPoint >= path.vectorPath.Count){
        reachedEndOfPath = true;
        return;
    }else{
        reachedEndOfPath = false;
    }
}

```

```

    if(enemyFound){
        Vector2 direction = ((Vector2)path.vectorPath[currentWayPoint] -
rb2d.position).normalized;
        Vector2 force = direction * chaseSpeed * Time.deltaTime;
        rb2d.AddForce(force);
    }
    float distance = Vector2.Distance(rb2d.position,
path.vectorPath[currentWayPoint]);
    if (distance < nextWaypointDistance){
        currentWayPoint++;
    }
    if(rb2d.velocity.x <= -0.01f){
        transform.localScale = new Vector3(-1f, 1f, 1f);
        facingRight = false;
    }else if(rb2d.velocity.x >= 0.01f){
        transform.localScale = new Vector3(1f, 1f, 1f);
        facingRight = true;
    }
}

void OnDrawGizmosSelected() {
    if(attackPoint == null){
        return;
    }
    Gizmos.DrawWireSphere(attackPoint.position, attackRange);
    Gizmos.DrawLine(groundDetection.position, new
Vector3(groundDetection.position.x, groundDetection.position.y - rayDistance,
groundDetection.position.z));
}

```

```
Gizmos.DrawLine(groundDetection.position, new  
Vector3(groundDetection.position.x + rayDistance, groundDetection.position.y,  
groundDetection.position.z));  
}  
}
```

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЗАПОРІЗЬКА ПОЛІТЕХНІКА»
КАФЕДРА ПРОГРАМНИХ ЗАСОБІВ

Дипломна кваліфікаційна робота бакалавра на тему «ПРОГРАМНА РЕАЛІЗАЦІЯ ВІДЕОГРИ В ЖАНРІ МЕТРОЇДВАНІЯ З ВИКОРИСТАННЯМ РУШІЯ UNITY»

Виконав: Кузнецов Віталій Олександрович, ст. гр. КНТ-227

Керівник: Зеленьова Ірина Яківна, к.т.н. доцент НУЗП

Рисунок Б.1 – Слайд №1

Об'єкт та предмет дослідження

Об'єкт дослідження - індустрія відеоігор

Предмет дослідження - методи реалізації відеогри в жанрі метроїдванія

Мета роботи - створення та тестування гри в жанрі метроїдванія на базі рушія Unity.

Виходячи з поставленої мети, можна виділити такі завдання:

- Проаналізувати схожі ігри цього жанру
- Створити загальну концепцію гри
- Реалізувати задумані ігрові механіки
- Протестувати готовий проект

Рисунок Б.2 – Слайд №2

Класифікація жанрів відеоігор

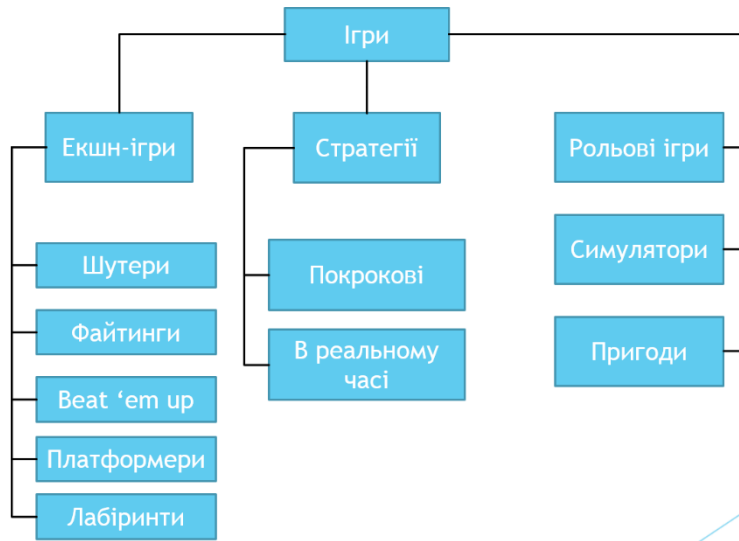


Рисунок Б.3 – Слайд №3

Ієрархія класів

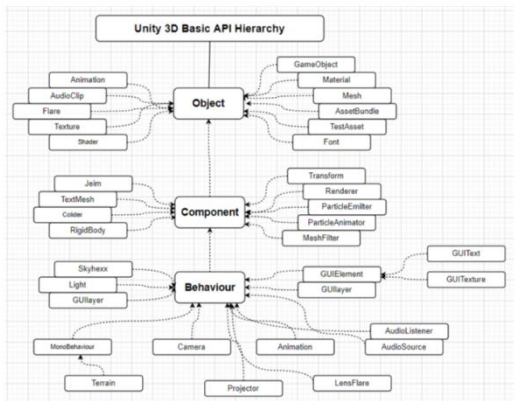


Рисунок 1 - Ієрархія класів рушія Unity

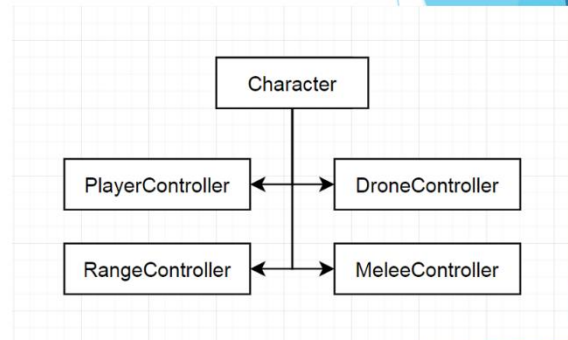


Рисунок 2 - Ієрархія класів гри Escape to Earth

Рисунок Б.4 – Слайд №4

Алгоритми пошуку цілі

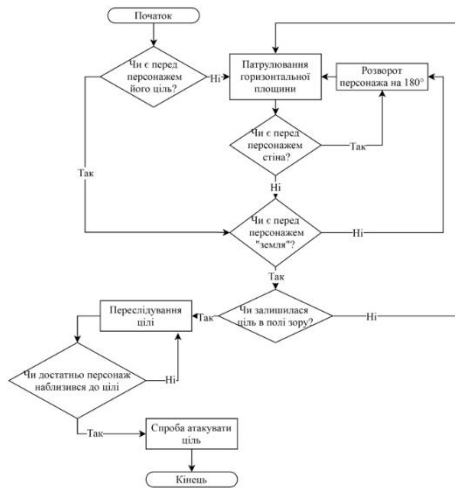


Рисунок 3 - Пошук цілі наземними ворогами

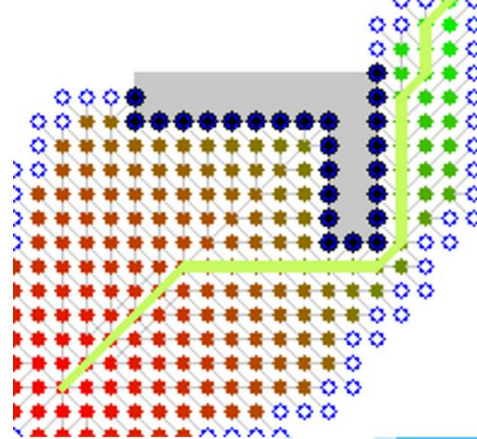


Рисунок 4 - Алгоритм пошуку шляху A* для літаючих супротивників

5

Рисунок Б.5 – Слайд №5

Візуальний дизайн меню в грі



Рисунок 5 - Головне меню гри



Рисунок 6 - Меню паузи в грі

6

Рисунок Б.6 – Слайд №6

Візуальний дизайн персонажів гри



Рисунок 7 - Зовнішній вигляд головного героя



Рисунок 8 - Зовнішній вигляд наземного ворога



Рисунок 9 - Зовнішній вигляд літаючого ворога



7

Рисунок Б.7 – Слайд №7

Зовнішній вигляд локацій



Рисунок 10 - Скріншот ігрового процесу



Рисунок 11 - Скріншот ігрового процесу

8

Рисунок Б.8 – Слайд №8

Висновок

Під час виконання дипломної роботи була розроблена та піддана тестуванню відеогра в жанрі метроїдванія, що має назву «Escape to Earth». При створенні проекту були реалізовані:

- Механіка пересування по локації
- Бойова система
- Механіка стрибків від стін та ривків на землі й в повітрі
- Графічний інтерфейс користувача

Створена для дипломного проекту гра є унікальною, бо все в ній починаючи з графіки та закінчуючи скриптами було розроблене власноруч. Окрім того, для виконання роботи були виконані наступні задачі:

- Досліджені аналогічні проекти в жанрі метроїдванія
- Створена загальна концепція гри
- За допомогою мови програмування C# реалізовані основні ігрові механіки
- Проведено функціональне та юзабіліті-тестування серед цільовою аудиторії

9

Рисунок Б.9 – Слайд №9

Дякую за увагу!

Рисунок Б.10 – Слайд №10