Міністерство освіти і науки України
Одеський національний технологічний університет
Кафедра комп'ютерної інженерії

ПОЯСНЮВАЛЬНА ЗАПИСКА ДО КВАЛІФІКАЦІЙНОЇ РОБОТИ

на тему *Проектування та розробка гри жанру "Метроідванія"*
(назва кваліфікаційної роботи згідно наказу ОНТУ)

Здобувача *Носаля В. М.*
(прізвище, ініціали)

2 курсу *543А* групи

Керівники: *д.т.н., проф. Артеменко С.В.*
асистент Орел А.С.
(посада, прізвище та ініціали)

Консультанти: *д.т.н., проф. Басюркіна Н.Й.*
(посада, прізвище та ініціали)

Кваліфікаційна робота допускається до захисту

Рішення кафедри від *05.06.2024* протокол № *8*

Завідувач кафедри комп. інженерії _____ Сергій АРТЕМЕНКО
(назва кафедри) (підпис) (Ім'я ПРІЗВИЩЕ)

Одеса – 2024 рік

ОДЕСЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНОЛОГІЧНИЙ УНІВЕРСИТЕТ

Факультет *комп'ютерної інженерії, програмування та кіберзахисту*
Кафедра *комп'ютерної інженерії*
Ступінь вищої освіти *бакалавр*
Спеціальність *123 «Комп'ютерна інженерія»*
Освітня програма *Мережеві технології та інтернет речей*

ЗАТВЕРДЖУЮ

Зав. кафедри *комп'ютерної інженерії*
Сергій АРТЕМЕНКО
« 30 » *серпня* 2023 року

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ ЗДОБУВАЧА
Носаля Віктора Михайловича

1. Тема роботи *Проектування та розробка гри жанру "Метроідванія"*

Затверджена наказом університету від « *30* » *серпня* 2023 р., наказ № *442-03*

2 Термін здачі здобувачем закінченої роботи

3. Вихідні дані роботи

1. Редактор «Microsoft Office». 2. Редактор «PyCharm» 3.Ідея

4. Перелік питань, які потрібно розробити

1. Вступ 2. Збір та аналіз роботи 3. Розробка концептуального документу
4. Реалізація графічної частини 5. Висновки

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

Слайд 6 та 7. Аналоги, Слайд 8. Концепт гри, Слайд 10. Рівні, Слайд 11. Персонажи, Слайд 14. Діаграми класів, Слайд 15. Розробка

6. Консультанти по роботі, із зазначенням розділів роботи, що стосуються їх

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
Економіка	*д.т.н., проф. Басюркіна Н.Й.*		
Охорона праці	*д.т.н., проф. Артеменко С.В.*		
Нормоконтроль	*д.т.н., проф. Артеменко С.В.*		

7. Дата видачі завдання *30.08.2023*

Керівники *Сергій АРТЕМЕНКО*

Завдання прийняв до виконання *Андрій ОРЕЛ*

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів кваліфікаційної роботи	Термін виконання етапів роботи	Примітка
1.	*Аналіз предметної області та дослідження існуючих аналогів*	*24.10.2023*	
2.	*Проектна/Дослідницька частина*	*30.11.2023*	
3.	*Практична реалізація та результати дослідження*	*25.01.2024*	
4.	*Техніко-економічна частина*	*13.02.2024*	
5.	*Кодування*	*28.03.2024*	
6.	*Економічні розрахунки*	*12.04.2024*	
7.	*Підготовка розділу охорони праці*	*15.04.2024*	
8.	*Оформлення пояснювальної записки*	*22.05.2024*	
9.	*Підготовка графічної частини та вихідного коду*	*13.05.2024*	
10.	*Захист кваліфікаційної роботи*	*21.06.2024*	

Здобувач – дипломник *Віктор НОСАЛЬ*

Керівники роботи *Сергій АРТЕМЕНКО*

Андрій ОРЕЛ

Несу відповідальність за ідентичність електронного та друкованого варіантів кваліфікаційної роботи, даю згоду на обробку персональних даних та не заперечую проти розміщення кваліфікаційної роботи на офіційних web-ресурсах ОНТУ.

Підтверджую, що в кваліфікаційній роботі відсутні порушення норм академічної доброчесності.

Здобувач – дипломник *Віктор НОСАЛЬ*

АНОТАЦІЯ

Ця кваліфікаційна робота присвячена проектуванню та розробці гри жанру Метроідванія.

У роботі розглядаються основні конструктивні, технологічні та техніко-експлуатаційні характеристики гри, а також рекомендації щодо її впровадження. Основна увага приділена використанню методів програмування таких як мова Python, бібліотека PyGame та редактор карт Tiled. Робота також включає об'єктно-орієнтований підхід та патерн проектування Model-View-Controller (MVC), що забезпечує модульність та легкість у підтримці та розширенні функціоналу гри.

В процесі розробки було застосовано методи модульного та інтеграційного тестування, а також методологію Agile для управління проектом.

В першому розділі розглянуті ключові особливості та характеристики ігор жанру метроідванія, визначені базові механіки та проведено аналіз існуючих аналогів. В другому розділі сформовано засади концептуального та дизайнерського документу. У третьому розділі були обрані та обгрунтовані засоби розробки, визначена реалізація механік, а також продемонстрована розробка демонстраційної версії гри. У четвертому розділі проведена оцінка ефективності розробки гри, описано маркетинговий, науково-технічний, економічний, соціальний ефект від розробки проекту. У п'ятому розділі розглянуто питання охорони праці.

Обсяг пояснювальної записки складає 116 сторінки.

Ключові слова: Метроідванія, розробка гри, Python, PyGame, об'єктно-орієнтоване програмування, MVC, Agile.

ABSTRACT

This qualification thesis is dedicated to the design and development of a Metroidvania genre game.

The thesis addresses the main structural, technological, and technical-operational characteristics of the game, as well as recommendations for its implementation. Particular attention is paid to the use of programming methods such as Python, the PyGame library, and the Tiled map editor. The work also includes an object-oriented approach and the Model-View-Controller (MVC) design pattern, which ensures modularity and ease of maintenance and extension of the game's functionality.

During development, methods of modular and integration testing were applied, as well as the Agile methodology for project management. The first chapter examines the key features and characteristics of Metroidvania genre games, identifies basic mechanics, and conducts an analysis of existing analogs. The second chapter forms the basis of the conceptual and design document. The third chapter selected and justified the development tools, defined the implementation of mechanics, and demonstrated the development of a demo version of the game. The fourth chapter assesses the effectiveness of game development, describes the marketing, scientific-technical, economic, and social impact of the project. The fifth chapter addresses occupational safety issues.

The explanatory note is 116 pages.

Keywords: Metroidvania, game development, Python, PyGame, object-oriented programming, MVC, Agile.

ЗМІСТ

АНОТАЦІЯ 1
ABSTRACT 2
ВСТУП 3
РОЗДІЛ 1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ДОСЛІДЖЕННЯ ІСНУЮЧИХ АНАЛОГІВ 10
1.1 Аналіз предметної області 10
1.2. Аналіз публікацій останніх років 14
1.3. Дослідження існуючих аналогів 16
1.4. Розробка технічного завдання 27
РОЗДІЛ 2. ПРОЕКТНА/ДОСЛІДНИЦЬКА ЧАСТИНА 31
2.1. Розробка концептуального документу 31
2.1.1. Загальна характеристика проекту 31
2.1.2. Опис світу та сюжету гри 32
2.1.3. Геймплей, механіки, жанр 34
2.2. Розробка дизайн-документа 35
2.2.1. Дизайн локацій та рівнів 35
РОЗДІЛ 3. ПРАКТИЧНА РЕАЛІЗАЦІЯ ТА РЕЗУЛЬТАТИ ДОСЛІДЖЕННЯ 42
3.1 Вибір і обґрунтування програмних засобів реалізації проекту (графічної та програмної частин) 42
3.1.1 Програмна частина 44
3.1.1 Графічна частина 46
3.2 Реалізація графічної частини 49
3.2.1 Створення сцени 49
3.2.2 Створення персонажів 51
3.3 Реалізація основного функціоналу коду 52
3.4 Тестування та відлагодження ігрової системи. 58
РОЗДІЛ 4. ТЕХНІКО-ЕКОНОМІЧНА ЧАСТИНА 62
4.1 Вибір та обґрунтування проектних рішень 42

					КРБ.КІ.1.442-03.4.9		
Змн.	Арк.	№ докум.	Підпис	Дата			
Розробив		Віктор НОСАЛЬ			*Проектування та розробка гри жанру "Метроідванія"*	Літ.	Арк. / Акрушів
Перевірив		Сергій АРТЕМЕНКО					
Рецензент		Андрій Севостьяненко				*гр. 543а, ОНТУ*	
Нормоконтроль		Сергій АРТЕМЕНКО					
Затвердив		Сергій АРТЕМЕНКО					

4.2 Економічна ефективність проекту 62
4.3 Ризики та способи їх мінімізації 69
РОЗДІЛ 5. ОХОРОНА ПРАЦІ 73
5.1 Організація робочого місця 73
5.2 Режим праці та відпочинку 74
5.3 Дотримання правил техніки безпеки 74
ВИСНОВКИ 76
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 81
ДОДАТКИ 85
Додаток А 86

ВСТУП

Актуальність теми. Сучасний стан ігрової індустрії свідчить про зростаючу популярність жанру Метроідванія. Цей жанр, поєднуючи елементи платформерів та рольових ігор, надає гравцям можливість досліджувати великі нелінійні світи, збирати різноманітні предмети та покращувати здібності персонажа. В останні роки випущено багато успішних проектів у цьому жанрі, таких як Hollow Knight, Ori and the Blind Forest, та Axiom Verge. Вони отримали високі оцінки критиків та гравців за інноваційний підхід до механік гри, чудову графіку та захопливий сюжет. Проте, незважаючи на успіхи цих ігор, ринок все ще залишається відкритим для нових інноваційних проектів, які можуть привнести щось нове у жанр.

Актуальність теми проектування та розробки гри жанру Метроідванія обумовлена не лише зростаючою популярністю цього жанру, але й необхідністю вирішення технічних та ігрових проблем, з якими стикаються розробники. Аналіз існуючих аналогів показує, що багато ігор стикаються з проблемами оптимізації, багів та недостатньої інноваційності. Хоча сучасні розробки демонструють високий рівень якості, багато проектів все ще потребують удосконалення у таких аспектах, як інтерактивність, глибина сюжету та унікальність ігрових механік. Прогалини знань у цій галузі та технічні протиріччя створюють сприятливі умови для проведення нових досліджень та експериментів.

Мета роботи. Метою даної кваліфікаційної роботи є вирішення проблемної ситуації шляхом аналізу існуючих рішень та пошуку нових методів, засобів, алгоритмів та моделей для створення інноваційної гри жанру Метроідванія. Використання сучасних інформаційних технологій дозволить розробити гру, яка буде відповідати вимогам ринку та потребам гравців, при

цьому уникаючи типових проблем, з якими стикаються інші розробники.

Об’єктот дослідження є процес розробки комп'ютерної гри жанру Метроідванія, що включає проектування, програмування, тестування та випуск готового продукту.

Предмет дослідження виступають методи вирішення конкретних проблем розробки ігор, зокрема інтеграція ігрових механік, оптимізація продуктивності, покращення графіки та звукового оформлення, а також розробка унікальних сюжетних ліній та персонажів.

Для досягнення мети роботи визначено такі задачі:

– Розробка концепції та сюжету гри;
– Створення графічного та звукового оформлення;
– Реалізація ігрового рушія та логіки;
– Розробка механік та рівнів гри;
– Здійснення тестування і усунення помилок;
– Випуск демо-версії гри.

Методи розробки включають:

1) Використання мови програмування Python;
2) Бібліотеки PyGame для роботи з графікою і звуком;
3) Редактора карт Tiled.

Наукова новизна одержаних результатів полягає у інтеграції інноваційних методів оптимізації продуктивності та створенні унікального сюжету, який відрізняв би проект від сучасних аналогів.

Практичне значення одержаних результатів полягає у можливості використання розробленої гри як базової платформи для подальших досліджень та вдосконалень, а також у її комерційному потенціалі.

РОЗДІЛ 1

АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ДОСЛІДЖЕННЯ ІСНУЮЧИХ АНАЛОГІВ

1.1 Аналіз предметної області

Індустрія відеоігор є однією з найбільш динамічних і швидкозростаючих галузей розваг у світі. Вона охоплює величезну кількість жанрів, платформ і цільових аудиторій, пропонуючи безліч унікальних ігрових досвідів. Серед цього різноманіття особливе місце посідає жанр «Метроідванія», який поєднує в собі елементи пригодницьких ігор, екшн-платформерів і головоломок.

Історія жанру «Метроідванія» починається в 1980-х роках, коли індустрія відеоігор переживала справжній ренесанс. Після важкої кризи на початку десятиліття, спричиненої перенасиченням ринку низькоякісними іграми, з'явилися нові потужні консолі та інноваційні ідеї, що підняли галузь на новий рівень. В епіцентрі цього відродження опинилася японська компанія Nintendo зі своєю революційною консоллю NES (Nintendo Entertainment System). Випущена в 1983 році, NES стала справжнім феноменом, завоювавши мільйони шанувальників завдяки культовим іграм, таким як «Super Mario Bros.», «The Legend of Zelda» та «Metroid».

Саме «Metroid», розроблена талановитим геймдизайнером Йосіо Сакамото і випущена в 1986 році, заклала основи жанру «Метроідванія». Ця гра стала справжнім проривом у світі платформерів, пропонуючи нелінійний, лабіринтоподібний дизайн рівнів замість типової послідовної прогресії.

Гравці взяли на себе роль Самус Аран - могутньої мисливиці за головами в футуристичному костюмі, яка висадилася на планету Зебес з метою знищити космічних піратів і страхітливих істот, що там оселилися. Однак, на відміну від інших платформерів, де рівні проходилися по черзі, гравцям довелося самостійно шукати шлях у величезному взаємопов'язаному світі, наповненому сховищами, таємницями та моторошними підземеллями.

Щоб просуватися далі, Самус мала збирати нові здібності та зброю, такі як ракетна установка, гравітаційна куля чи можливість ковзати під низькими перешкодами. Кожна нова здатність відкривала доступ до раніше недосяжних областей, змушуючи гравців повертатися до вже відвіданих територій і по-новому досліджувати їх.

Ця нелінійна концепція «розблокування» і «дослідження» стала визначальною рисою жанру «Метроідванія». Гравці були змушені ретельно вивчати кожен куточок величезного світу, щоб знайти сховані секрети, енергетичні поповнення та шляхи до прогресу. «Metroid» ламала усталені канони платформерів, пропонуючи безпрецедентну свободу пересування та занурюючи гравців у атмосферу самотнього дослідження таємничої чужопланетної локації.

Незважаючи на інноваційний дизайн, «Metroid» не стала суперпопулярною грою відразу після релізу. Її успіх був більш стриманим, ніж у «Super Mario Bros.», частково через складність та відсутність персонажного полотна в рекламній кампанії. Тим не менш, вона здобула культовий статус і лояльну базу шанувальників, які високо оцінили її атмосферу, нелінійність та виклики.

Справжній ренесанс жанр «Метроідванія» пережив більш ніж десятиліття по тому, коли Конамі випустила легендарну гру «Castlevania: Symphony of the Night» в 1997 році. Ця гра не лише стала визначною подією в серії «Castlevania», а й ознаменувала нову еру для жанру загалом.

«Symphony of the Night» зберегла основоположні принципи «Metroid»: нелінійний дизайн рівнів, розблоковані здібності для доступу до нових територій та акцент на дослідженні. Але водночас вона додала ще більше глибини та атмосфери до цієї формули.

Гра розгорталася у похмурому готичному фентезійному світі. Протагоніст Ален - спадкоємець лінії Дракули, який прокинувся через чотири роки після подій попередньої гри. Замок Дракули знову виринув з туманів, і Алену належало дослідити його лабіринти, підземелля та сховані секрети.

На своєму шляху Ален здобував надзвичайні здібності, такі як перетворення на туман, підіймання важких предметів чи випуск смертоносних магічних куль. Кожна нова сила відкривала йому доступ до раніше недосяжних ділянок замку.

Проте «Symphony of the Night» відрізнялася не лише новими механіками та здібностями, а й надзвичайно багатою атмосферою та деталізованою розповіддю історії. Сюжетна лінія розкривалася поступово, через випадкові діалоги з деякими персонажами чи записки в сховищах. Світ гри був густо наситений готичними мотивами, моторошними істотами та похмурою красою, занурюючи гравців у справжню атмосферу пригод.

Успіх «Symphony of the Night» був приголомшливим, і гра стала культовою класикою. Вона об'єднала найкращі елементи «Метроіду» та додала до них неперевершену естетику, оповідь та незабутню атмосферу. Ця гра стала взірцем для наступних представників жанру «Метроідванія».

Зараз однією з найбільш характерних рис жанру «Метроідванія» є нелінійний, лабіринтоподібний дизайн ігрового світу. На відміну від лінійних платформерів, де рівні проходяться послідовно, в «Метроідваніях» гравець має свободу пересуватися величезною, взаємопов'язаною локацією, досліджуючи її закутки та таємниці у будь-якому порядку. Однак, доступ до певних областей спочатку обмежений перешкодами, які неможливо подолати без спеціальних здібностей.

Ця концепція «відкриття та розблокування» лежить в основі геймплею «Метроідванія». Гравець постійно стикається з недосяжними областями, загадковими дверима чи шляхами, заблокованими перешкодами. Проте, після отримання нової здібності, гравець може повернутися до цих місць і продовжити дослідження, відкриваючи нові секрети, битви з босами та сюжетні фрагменти. Ця циклічна природа геймплею породжує відчуття постійного прогресу та винагороди, надихаючи гравця ретельно вивчати кожен куточок ігрового світу у пошуках прихованих скарбів, сил-предметів і таємниць. Отримання нових здібностей не лише розширює можливості

персонажа, а й змінює саме сприйняття ігрового простору, змушуючи гравця по-новому поглянути на вже відвідані місця.

Важливою складовою «Метроідванія» є атмосфера та оповідання історії. Більшість ігор цього жанру мають захоплюючий науково-фантастичний або темний фентезійний сеттинг, насичений таємницями та загадками. Сюжетна лінія часто розкривається фрагментарно, через знахідки гравця - записки, артефакти чи взаємодію з незначними персонажами. Ці підхідо залишають багато простору для роздумів і припущень, занурюючи гравця в атмосферу пригод та самотнього дослідження.

Незважаючи на свою специфічну природу, жанр «Метроідванія» став популярним серед розробників та шанувальників завдяки своїй унікальній концепції нелінійного дизайну рівнів, акценту на дослідженні та отриманні нових здібностей. Ігри цього жанру пропонують захопливий досвід відкриття таємниць, вирішення головоломок і безперервного розширення можливостей гравця в межах єдиного, взаємопов'язаного світу. Але, розробка якісної «Метроідванії» є складним і трудомістким завданням, що вимагає ретельного планування, зваженого дизайну рівнів та ігрового балансу. Одна з основних проблем, з якою стикаються розробники, - це забезпечення відчуття прогресу та винагороди в нелінійному ігровому просторі.

На відміну від лінійних ігор, де нові виклики та здібності представляються послідовно, в «Метроідванії» гравець може випадково натрапити на складні перешкоди чи могутніх ворогів дуже рано. Якщо ці елементи не будуть вдало збалансовані, гравець може відчути безпорадність або, навпаки, легко подолати виклики, втративши відчуття прогресу. Тому створення вдалої прогресії складності та розміщення перешкод, здібностей і ворогів у правильних місцях має вирішальне значення для забезпечення захопливого та збалансованого ігрового досвіду.

Проблемою також є уникнення змушеного бекгтрекінгу (повернення в попередні локації). Оскільки гравець постійно отримує нові здібності, що розблоковують доступ до раніше недосяжних місць, існує ризик занадто часто

примушувати його повертатися на старі локації. Це може призвести до відчуття повторюваності та нудьги. Тому розробники повинні ретельно планувати розміщення нових можливостей та секретів, щоб зберегти баланс між дослідженням нових територій і повторним відвідуванням старих.

Складність полягає також у створенні органічного, переконливого ігрового світу, який би надихав гравця на дослідження. Кожна локація повинна бути ретельно продумана, з власною атмосферою, загадками та секретами, які заохочують ретельну розвідку. Водночас, світ має бути логічно влаштований та зв'язаний, щоб уникнути плутанини чи відчуття штучності.

Також проблемою для розробників є забезпечення високої якості виробництва на всіх рівнях розробки:

- дизайну рівнів;
- художнього оформлення;
- програмування та звукового супроводу.

Оскільки «Метроідванія» зазвичай являє собою великий, взаємопов'язаний ігровий світ, обсяг роботи є значним, а витримати високі стандарти якості на всіх етапах створення гри є непростим завданням. Деталізоване художнє оформлення, плавна анімація, атмосферний саундтрек і безперебійна робота ігрового рушія мають вирішальне значення для занурення гравця в атмосферу та надання йому незабутніх вражень від дослідження ігрового світу.

1.2. Аналіз публікацій останніх років

Жанр «Метроідванія» продовжує привертати значну увагу як з боку шанувальників відеоігор, так і науковців та дослідників. Протягом останніх п'яти років з'явилося чимало публікацій, що аналізують різні аспекти цього унікального ігрового жанру, починаючи від його історії та впливу на індустрію до специфічних дизайнерських рішень та нових підходів до розробки.

Одним із ґрунтовних досліджень історії та еволюції жанру «Метроідванія» є стаття «Ретроспектива жанру «Метроідванія»: від

«Метроіду» до «Орі та сліпучої Волі» авторів Сари Арройо та Хосе Л. Гонсалеса, опублікована в науковому журналі «Трансакції з інтерактивних систем» (ACM Transactions on Interactive Intelligent Systems) у 2019 році. У своїй роботі дослідники детально розглядають витоки жанру, його визначальні ознаки та еволюцію протягом десятиліть, аналізуючи такі визначні ігри, як «Metroid», «Castlevania: Symphony of the Night», «Shadow Complex» та «Ori and the Blind Forest». Вони також вивчають вплив цих ігор на індустрію та їхню роль у популяризації нелінійного дизайну рівнів і концепції «розблокування й відкриття» [1].

Дослідження специфічних дизайнерських рішень та підходів, що використовуються в іграх жанру «Метроідванія», є ще одним популярним напрямком досліджень. Наприклад, у статті «Дизайн рівнів у «Метроідваніях»: просторовий аналіз та рекомендації» (опублікованої в журналі «Трансакції з інтерактивних систем» у 2021 році) автори Джеремі Бургер, Джин Кван та Меліса Лопріор вивчають особливості дизайну рівнів у цьому жанрі, зосереджуючись на таких аспектах, як зв'язність ігрового простору, розміщення перешкод і розблокованих здібностей, а також способи заохочення дослідження. Дослідники пропонують конкретні рекомендації для розробників, щоб гармонійно поєднати ці елементи та створити захопливий та збалансований ігровий досвід [2;6;7].

Ще однією важливою темою, яку висвітлюють вчені, є роль оповідання історії та створення атмосфери в іграх жанру «Метроідванія». У статті «Розповідь історій у «Метроідванії»: фрагментація, невизначеність та занурення» (опублікованій у збірнику «Нарація в інтерактивних медіа» у 2020 році) автор Емілія Родрігес досліджує, як ці ігри використовують фрагментарний і загадковий підхід до розкриття сюжетної лінії, залишаючи багато простору для роздумів та інтерпретацій гравців. Дослідниця аналізує конкретні приклади з ігор, таких як «Hollow Knight» та «Dead Cells», показуючи, як цей підхід посилює занурення в ігровий світ і створює атмосферу таємниці та самотнього дослідження [3;8].

Окрім наукових публікацій, жанр «Метроідванія» також став предметом обговорення в численних аналітичних статтях та оглядах на відомих ігрових порталах та у фахових виданнях, присвячених відеоіграм. Наприклад, у статті «Вічна чарівність «Метроідваній» на сайті Polygon (2022 рік) автор Кріс Пліні аналізує, чому ці ігри продовжують захоплювати гравців попри свій вік, зосереджуючись на елементах дизайну, що заохочують дослідження та відчуття прогресу [4;5].

1.3. Дослідження існуючих аналогів

За багато років розвитку жанру Метроідванія він зберіг основні елементи, проте постійно вдосконалювався, переходячи від двовимірних до тривимірних світів. Розміри ігрових всесвітів та складність механіки зростали від гри до гри. Найпопулярнішими та найвисоко оціненими гравцями на даний момент є такі ігри:

Super Metroid, яка встановила базові стандарти жанру.

Super Metroid - це одна з найвизначніших грів для SNES, розроблена studio Nintendo R&D1 та видана 1994 року. Гра стала прямим сиквелом до оригінальної Metroid 1987 року та Metroid II: Return of Samus 1991 року для Game Boy.

Super Metroid встановила базові концепції та елементи, які згодом стали фундаментальними для жанру метроідванії. Гравцеві доводиться досліджувати величезний, ретельно розроблений та пов'язаний світ планети Зебес.

З кожним новим відкриттям простір стає все більш доступним. Гравцеві доводиться розв'язувати головоломки, відкривати нові райони завдяки отриманим здібностям та покращенням, знищувати босів та вивчати історію планети.

Це створює почуття задоволення та занурення в містичний світ гри. Високоякісна художня стилізація, захоплююча музика та атмосферні звукові ефекти створюють незабутню геймплейну експерієнцію.

Найбільше досягненням Super Metroid стала можливість нелинійного дослідження відкритого світу та його поступового відкриття завдяки отриманим здібностям головного персонажа - Семус Аран.

Гравці можуть експериментувати з різними шляхами проходження та самостійно визначати послідовність подій. Це стало фундаментальним елементом метроідванії.

Також Super Metroid встановила стандарти для дизайну рівнів, розташування секретів, прихованих районів та міні-босів. Її атмосферний стиль оповідання за допомогою візуальних та звукових ключів проклав шлях для багатьох майбутніх ігор жанру.

З усіх елементів, Super Metroid зробила найбільший акцент на дослідженні, знаходженні секретів та відчутті задоволення від їх відкриття. Це змушувало гравців переглядати кожен куточок світу.

Таємнича атмосфера, складний дизайн рівнів з численними секретами та опціональними шляхами, 16-бітна графіка - все це зробило Super Metroid шедевром жанру та вартим наслідування прикладом для майбутніх ігор-метроідваній. Саме ця гра задала той стандарт, до якого довгий час прагнули дотягнутися розробники подібних проєктів [9].

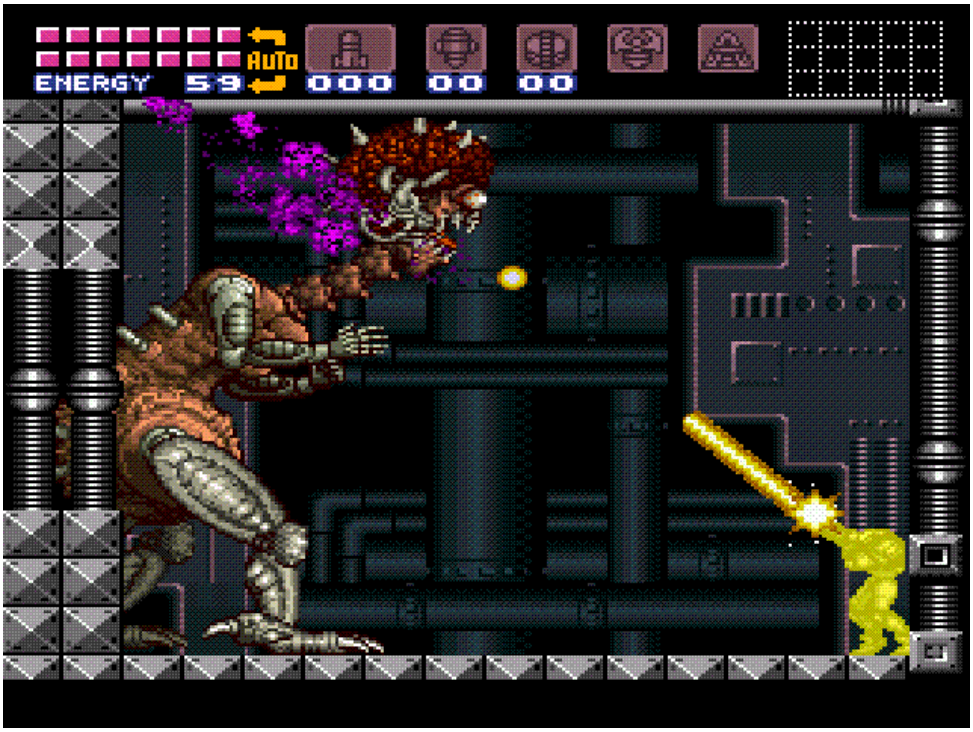

Рис. 1.3 – Ігровий процес «Super Metroid»

Super Mario Bros. Wonder - покращена версія старої класичної гри Super Mario.

Super Mario Bros. вперше вийшла 1985 року на Nintendo Entertainment System і революціонізувала жанр платформерів. Гра швидко здобула популярність завдяки чудовій геймплейній механіці, яскравій візуальній складовій та гумористичному сценарію. Понад 30 років по тому ця класична гра продовжує впливати на подальший розвиток інтерактивної розваги.

У 2023 році компанія Nintendo випустила покращену версію оригінальної Super Mario Bros. під назвою Super Mario Bros. Wonder. Дана гра була розроблена з метою поєднання класичного ігрового процесу 1985 року з сучасними технологіями та функціями. Nintendo зуміла вдосконалити ігровий досвід, не позбавляючи гру її ретро-чарівності.

Серед найпомітніших оновлень Super Mario Bros. Wonder слід виділити наступні складові: покращену графіку вищої роздільної здатності, яка дозволяє побачити культовий світ Mushroom Kingdom зовсім з іншої сторони; опцію збереження прогресу та переносу збережень між різними пристроями для більшої зручності користувача; розширення ігрового процесу шляхом

додавання нових рівнів та секретів; удосконалену музичний супровід з новими треками та реміксами класичних мелодій.

Попри те, що потужніша графіка та збереження прогресу роблять гру більш доступною для сучасних гравців, її основний геймплей залишився практично незмінним. Гравці досі керують Маріо чи Луїджі, які проходять рівні, збирають монетки, поролають гриби та перемагають ворогів у пошуках рятування принцеси Піч. Це дає змогу ностальгійно пережити радість від оригінальної гри, а також дарує неповторне задоволення від відкриття нових моментів.

Будучи продовженням і поєднанням класики та сучасних стандартів, Super Mario Bros. Wonder створює ідеальну основу для залучення до світу відеоігор як нових, так і колишніх геймерів. Це оновлення зберігає турадичінні цінності та головну ідею оригінальної Super Mario Bros., збільшуючи радість від проходження легендарної гри для сучасного покоління. Таким чином, гра сприяє подальшому розвитку жанру платформерів, демонструючи, як оновлення класики можуть удосконалити ігровий досвід за допомогою новітніх інновацій.

Сучасні технології також дозволили розширити соціальний аспект в Super Mario Bros. Wonder. Створення онлайн-рангів та системи досягнень сприяло розвитку спільноти і конкуренції. Гравці можуть порівнювати свої результати, ділитися досягненнями в соціальних мережах, а також брати участь у спільних змаганнях. Це робить гру більш цікавою та захопливою для сучасних користувачів, які цінують можливість взаємодіяти з іншими людьми під час ігрового процесу [10].

Рис. 1.2 – Ігровий процес «Super Mario Bros. Wonder»

ENDER MAGNOLIA: Bloom in the Mist – сучасна гра, в якій є пригода, магія, містика, дослідження, RPG.

Гра ENDER MAGNOLIA: Bloom in the Mist вперше вийшла у 2023 році для персональних комп'ютерів та консолей нового покоління. Розроблена однією з малих незалежних студій, вона стала несподіваним хітом, зібравши позитивні відгуки критиків та численну аудиторію гравців.

ENDER MAGNOLIA належить до жанру JRPG (Japanese Role Playing Game) та розвиває його традиційні механіки. Гравці беруть управління за молодою дівчиною Магнолією, яка опиняється у фентезійному світі, заповненому чудовими місцями та небезпечними створіннями. Проходячи квести, досліджуючи локації та борючись з ворогами, героїня поступово розкриває свої магічні здібності та дізнається про своє призначення.

Гра виділяється яскравою анімаційною складовою, продуманими боями в реальному часі та численними нелінійними квестами. Супровід саундтреків доповнює захоплюючий світ ENDER MAGNOLIA. Незважаючи на бюджетне виробництво, рівень опрацювання деталей та розмаїття контенту ставить

цю гру на рівень блокбастерів від великих компаній.

Завдяки сучасній оптимізації вона стає доступною для гравців різних платформ. ENDER MAGNOLIA з успіхом використовує найкращі напрацювання Японської РПГ-школи та популяризує її в нових регіонах. Водночас вона вносить помітні вдосконалення у жанр, збагачуючи його свіжими ідеями [11].

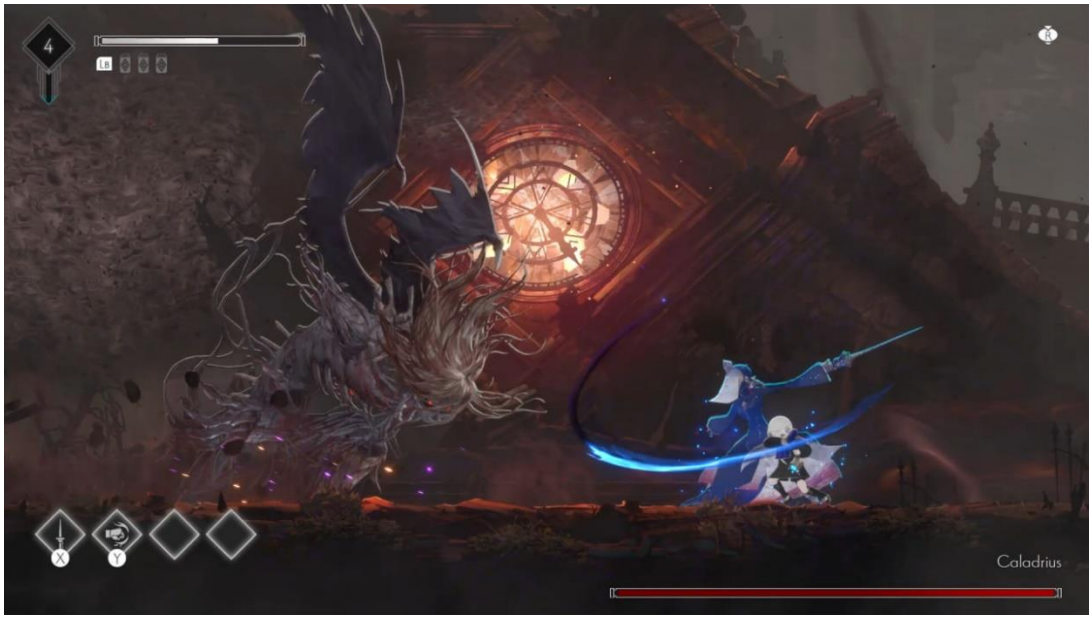

Рис. 1.3 – Ігровий процес «ENDER MAGNOLIA: Bloom in the Mist»

Ori and The Blind Forest – унікальна подорож по чарівному лісу Нібель.

Гра Ori and The Blind Forest вийшла 2015 року та одразу здобула популярність завдяки своїй захопливій атмосфері та геймплею. Розроблена невеликою американською студією Moon Studios, вона довела жанр платформера до нового рівня за допомогою чудової візуальної складової.

У ролі духа Орі гравці досліджують загадковий ліс Нібель, відновлюючи його колишню красу. Яскрава та деталізована графіка передає таємничість цього чарівного світу. Пройшовши численні динамічні рівні, герой навчається новим навичкам, розгадуючи загадки древніх руїн.

Захоплююча система пересування дозволяє Орі літати, стрибати, бігати по стінах та використовувати особливі здібності. Пройшовши гру, гравці відкривають усі таємниці минулого лісу та розуміють вагоме призначення героя. Прекрасний супровідний саундтрек тільки посилює ефект присутності.

Завдяки вдалим інноваціям Ori and The Blind Forest стала еталоном для подальшого розвитку жанру платформерів. Гра довела, як за допомогою яскравої візуалізації можна створити глибокий ігровий світ, здатний захопити гравців на роки. Саме завдяки таким творам жанр продовжує еволюціонувати та підкорювати нові аудиторії.

Рис. 1.4 – Ігровий процес «Ori and The Blind Forest»

1. Salt and Sanctuary - чистокровна суміш Dark Souls та Castlevania: Symphony of the Night, в якій більше від першого з батьків.

Гра Salt and Sanctuary, розроблена невеликою незалежною командою Ska Studios, вийшла 2016 року та одразу привернула увагу шанувальників жанру souls-like. Запозичивши основні механіки у культової серії Dark Souls, вона поєднала їх з елементами metroidvania та отримала сильну прихильну аудиторію.

Salt and Sanctuary переносить гравців у світ, де море поглинуло землю. Керований ними лицар досліджує руїни залишених споруд та островів. Графіка виконана у стилі аніме з використанням піксель-арта, що додає їй особливого шарму. Як і в Dark Souls, тут доводиться боротися зі складними ворогами та босами, уникаючи одного удару смерті.

Цікавим оновленням жанру стала можливість вибирати одну з кількох класів персонажа та розвивати його унікальні вміння. Подорожуючи затопленими землями, гравці відкривають нові райони, зброю та предмети, допомагаючи персонажу в боротьбі.

Характерною рисою Salt and Sanctuary стала наявність інших гравців, з якими можна об'єднуватись для проходження чи битися в PvP-боях. Протягом подорожі вдало вплітаються незалежні сюжетні квести. Атмосферний саундтрек ще більше підсилює похмурість світу.

Незважаючи на обмежений бюджет, рівень опрацювання деталей у Salt and Sanctuary залишається на зразковому рівні. Графіка, звукове та ігрове оформлення вдало поєднують в собі найкращі елементи обох батьківських франшиз. Це дозволило проєкту невеликої команди здобути визнання як серед критиків, так і серед шанувальників жанру.

Salt and Sanctuary успішно розвинула жанр soulslike, додавши в нього нові глибини за допомогою metroidванійних складових та ігрового процесу в аніме-стилі. Незважаючи на брак фінансових можливостей, творці зуміли створити неперевершений для інді-проєкту тайтл, здатний конкурувати з ААА-виробництвом. Це ще раз довело життєздатність авторських ігор навіть у складних жанрах.

Рис. 1.5 – Ігровий процес «Salt and Sanctuary»

Hollow Knight - сучасна метроідванія в незвичайному мальованому стилі, яку деякі вже встигли охрестити класикою.

Гра Hollow Knight, створена незалежною австралійською компанією Team Cherry, вийшла у 2017 році та одразу привернула увагу як критиків, так і гравців завдяки своїм якісним і неординарним якостям. Запозичивши основні механіки метроідваній, вона отримала унікальну візуальну складову та захопливий сюжет.

Hollow Knight виконана у стилі вітражів та акварелі. Цей незвичайний художній стиль дозволив створити неповторний світ гри, населений таємничими істотами. Гравці керують Лицарем, який прагне дізнатись таємниці покинутого Порожнечного міста. Досліджуючи його затоплені райони, вони відкривають унікальних персонажів та історію цивілізації.

Графічна складова Hollow Knight досягає чудового балансу між естетичністю та функціональністю. Вона не схожа ні на одну попередню гру, але при цьому задає правильний настрій пригод. Як і належить метроідванії, тут є великі затоплені локації з невідомими куточками та секретами. Поступово герой вивчає нові здібності, що дає можливість досліджувати нові райони.

Завоювання кожної території супроводжується постійним відчуттям небезпеки, оскільки на шляху чіпляння ієрархічних босів. Проте, незважаючи на складність, гра надає велику свободу та не ставить жорсткі рамки. Фантастичний саундтрек тільки підсилює атмосферу пригод.

За короткий час після виходу Hollow Knight отримала низку нагород та визнання як критиків, так і гравців. Її нерідко називають однією з кращих метроідваній усіх часів та класикою жанру. Це стало можливим завдяки якісному поєднанню геймплею з унікальним візуальним стилем. Гра довела життєздатність авторських проєктів навіть у насиченому жанрі та встановила нові стандарти якості для indie-ігор.

Таблиця 1.3

Порівняльна таблиця оглянутих ігор

Критерій	**Super Metroid**	**Super Mario Bros. Wonder**	**ENDER MAGNOLIA**	**Ori and The Blind Forest**	**Salt and Sanctuary**	**Hollow Knight**
Жанр	Метроідванія	Платформер/Метроідванія	JRPG/Метроідванія	Платформер	Souls-like/Метроідванія	Метроідванія
Візуальний стиль	16-бітна графіка	Покращена піксель-графіка	Яскрава анімація	Детальна ручна графіка	Піксель-арт в стилі аніме	Мальований стиль вітражів
Ймовірний движок	Внутрішній SNES	Внутрішній Nintendo	Unity або Unreal	Внутрішній рушій	Внутрішній 2D рушій	Unity
Мова програмування	Асемблер	Асеблер та C++	C# та C++	C++	C++	C#

Переваги	Встановила стандарти, атмосфера	Класичний геймплей, оптимізація	Якісна анімація, глибина	Захоплив ий світ, оригін альніст ь	Складний виклик, багатогран ність	Унікальна візуалізація, атмосфера
Недоліки	Застарілі технології	Відсутність інновацій	Висока складність	Можли ва надмір на складн ість	Висока складність	Висока складність
Атмосфера та оповідан-ня	Містична атмосфера	Гумористи чний	Фентезійн ий світ	Таємн ича атмосф ера	Похмурий світ	Глибока історія цивілізації
Інноваційніс ть	Встановила стандарти метроідвані ї	Оновлена класика	Новий погляд на JRPG	Іннова ційна візуалі зація	Новий підхід до souls-like	Унікальна візуальна подача
Ігровий процес	Дослідже ння, відкриття здібностей	Класичний платформе р	Численні квести, бої	Акроб атичні пересу вання	Виснажли ві бої з босами	Секретні райони, нові здібності
Секрети та дослідження	Багато секретів і таємниць	Нові секрети і квести	Нелінійні квести	Древні загадк и	Відкриття предметів	Дослідженн я покинутих місць
Звукове оформлення	Атмосферн а музика	Класичні і нові мелодії	Чудовий саундтрек	Захопл ивий саундт рек	Похмурий саундтрек	Фантастичн ий саундтрек

Свобода та нелінійність	Свобода вибору шляху	Лінійне проходження	Численні квести	Відкритий світ для дослідження	Свобода руху	Вільне дослідження світу
Рівень складності	Середній	Низький	Різний	Помірний	Високий	Високий
Вплив та визнання	Еталон жанру, шедевр	Оновлена класика	Високі оцінки	Новий рівень платформерів	Визнання критиків	Нова класика метроідванії

Продовження таблиці 1.3

1.4. Розробка технічного завдання

Метою даного проекту є створення двовимірної комп'ютерної гри жанру Метроідванія, використовуючи сучасні технології та інструментарій. Основні цілі проекту передбачають створення функціональної гри, яка буде цікавою та захопливою для гравців, реалізує ключові механіки жанру Метроідванія, матиме кілька динамічних та різноманітних рівнів, міститиме яскравих та небезпечних ворогів, а також буде зручною в управлінні та інтуїтивно зрозумілою.

Для досягнення цих цілей передбачено низку завдань, серед яких розробка концепції та сюжету гри, створення графічного та звукового оформлення, реалізація ігрового рушія та логіки, розробка механік та рівнів гри, здійснення тестування і усунення помилок, а також випуск демо-версії гри. Цей комплексний підхід дозволить забезпечити високу якість кінцевого продукту та максимальну задоволеність гравців.

Гра буде реалізована у двовимірній графіці в стилі 16-бітових ігор, що надасть їй ретро-відчуття та ностальгічний шарм.

Ігровий світ складатиметься з кількох різноманітних локацій, кожна з яких матиме свої унікальні характеристики та виклики. Основними ворогами будуть монстри, які виглядають як гриби. Гравець повинен буде уникати небезпек, боротися з ворогами та вирішувати головоломки, щоб просуватися далі у грі.

Основні елементи ігрового процесу включають вільне переміщення по відкритому ігровому світі, збір корисних предметів та знарядь, вдосконалення характеристик персонажа, взаємодію з об'єктами та неігровими персонажами, битви з різноманітними ворогами, розв'язання головоломок та завдань, дослідження секретних місць та таємниць, а також нелінійний сюжет з можливістю вибору шляху дій. Ці елементи створять багатий та захопливий ігровий досвід, який зацікавить гравців та мотивуватиме їх повертатися до гри знову і знову.

Головні механіки гри включають пересування (біг, стрибки, лазіння), атаку ворогів (удари, магія), збір корисних предметів, взаємодію з ігровим світом, розв'язання головоломок, вдосконалення персонажу (очки досвіду, здоров'я), екіпіровку (зброя, обладунки) та збереження прогресу. Ці механіки забезпечать глибокий та варіативний геймплей, що дозволить гравцям експериментувати з різними стратегіями та підходами до проходження гри.

Для реалізації проекту будуть використані такі технічні засоби як мова програмування Python, бібліотека PyGame для роботи з графікою і звуком, редактор карт Tiled, система класів і об'єктів, а також патерн MVC для поділу логіки. Основні класи, що будуть використовуватися в проекті, включають клас Game, який є центральним ядром рушія, клас Scene, який є базовим класом для рівнів, клас Player для керування гравцем, клас Enemy для поведінки ворогів, клас Item для предметів та здобутків, клас Camera для камери спостереження, а також тайли, текстури та спрайти. Ці технічні компоненти забезпечать високий рівень деталізації та реалістичності ігрового

світу, а також дозволять розробникам легко додавати нові елементи та функціональність у гру.

Процес розробки передбачає інтенсивне тестування на всіх етапах, що включає тестування окремих компонентів (класів), інтеграційне тестування, альфа та бета-тестування, пошук та усунення помилок, тестування ігрового балансу, усунення помилок продуктивності, а також забезпечення якості користувацького досвіду. Це дозволить виявити та виправити всі можливі недоліки та проблеми, що забезпечить високий рівень якості кінцевого продукту.

Результатом проекту має стати функціональна демонстраційна версія гри, яка реалізує основні механіки жанру Метроідванія. Подальшими кроками може стати розширення гри, включаючи додавання нових рівнів, здібностей персонажа, ворогів та сюжетних ліній. Розробка гри дозволить набувати досвід у створенні ігор та закріпити навички програмування, що буде корисним для подальших проектів та професійного розвитку розробників.

В цьому розділі було досліджено жанр відеоігор Метроідванія, його історію та еволюцію, а також найвідоміші зразки цього жанру. Метроідванія поєднує в собі елементи пригодницьких ігор, платформерів та головоломок, при цьому надаючи особливий акцент на не лінійність світу та його поступове розкриття завдяки новим здібностям гравця.

Цей жанр розпочав свій розвиток у 1980-х роках з виходом гри «Metroid» для Nintendo NES. У цій грі були уперше запропоновані нелінійний дизайн рівнів та принципи «розблокування», які згодом стали фундаментальними для жанру. Водночас зародження метроідванії пов'язано і з серією «Castlevania», зокрема із грою «Castlevania: Symphony of the Night», випущеною у 1997 році, яка досягла визначального успіху для жанру, додавши йому глибини за допомогою сюжету та атмосфери.

Сучасні представники жанру переважно зберігають його фундаментальні елементи, такі як нелінійність, дослідження світу та розблокування нових можливостей, продовжуючи розвивати та

вдосконалювати його. Так, такі тайти як «Ori and the Blind Forest», «Hollow Knight», «Salt and Sanctuary» досягли визнання критиків і гравців та встановили нові стандарти якості. Проте створення якісної метроідванії залишається складним завданням, що вимагає глибокого дизайну та балансу.

Водночас жанр продовжує активно розвиватися та еволюціонувати, переходячи від двовимірних до тривимірних світів, а також збагачуючи ігровий процес новими механіками. Це демонструє його життєздатність і здатність як приваблювати нових розробників та аудиторію гравців, пропонуючи унікальний досвід дослідження та постійного прогресу гравця всередині складного ігрового світу.

РОЗДІЛ 2. ПРОЕКТНА/ДОСЛІДНИЦЬКА ЧАСТИНА

2.1. Розробка концептуального документу

2.1.1. Загальна характеристика проекту

Дана робота присвячена розробці прототипу комп'ютерної гри за допомогою інструментарію Python і бібліотеки Pygame. Метою проекту було створення гри для опанування основ програмування ігор.

Ігровий процес полягає у керуванні персонажем на двовимірному екрані за допомогою клавіатури чи геймпада. Гравець зможе переміщатись та стрибати, а також взаємодіяти з простими об'єктами та ворогами.

Жанр гри – метроідванія, оскільки він дозволяє реалізувати основні ігрові механіки - переміщення, стрибки, взаємодію з об'єктами, що є достатнім для опанування основ розробки ігор на даній технологічній платформі. Подальша робота може передбачати доопрацювання та розширення функціоналу для створення цікавішої демонстраційної гри.

У результаті роботи ми отримали працюючу демонстраційну гру, яка відтворюватиме основи ігрового процесу у жанрі метроідванія, що дозволила опанувати базові навички програмування ігор та перевірити роботу рушія.

Гра розрахована на ПК. В інтерфейсі передбачається використання клавіатури або геймпада. Мета рейтингу - загальна аудиторія, оскільки це демонстраційна гра дитячого характеру.

Основні завдання проекту, які були виконані:

- Створити ігровий рушій на Python/Pygame
- Реалізувати персонажа та його керування
- Додати інтерактивні елементи
- Розробити ігровий процес
- Провести тестування та налагодження

Завершення проекту дозволило опанувати основи програмування ігор, ознайомитись з Python і Pygame, що стало базою для подальшого вдосконалення навичок через створення більш складних ігор.

Основними етапами роботи були:

1) Розробка концепції
2) Створення ігрового рушія
3) Реалізація ігрових об'єктів
4) Розробка ігрових механік
5) Тестування та налагодження

2.1.2. Опис світу та сюжету гри

Небо над лісом хмарувато і загрозливо. Молодий чаклун Мерлін спішила додому після тривалих студій у школі магії. Вона знала, що ліс небезпечним вночі, тому прагнула якнайшвидше добратися до свого замку.

Проте по дорозі у лісі Мерлін побачила дивну фігуру, що ледь вимальовувалася крізь кущі. З цікавості чаклунка підійшла ближе і побачила чудовисько, яке поскакувало по лісі та завдавало шкоди деревам. Чудовисько являло собою суміш грибів різних форм та розмірів, що незрозуміло як пересувалися поміж дерев та кущів.

Мерлін спробувала зачарувати чудовисько, проте її магія не мала ефекту. Тоді вона вирішила битися з ним магічним мечем. Важко було влучити у рухливий килим з грибів, але Мерлін вдало застосувала два швидкі стрибки і нанесла декілька ударів мечем. Чудовисько розпалося на окремі гриби, які поскакали в різні боки та швидко зникли серед дерев.

Мерлін задихалася від бою, але задоволено посміхалася - їй вдалося перемогти цю небезпечну істоту. Раптом її увагу привернув стогін у нетрях лісу. Магічним світлом вона прокралася крізь густий чагарник і побачила пораненого чоловіка.

Це був лицар на ім'я Галгамор, який зазнав нападу грибних чудовиськ у лісі. Його коня вже не було видно. Мерлін віднесла пораненого лицаря до

найближчої поляни і надала йому першу допомогу за допомогою магії цілющих зілля. Поступово Галгамор оговтався і розповів Мерлін, що ці грибні істоти з'явилися в останні дні та тероризують мешканців навколишніх земель.

Мерлін з Галгамором вирішили об'єднати сили, щоб визначити причину появи цих істот та знайти спосіб їх подолати. Разом вони повернулися в замок Мерлін. Там чаклунка провела дослідження за допомогою магічних артефактів і книг. Вона з'ясувала, що причиною появи грибних монстрів є згубний вплив темної магії, яка проникла в коріння Древа Життя, що росло посеред Чарівного лісу.

Ця магія поширювала отруйний вплив по всьому лісу, перетворюючи тварин і рослини на чудовиськ. Тепер стало зрозуміло - потрібно вирушати до Древа Життя і знайти джерело темної магії, щоб звільнити ліс від прокляття.

Мерлін і Галгамор зібрали припаси та озброєння і вирушили в небезпечну подорож. По дорозі їм довелося минати численні випробування. У глибоких нетрях лісу парою нападали грибні велети, на яких Мерлін вправно метала заклинання, а Галгамор вправно розтерзував їх мечем.

Вночі Мерлін не спалося, і вона заснула під деревом неподалік від табору. Раптом її обхопили гігантські грибні корені та затягнули у темну печеру.

Галгамор прокинувся й не побачив Мерлін поруч. Він швидко здогадався, що сталося, та спрямувався до печери, звідки чулися заклики на допомогу. Лицар забрав меча та смолоскип і сміливо кинувся рятувати чаклунку.

Входячи до темряви підземелля, Галгамор готувався до бою, його погляд був зосереджений, а рука міцно стискала меч. Глибоко під землею лежала небезпека, яка могла покласти край його подорожі. Але Галгамор знав, що повинен йти далі, щоб врятувати Мерлін.

Перед ним стояла нелегка задача: винищити всіх ворогів, що стоять на його шляху. Гриби, які стали смертельною загрозою для всіх живих істот,

охороняли кожен коридор і кожну кімнату підземелля. Вони володіли могутньою магією, яка могла вивести з ладу навіть найсильнішого воїна.

Головна мета Галгамора була ясна - знайти і знищити Головного Гриба, який керував усіма іншими. Тільки знищивши його, він зможе звільнити Мерлін і покласти край пануванню грибів.

Гравець, керуючи Галгамором, повинен буде досліджувати кожен кут підземелля, битись із небезпечними грибами, які спочатку будуть слабкі, а далі вже с кожнем рівнем становитись сильнішими, та розгадувати таємниці, приховані в цих темних коридорах. Кожен крок вперед вимагатиме стратегічного мислення та швидкої реакції.

2.1.3. Геймплей, механіки, жанр

Жанр: Метроідванія,

Вікові обмеження: відсутні.

Параметр	Мінімальні вимоги
ОС	Windows 10
Процесор	Intel Core i3 Dual Core
Оперативна пам’ять	4 GB ОП
Відеокарта	Інтегрована графіка Intel HD Graphics 4000 або краще
Місце на диску	500 MB

Основні ігрові процеси:

- Пересування гравця (біг, стрибки, подвійний стрибок, рівінь)
- Атакування ворогів
- Збирання предметів
- Переходи між рівнями

Механіки руху:

– Гравець може рухатися вліво/вправо, стрибати, виконувати подвійний стрибок та швидкий рівінь у бік

– Рух контролюється клавіатурою або підключеним джойстиком

– Гравець взаємодіє з ландшафтом та іншими об'єктами на рівні

– Механіки бою:

– Гравець може атакувати ворогів мечем у чотирьох напрямках (вгору, вниз, вліво, вправо)

– Атака здійснюється натисканням певної кнопки

– Вороги отримують шкоду від атак гравця та можуть бути вбиті

– Механіки взаємодії:

– Гравець може зіштовхуватися з перешкодами на рівні

– Існують контрольні точки, які відновлюють стан гравця після його смерті

– Гравець може переходити між рівнями через спеціальні виходи

В грі є кілька видів вороги – гриби, які рухаються по певній траєкторії та можуть завдавати шкоди гравцеві у разі зіткнення. Рівні складаються з різних плиток ландшафту, води та інших елементів.

2.2. Розробка дизайн-документа

2.2.1. Дизайн локацій та рівнів

Для нашої гри ми розробили рівень «Map Test, Map Test2 та Map Test3» на основі бібліотеки Pygame. Нижче наведено детальний опис рівней, його складових елементів, задач, а також ворогів, що зустрічаються на цьому рівні.

Але спочатку треба зазначити, що основні задачі всіх рівней включають:

1) Досягнення кінцевої точки - основною задачею гравців є дістатися до виходу на рівні, який веде до наступного рівня. Вихід розташований на координатах (3136, 320).

2) Уникнення ворогів та пасток - гравцям потрібно уникати ворогів та пасток, що розташовані на рівні. Вороги, такі як гриби, можуть завдавати шкоди гравцям при контакті, тому їх необхідно уникати або знищувати

Та основні характеристики всіх рівней мають:

– Розмір - 50 плиток у ширину і 17 плиток у висоту.

– Плитки - Використовуються плитки з трьох наборів: terrain (основний ландшафт), water (водні об'єкти) і enemies (вороги).

Складові рівня Map Test. Навколишнє середовище.

Основний ландшафт рівня складається з різноманітних перешкод, таких як платформи, водні об'єкти та архітектурні елементи. Плитки з кодом «14» використовуються для створення більшості поверхонь, тоді як плитки з кодом «26» і «27» використовуються для більш специфічних елементів ландшафту.

Водні об'єкти позначені плиткою з кодом «73». Вони розташовані у певних ділянках рівня, створюючи психологічні перешкоди для гравців.

Ігрові об'єкти:

Точки взаємодії, які включають чекпоінти та виходи. Чекпоінт розташований на координатах (128, 576) і позначений об'єктом «Checkpoint». Вихід з рівня знаходиться на координатах (3136, 320) і позначений об'єктом «map_test_2».

Вороги:

Гриби (mushroom) - розташовані по всьому рівню і позначені плиткою з кодом «77». Вони розташовані на різних координатах, таких як (1536, 704), (1212, 898), (2880, 960) та інших.

Друга локація «map_test_2». Гравець починає свій шлях на координатах (186, -188). Чекпоінт, який зберігає прогрес гравця, розташований на координатах (702.667, 837.333).

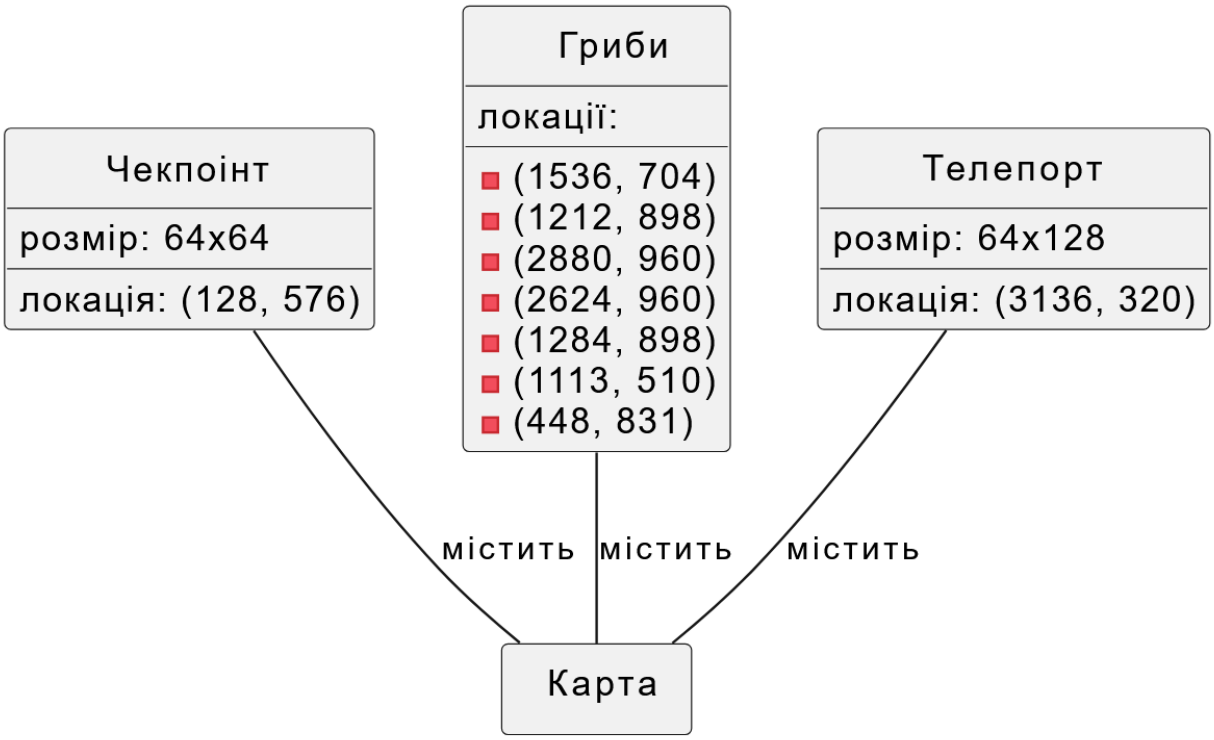

Рис. 2.1 – Схема елементів ігрової карти №1

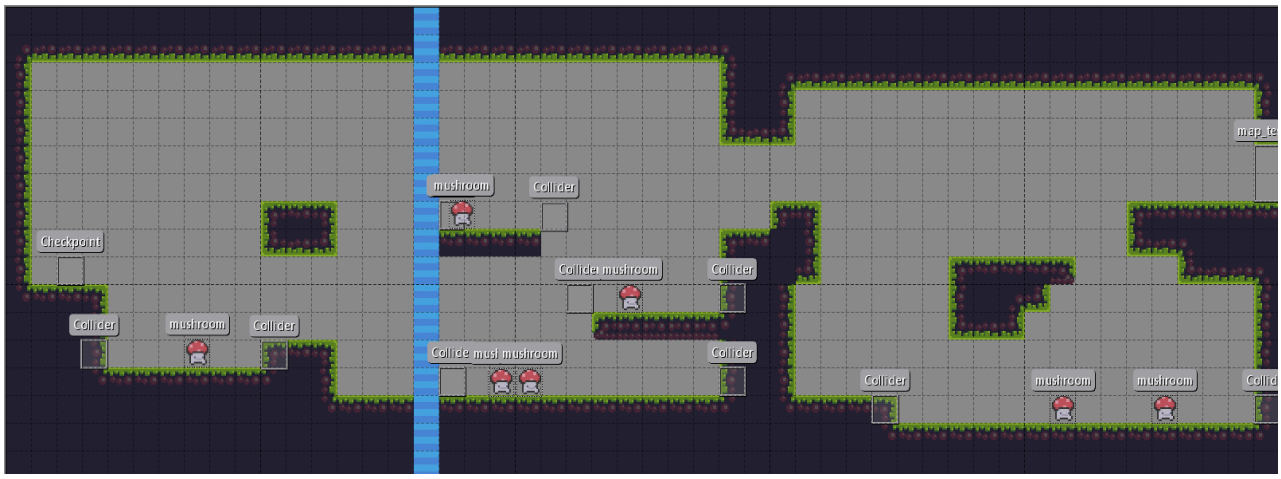

Рис. 2.2 - Перша локація

Навколишнє середовище рівня у грі визначається за допомогою шару «Terrain». Цей шар складається з різноманітних плиток, які створюють основний ландшафт та перешкоди. Плитки з кодом «14» є базовими для створення рівня, вони формують загальну структуру території. Також використовуються плитки з кодами «26» і «27», які створюють більш специфічні елементи ландшафту, такі як стіни та платформи.

Початок рівня розташований у лівому верхньому куті, де гравець починає свій шлях. Ландшафт рівня складається з різноманітних перешкод, які вимагають від гравця проявляти спритність та швидкість реакції. Наприклад, на рівні є платформи, на які потрібно стрибати, та стіни, які необхідно обійти або подолати.

Деталі навколишнього середовища:

– Початок рівня знаходиться в лівому верхньому куті.

– Ландшафт включає платформи та стіни, які створюють складні маршрути для проходження.

– Плитки «26» і «27» використовуються для створення особливих елементів ландшафту, які ускладнюють шлях гравця.

Ігрові об'єкти на цьому рівні включають чекпоінти, які зберігають прогрес гравця, та виходи, що дозволяють перейти на наступний рівень. Чекпоінт розташований на координатах (702.667, 837.333) і забезпечує збереження прогресу, щоб гравець міг продовжити гру з цього місця у випадку смерті.

На рівні є два виходи:

– Вихід «map_test» розташований на координатах (576, 768).

– Вихід «map_test_3» розташований на координатах (2298.67, 513.333).

Ці точки виходу дозволяють гравцеві завершити рівень та перейти на наступний, забезпечуючи плавний перехід між рівнями та підтримуючи інтерес гравця до гри.

Деталі ігрових об'єктів:

– Чекпоінти, які дозволяють зберігати прогрес гравця на рівні, розташовані на координатах (702.667, 837.333).

– Точки виходу, які дозволяють перейти на наступний рівень, розташовані на координатах (576, 768) та (2298.67, 513.333).

Вороги:

Вороги на цьому рівні представлені об'єктами типу «mushroom» (гриб). Вони розташовані на різних координатах по всьому рівню, створюючи загрози для гравця та роблячи проходження рівня більш складним. Коллайдери створюють невидимі стіни та перешкоди, які обмежують рух гравця та ворогів, забезпечуючи правильний баланс складності та контроль над геймплеєм.

Вороги розташовані на таких координатах:

- (2304, 896)
- (2500, 900)
- (1773.33, 894)
- (1088, 960)
- (1921.33, 640)
- (2109.33, 642.667)

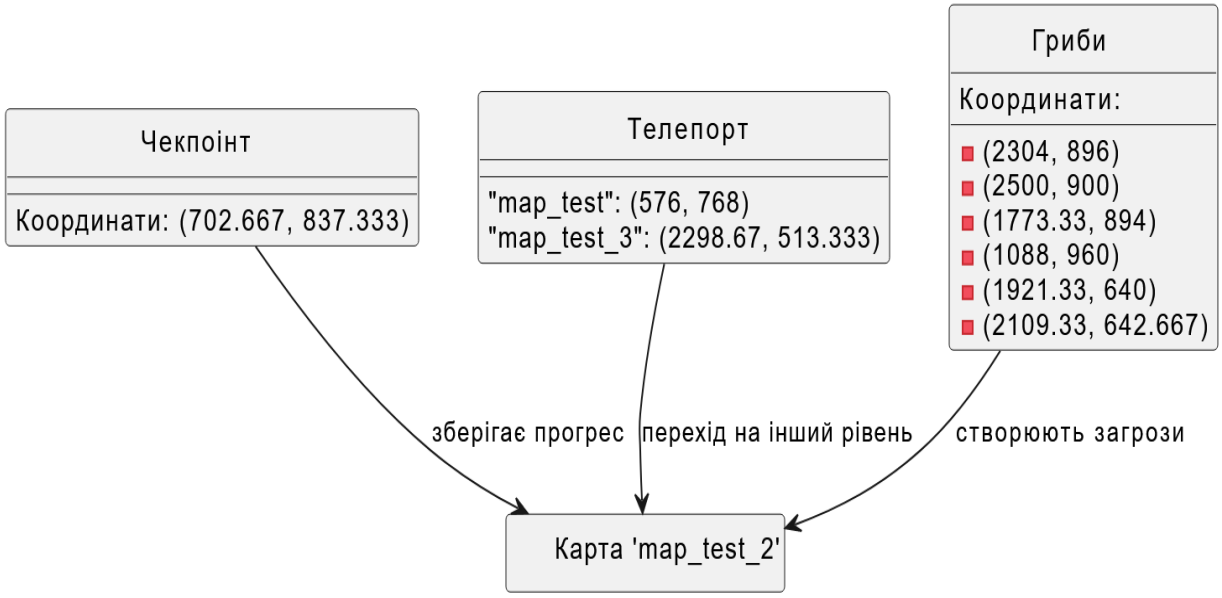

Рис. 2.3 – Схема елементів ігрової карти №2

Рис. 2.4 – Друга локація

Третя локація «map_test_3». Цей рівень схожий на минулі майже по всім парметрам, але тут гравець починає свій шлях на координатах (2304, 512). Початок рівня розташований з правої сторони, і цього разу гравець буде рухатися в ліву сторону до виходу. На шляху гравця буде розміщено 5 ворогів, кожен з яких знаходиться на певних координатах. Гравцю необхідно уникати або знищувати ворогів, щоб дістатися до виходу.

Розглянемо розташування ворогів:

1) Ворог 1 - розташований на координатах (2304, 896)
2) Ворог 2 - розташований на координатах (2500, 900)
3) Ворог 3 - розташований на координатах (1773.33, 894)
4) Ворог 4 - розташований на координатах (1088, 960)
5) Ворог 5 - розташований на координатах (2109.33, 642.667)

На рівні розміщений вихід, який знаходиться на координатах (194, 130). Гравцю потрібно буде пройти через усі перешкоди і ворогів, щоб дістатися до цього виходу і завершити гру.

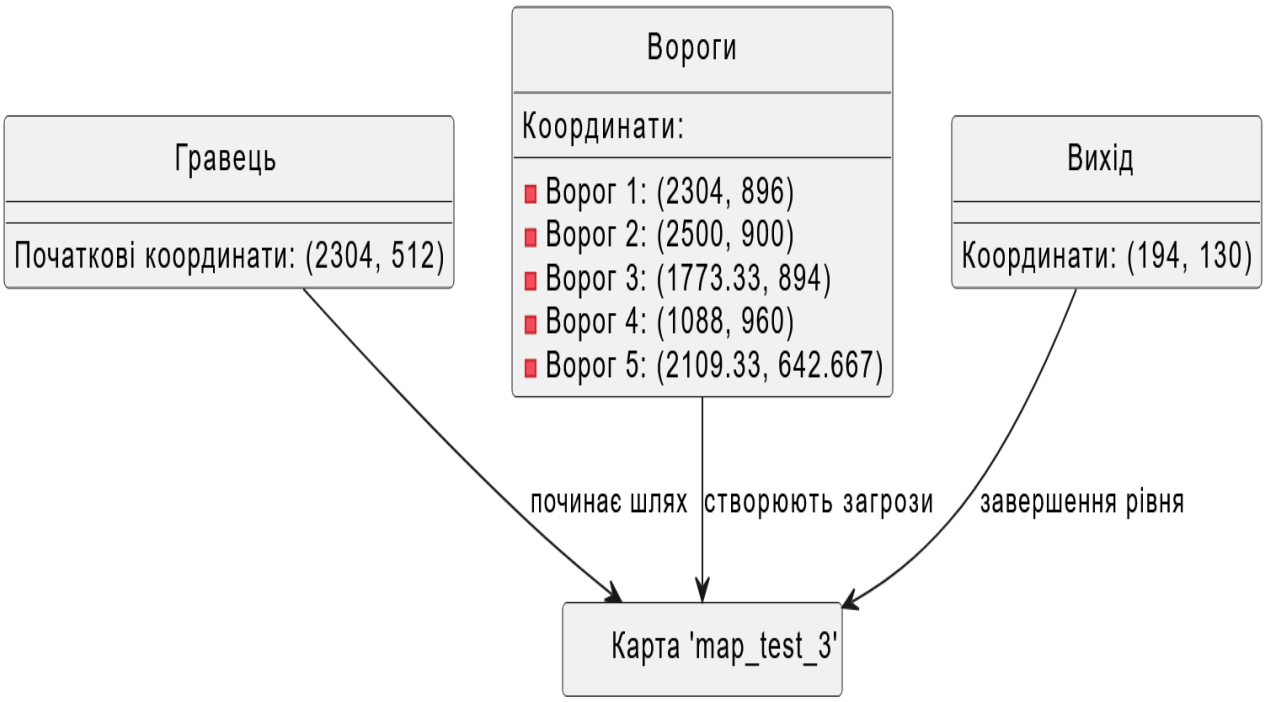

Рис. 2.5 – Схема елементів ігрової карти №3

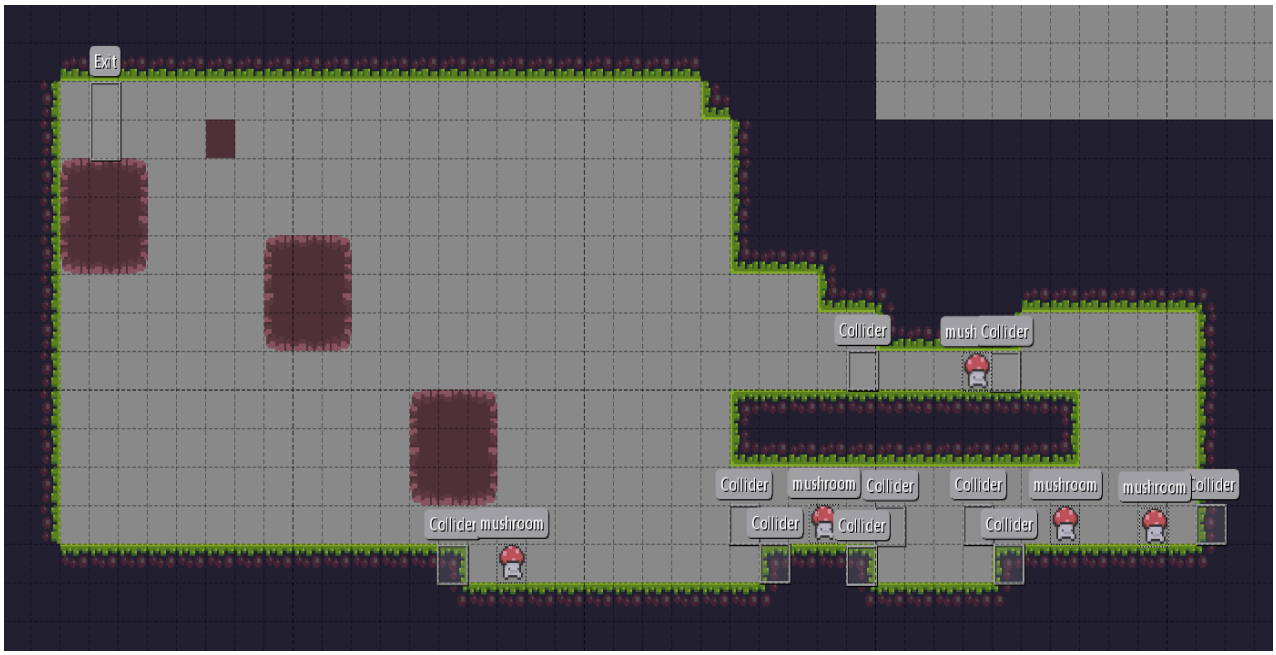

Рис. 2.6 – Третя локація

РОЗДІЛ 3. ПРАКТИЧНА РЕАЛІЗАЦІЯ ТА РЕЗУЛЬТАТИ ДОСЛІДЖЕННЯ

3.1 Вибір і обґрунтування програмних засобів реалізації проекту (графічної та програмної частин)

3.1.1 Вибір та обґрунтування проектних рішень

У сучасних умовах розробка комп'ютерних ігор вимагає ретельного аналізу та обґрунтування проектних рішень, що включає вибір оптимальної технологічної платформи, інструментів розробки, а також технічних та економічних аспектів. Основними критеріями для вибору технологій є функціональність, продуктивність, зручність використання, підтримка з боку спільноти розробників та вартість.

У нашому проекті вибір припав на мову програмування Python та бібліотеку PyGame. Python обрано через його простоту у використанні, високу читабельність коду та активну підтримку з боку спільноти. Бібліотека PyGame надає потужні інструменти для роботи з графікою та звуком, що є критично важливими для створення якісного ігрового продукту. Окрім того, використання редактора карт Tiled дозволяє значно спростити процес створення ігрових рівнів та забезпечити високу деталізацію ігрового світу.

Таблиця 3.1

Основні технології, використані у проекті

Технологія	Опис	Причини вибору
Python	Мова програмування	Простота, читабельність, активна спільнота
PyGame	Бібліотека для роботи з графікою та звуком	Потужні інструменти, зручність
Tiled	Редактор карт	Спрощення процесу створення рівнів, висока деталізація

Вибір проектних рішень також включає визначення ключових механік гри та їх реалізацію. Головні механіки нашої гри включають пересування (біг, стрибки, лазіння), атаку ворогів (удари, магія), збір корисних предметів, взаємодію з ігровим світом, розв'язання головоломок, вдосконалення персонажу (очки досвіду, здоров'я), екіпіровку (зброя, обладунки) та збереження прогресу. Ці механіки забезпечують глибокий та варіативний геймплей, що дозволяє гравцям експериментувати з різними стратегіями та підходами до проходження гри.

Таблиця 3.1.

Основні механіки гри

Механіка	Опис	Значення для геймплею
Пересування	Біг, стрибки, лазіння	Вільне переміщення по ігровому світу
Атака ворогів	Удари	Боротьба з ворогами
Взаємодія з ігровим світом	Об'єкти, персонажі	Взаємодія з середовищем та персонажами

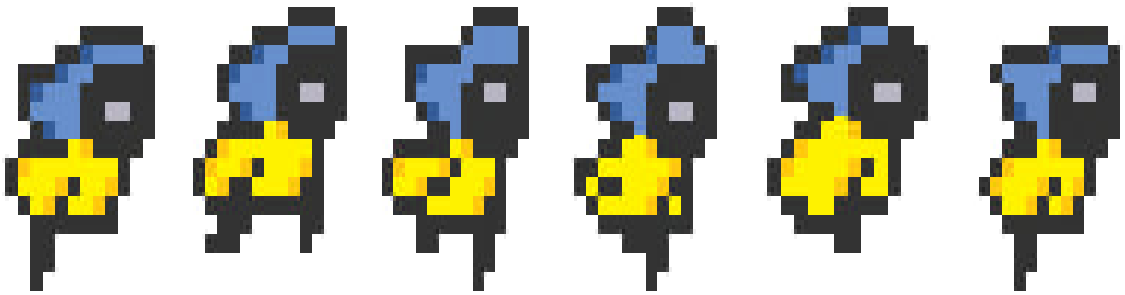

Рис 3.1 – Механіка пересування головного героя

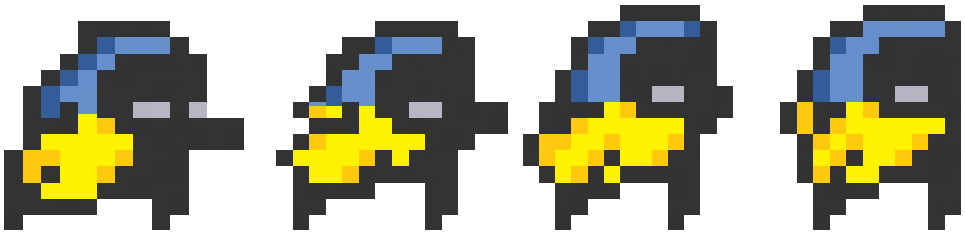

Рис 3.2 – Механіка удару

Реалізацію механік дивиться в додатку до диплому.

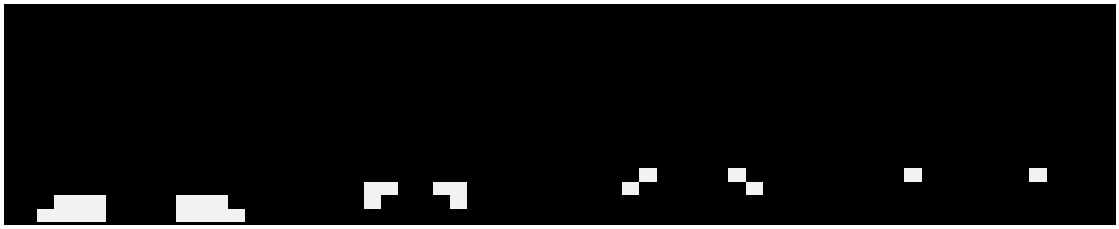

Рис. 3.3 – Взаємодія з середовищем до стрибку

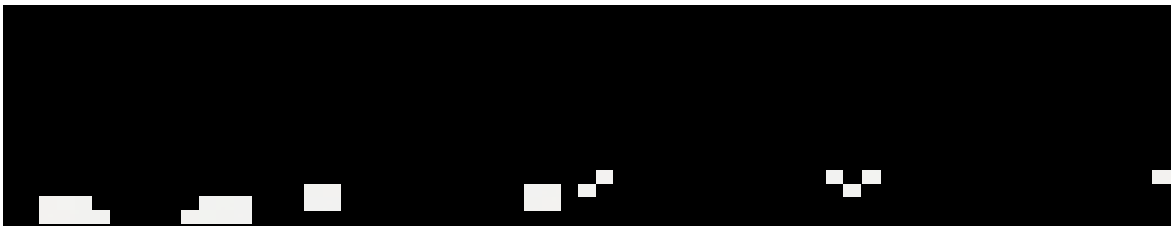

Рис. 3.4 – Взаємодія з середовищем після стрибку

3.1.2 Програмна частина

Програмна частина гри була реалізована за допомогою мови програмування Python та бібліотеки Pygame. Python був обраний як основна мова програмування через ряд переваг, які він надає:

1) Це простота та зручність самої мови. Python спрощує написання коду завдяки високому рівню абстракції та мінімальній кількості синтаксичних конструкцій, що ускладнюють програмування. Це дуже важливо для швидкого прототипування та оперативного внесення змін.

2) Python є інтерпретованою мовою, тобто код можна тестувати та відлагоджувати безпосередньо під час його написання, на відміну від компілюваних мов. Це дозволяє скоротити цикл розробки та оперативно реагувати на помилки. Для проекту з агільною методологією розробки, як 2D платформер, це є важливою перевагою.

3) Python користується великою популярністю серед розробників, тому існує величезна кількість різноманітних бібліотек та фреймворків, які можна використати у проекті. Це суттєво спрощує реалізацію складних задач та дозволяє уникнути «винаходу велосипеда».

Однією з таких популярних бібліотек для розробки ігор на Python є Pygame. Вона забезпечує комплексний функціонал для роботи з графікою, звуком, фізикою та інтерфейсом, який необхідний для багатьох ігрових жанрів. Pygame спрощує реалізацію таких концепцій як рендеринг спрайтів, завантаження тайлсетів, анімація, обробка подій клавіатури/миші та інтерфейсу користувача.

Наприклад, для роботи з графічними ресурсами Pygame надає об'єкти Surface та Sprite. Surface представляє 2D поверхню, на якій відбувається рендеринг, а Sprite є класом, що поєднує об'єкт Surface і прямокутник для відображення на екрані. Це дозволяє легко реалізувати рухомі об'єкти з різними спрайтами. Для завантаження тайлсетів Pygame має вбудовані функції для роботи з файловими форматами PNG, JPG та ін.

Іншим корисним функціоналом є робота зі звуком (відтворення музики, ефектів), обробка подій клавіатури/миші, фізичний рушій, робота з шriftами, колізіями та ін. Це дозволяє зосередитися лише на ігровій логіці, а не «винаходити велосипед» для технічної частини.

Для підвищення продуктивності розробки та якості коду був обраний об'єктно-орієнтований підхід. Ігровий процес був розділений на окремі класи, які відповідають за певні функціональні частини: Player, TileMap, Enemies, UI та ін. Кожен клас містить лише ту логіку, яка йому притаманна, що зробило код чіткішим, зрозумілішим та легким для подальшої модифікації та розширення.

Крім того, об'єктно-орієнтований дизайн дозволяє легко наслідувати базові класи та створювати похідні для конкретних підкласів супротивників, предметів тощо. Таким чином, їх поведінка та логіка може відрізнятися, але залишатися сумісною з базовою, що також спрощує масштабування проекту шляхом додавання нового функціоналу та елементів.

В результаті застосування Python та Pygame вдалося ефективно реалізувати всі необхідні ігрові функції 2D гри: рух гравця, колізії, анімацію,

роботу з тайлмапами, штучний інтелект супротивників тощо. Код виявився чітким, зрозумілим та легким у подальшій оптимізації.

3.1.3 Графічна частина

Для реалізації графічної частини також підійша бібліотека Pygame. Pygame є популярною бібліотекою для розробки двовимірних ігор на мові програмування Python. Її популярність серед розробників пояснюється простотою використання, широкими можливостями та активною підтримкою спільноти. Крім того, Pygame забезпечує високий рівень абстракції, що дозволяє зосередитися на розробці гри, а не на низькорівневих деталях графічного рендерингу.

Наприклад, модулі Pygame для роботи з графікою та зображеннями дозволяють завантажувати, відображати та маніпулювати зображеннями у форматах PNG, JPEG та інших.

Також для частини нашої гри ми вирішили використати простий графічний редактор Paint. Оскільки планувалось, що наша гра буде мати піксельну, графіку, то Paint ідеально підходійшов для її створення.

Піксельна графіка - це графіка, побудована з окремих пікселів, тобто точок з кольором. На відміну від векторної графіки, де об'єкти мають чіткі криві та форми, піксельна графіка складається саме з пікселів, що дає нам можливість створювати прості, але цікаві на вигляд об'єкти та персонажів.

Піксельна графіка дуже популярна в ретро-стильових іграх, оскільки нагадує старі ігри 8 та 16-бітних ер. Ми хотіли надати нашій грі саме такого ретро-стилю, тому цей стиль графіки ідеально підходить.

Також з переваг піксельної графіки – це її простота створення. Намальовувати прості піксельні зображення можна дуже швидко на базовому редакторі, як Paint, що дозволяє нам економити час та ресурси, не витрачаючись на складні 3D-моделі чи векторну графіку.

Ще однією основною складовою графічної частини в нашій грі є використання тайл-сетів та редакторів карт, таких як Tiled. Tiled є потужним інструментом для створення двовимірних рівнів гри, що дозволяє зручно розміщувати тайли та інші об'єкти на карті. Використання Tiled значно спрощує процес створення рівнів та їх подальшу інтеграцію в гру. Карти, створені в Tiled, зберігаються у форматі TMX, який може бути легко завантажений та оброблений у Pygame за допомогою спеціальних модулів, таких як pytmx.

Він дозволяє розробникам створювати та редагувати двовимірні ігрові рівні за допомогою тайлів, які є графічними елементами, що повторюються. Цей підхід має багато переваг, зокрема можливість швидкого створення великих ігрових просторів за допомогою відносно невеликої кількості графічних ресурсів. Крім того, використання тайлів дозволяє легко змінювати дизайн рівня, замінюючи лише кілька елементів, без необхідності переробляти всю карту.

Однією з головних переваг Tiled є його зручний та інтуїтивно зрозумілий інтерфейс, який дозволяє швидко та ефективно розміщувати тайли на карті. Розробники можуть легко маніпулювати тайлами, змінюючи їх положення, обертання та інші властивості, що дозволяє створювати складні та цікаві рівні. Крім того, Tiled підтримує роботу з кількома шарами, що дозволяє розробникам розділяти різні елементи карти на окремі рівні, такі як фон, об'єкти переднього плану та колізії. Це забезпечує більшу гнучкість та контроль над дизайном рівнів.

Tiled також підтримує використання об'єктів, які можуть бути розміщені на карті разом з тайлами. Об'єкти можуть бути використані для створення різних елементів гри, таких як персонажі, вороги, предмети та інші інтерактивні об'єкти. Кожен об'єкт може мати свої власні властивості, такі як позиція, розмір та тип, що дозволяє легко інтегрувати їх у гру та забезпечити взаємодію з іншими елементами.

Карти, створені в Tiled, зберігаються у форматі TMX (Tiled Map XML), який є стандартом для зберігання даних двовимірних ігрових карт. Формат TMX має кілька важливих переваг, зокрема можливість зберігати інформацію про тайли, об'єкти, шари та інші елементи карти в зручному для обробки вигляді. Завдяки XML-структурі, файли TMX можуть бути легко завантажені та оброблені у різних мовах програмування та бібліотеках, що забезпечує високу гнучкість та сумісність.

3.2. Реалізація графічної частини

3.2.1 Створення сцени

Сцена є основою будь-якої гри, адже саме на ній розгортаються всі події. У нашій грі сцена створюється за допомогою програми Tiled, яка дозволяє зручно розміщувати тайли для формування ігрового світу.

Тайли (tiles) є невеликими графічними елементами, які використовуються для створення ігрових рівнів. Вони представляють різні частини навколишнього середовища, такі як земля, вода, червоні камні тощо.

Процес створення сцени.

Перший крок у створенні сцени – вибір відповідних тайлів. Для нашої гри ми використовували піксельні тайли, створені в Paint, що дозволило їх легко змінювати та адаптувати під потреби гри (наприклад, флаг України не зразу був додан).

Далі, використовуючи Tiled, ми створили карту рівня, розміщуючи тайли на відповідних місцях. Як виявилось, Tiled дозволяє зручно маніпулювати тайлами, змінювати їхні позиції та додавати нові елементи.

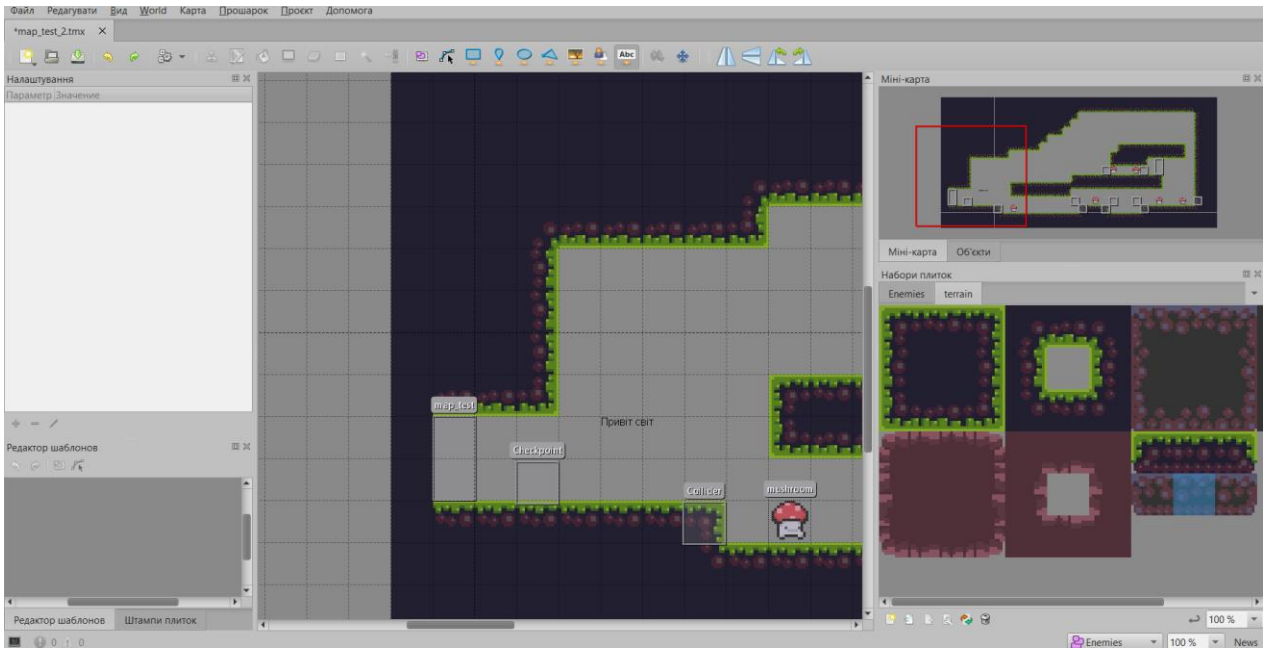

Рис. 3.5 - Створення сцени в Tiled

Рис. 3.6 – Перша локація

Рис. 3.7 – Друга локація

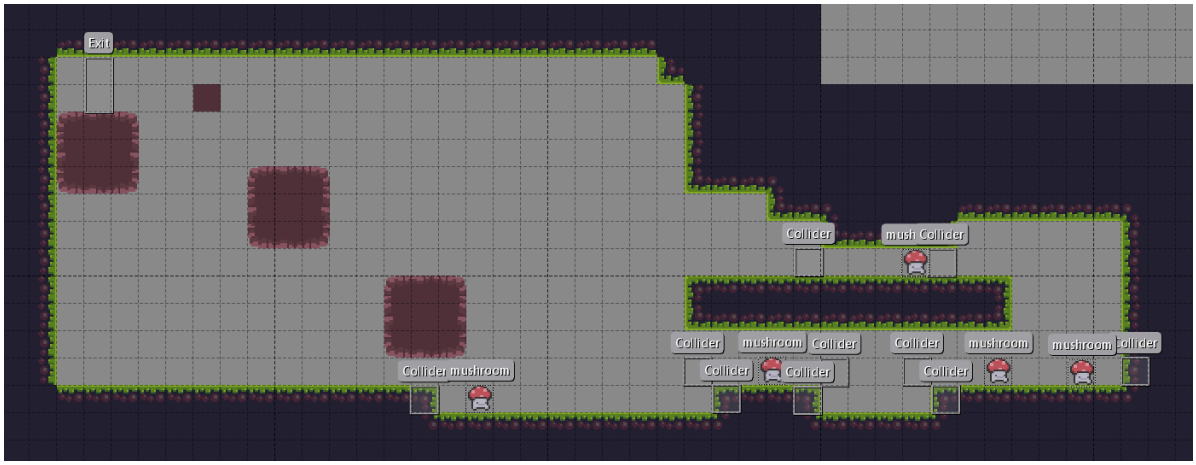

Рис. 3.8 – Третя локація

3.2.2 Створення персонажів

Персонажі є основними елементами будь-якої гри. У нашій грі персонажі також виконані в піксельному стилі, що відповідає загальному стилю гри.

Процес створення персонажів відбувався у кілька етапів:

Спочатку розроблялась концепція персонажів – їхній зовнішній вигляд, характери, ролі в грі, що допомагало створити унікальних і запам'ятовуваних героїв.

Наступним кроком було створення спрайтів. Спрайти – це окремі зображення персонажів, які використовуються для їхньої анімації. Для цього використовувався Paint, що дозволяло детально пропрацювати кожен піксель.

Окремим важливим етапом була розробка анімації персонажів. Анімація надає персонажам життя та робить гру більш динамічною. В нашій грі вона реалізовується шляхом зміни спрайтів у певній послідовності.

Для графічної частини використовувся редактор Paint.

Роглянемо на прикладі спрайти, для анімації атаки в праву сторону:

Рис. 3.9 - Спрайти персонажів, створені в Paint

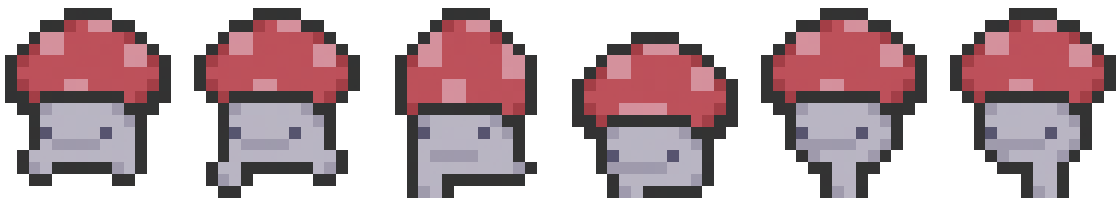

Рис. 3.10 – Спрайт ворога у русі

3.3 Реалізація основного функціоналу коду та механіки

Наш код базується на використанні об'єктно-орієнтованого підходу, що дозволяє ефективно організувати та керувати різними компонентами гри. Основою цього підходу є класи, які представляють різні об'єкти в грі, такі як персонажі, предмети, сцени тощо. Кожен клас має свої властивості та методи, що дозволяють йому виконувати певні дії.

Після запуску головного файлу гри основні класи починають взаємодіяти один з одним, забезпечуючи функціональність та динаміку гри. Ось список основних класів, які використовуються в коді:

1) Game - головний клас, що керує грою в цілому.
2) Scene - клас, який представляє окрему сцену або рівень гри.
3) Character - базовий клас для всіх ігрових персонажів.
4) Player - клас, який розширює Character і представляє гравця.
5) Enemy - клас, який розширює Character і представляє ворогів у грі.
6) Weapon - клас, який представляє зброю гравця.
7) UI - клас, який відповідає за інтерфейс користувача.
8) Transition - клас, що відповідає за анімацію переходів між рівнями.
9) Timer - клас для управління часом у грі.
10) Tile - клас, який представляє базовий тайл.
11) AnimatedTile - клас, який розширює Tile і представляє анімовані тайли.
12) ExitTile - клас, який представляє вихід на інший рівень.
13) Checkpoint - клас, який представляє контрольну точку на рівні.
14) CollisionManager - клас, який відповідає за обробку зіткнень між об'єктами.
15) CameraGroup - клас, що відповідає за управління камерою та відображення спрайтів на екрані.
16) ResourceManager - клас для управління ресурсами гри.
17) ParticuleManager - клас для управління анімаціями часток.

18) ParticuleEffect - клас, який представляє анімацію частки.

19) Level - клас, який представляє рівень гри і включає логіку та об'єкти рівня.

20) InputManager - клас для обробки вводу користувача.

21) GameStateManager - клас для управління станами гри.

22) Vector2d - клас, який представляє двовимірний вектор і містить методи для математичних операцій.

23) Support - клас, що містить допоміжні функції для різних задач у грі.

Розглянемо тепер його послідовну роботу:

Почнемо з розгляду основного класу, який відповідає за керування грою в цілому. Цей клас називається Game і є центральним елементом нашого коду. Він виконує кілька важливих функцій, які забезпечують правильну роботу гри та її взаємодію з користувачем.

Спочатку клас Game відповідає за ініціалізацію гри. Це означає, що при створенні об'єкта цього класу ми виконуємо всі необхідні налаштування та підготовку до старту гри. У конструкторі класу ми ініціалізуємо всі глобальні змінні, які використовуються в грі. Серед них поточна сцена, стан гри та інші параметри, які контролюють загальний хід гри. Наприклад, поточна сцена може бути представлена як об'єкт класу Scene, який містить усі об'єкти та логіку, пов'язані з поточним рівнем гри. Стан гри може включати інформацію про те, чи гра активна, чи знаходиться в паузі, чи завершилася.

Далі клас Game починає управляти основним циклом гри. Основний цикл гри — це безперервна послідовність дій, яка повторюється на кожному кроці виконання програми. Він включає в себе оновлення стану гри, обробку подій користувача та рендеринг сцени. Для цього в класі Game ми визначаємо методи start(), update() та render(), які керують основними етапами життєвого циклу гри.

Метод start() відповідає за початкову ініціалізацію гри. Він викликається один раз при запуску гри і виконує всі необхідні налаштування, такі як

завантаження ресурсів, ініціалізація об'єктів і сцен, встановлення початкових параметрів гри. У цьому методі ми завантажуємо всі необхідні графічні ресурси, такі як спрайти персонажів, тайли сцени та інші об'єкти. Завантаження ресурсів здійснюється за допомогою спеціального класу ResourceManager, який відповідає за управління всіма ресурсами гри. Також у методі start() ми створюємо початкову сцену гри і встановлюємо її як поточну сцену.

Метод update() викликається на кожному кроці основного циклу гри і відповідає за оновлення стану гри. У цьому методі обробляються всі дії гравця, оновлюється стан об'єктів та перевіряються умови гри. Наприклад, у методі update() ми перевіряємо натискання клавіш для управління персонажем, оновлюємо позиції об'єктів, перевіряємо зіткнення та інші важливі події. Цей метод є центральним місцем, де відбувається основна логіка гри. Крім того, у методі update() ми можемо перевіряти різні умови завершення гри або переходу на наступний рівень.

Метод render() відповідає за рендеринг сцени на екрані. У цьому методі викликаються методи рендерингу для всіх об'єктів сцени, таких як персонажі, тайли, вороги та інші елементи. Для відображення інтерфейсу користувача використовується клас UI, який забезпечує зручне та інтуїтивно зрозуміле відображення всіх необхідних даних, таких як здоров'я гравця, кількість зібраних предметів та інші важливі показники. Клас CameraGroup відповідає за управління камерою та відображення спрайтів на екрані. Він дозволяє слідкувати за рухом гравця та застосовувати ефекти, такі як тряска екрану під час отримання ушкоджень або наближення прицілу під час атаки.

Клас Player представляє гравця і включає методи для управління рухом, атакою та іншими діями. У цьому класі визначені методи для обробки вводу користувача, руху персонажа, перевірки зіткнень та оновлення стану. Player також включає логіку для взаємодії з іншими об'єктами сцени, такими як вороги, предмети та контрольні точки. Це дозволяє гравцеві виконувати різноманітні дії, такі як стрибки, атаки, збори предметів та проходження

рівнів. Крім того, клас Player має методи для обробки ушкоджень, що дозволяє зменшувати здоров'я персонажа при зіткненні з ворогами або небезпечними об'єктами сцени.

Клас Enemy представляє ворогів у грі. Він включає логіку для руху, атаки та взаємодії з гравцем. У методах цього класу визначені дії ворогів, такі як патрулювання, атака гравця та реагування на отримання ушкоджень. Enemy також має методи для перевірки зіткнень з іншими об'єктами сцени, що дозволяє ворогам уникати перешкод та реагувати на дії гравця. Це створює динамічний ігровий процес, де гравець повинен стратегічно підходити до боротьби з ворогами, використовуючи свої навички та можливості.

Клас Weapon відповідає за управління зброєю гравця. Він включає методи для анімації зброї, перевірки зіткнень зі ворогами та оновлення стану зброї. Weapon дозволяє гравцеві атакувати ворогів, використовуючи різні види зброї, кожен з яких має свої унікальні властивості та можливості. Це додає грі різноманітності та дозволяє гравцеві вибирати найбільш підходящу зброю для кожної ситуації. Крім того, клас Weapon включає логіку для управління запасами зброї та боєприпасів, що дозволяє гравцеві збирати та використовувати різні види боєприпасів під час гри.

Клас Transition забезпечує анімацію переходів між рівнями. Він виконує завдання плавного переходу від однієї сцени до іншої, змінюючи прозорість екрану та застосовуючи інші ефекти. Це дозволяє створити більш органічний ігровий процес, де переходи між рівнями відбуваються без різких змін і переривань. Transition також може використовуватися для створення інших ефектів, таких як зміна часу доби або погодних умов, що додає грі глибини та реалістичності.

Клас Timer використовується для управління часом у грі. За допомогою цього класу можна створювати таймери для виконання певних дій через задані проміжки часу, таких як тимчасова невразливість гравця або анімація часток. Timer дозволяє точно контролювати час виконання різних подій у грі, що додає їй динамічності та інтерактивності. Крім того, Timer може

використовуватися для обмеження часу на виконання завдань, що додає грі елемент змагання та виклику.

Клас CameraGroup відповідає за управління камерою та відображення спрайтів на екрані. Він дозволяє слідкувати за рухом гравця та застосовувати ефекти, такі як тряска екрану під час отримання ушкоджень або наближення прицілу під час атаки. CameraGroup також дозволяє змінювати масштаб сцени, що дозволяє гравцеві бачити більше або менше деталей залежно від ситуації. Це додає грі глибини та дозволяє гравцеві краще орієнтуватися у просторі.

Клас ParticuleManager відповідає за управління анімаціями часток. Він створює ефекти часток, такі як спалахи під час стрибка або анімації смерті ворогів. ParticuleManager дозволяє додати грі візуальні ефекти, що роблять її більш привабливою та динамічною. Крім того, анімації часток можуть використовуватися для відображення різних подій, таких як вибухи, вогонь або магічні ефекти, що додає грі різноманітності та реалістичності.

Клас InputManager обробляє ввід користувача, зчитує натискання клавіш та миші і передає цю інформацію до інших класів гри, таких як Player. Це дозволяє гравцеві керувати своїм персонажем, рухати його по сцені, атакувати ворогів та взаємодіяти з об'єктами. InputManager також може обробляти ввід з інших пристроїв, таких як геймпади, що робить гру більш доступною для різних гравців.

Клас GameStateManager відповідає за управління станами гри. Він дозволяє зберігати та завантажувати стан гри, змінювати поточний стан і забезпечувати плавний перехід між різними станами. GameStateManager дозволяє гравцеві зберігати свій прогрес та продовжувати гру з того місця, де він зупинився. Крім того, цей клас може використовуватися для реалізації різних ігрових режимів, таких як кампанія, мультиплеєр або режим виживання.

Після запуску головного файлу гри всі ці класи починають під'єднуватися та взаємодіяти один з одним, забезпечуючи функціональність та динаміку гри. Кожен клас виконує свої специфічні завдання, що дозволяє

створити комплексний ігровий досвід для користувачів. Наприклад, коли гравець рухається по сцені, клас Player оновлює його положення та перевіряє зіткнення з іншими об'єктами. У цей час CameraGroup слідкує за рухом гравця та відповідно змінює положення камери. Якщо гравець атакує ворога, Weapon перевіряє зіткнення з ворогом та наносить йому ушкодження. У той же час ParticuleManager створює ефект анімації удару, додаючи грі візуальної привабливості.

Якщо гравець отримує ушкодження, Player зменшує його здоров'я та запускає відповідні анімації за допомогою ParticuleManager. CameraGroup застосовує ефект тряски екрану, що робить подію більш реалістичною. Якщо здоров'я гравця зменшується до нуля, GameStateManager змінює стан гри на «гра закінчена», і UI відображає повідомлення про завершення гри.

Крім того, всі ці класи можуть взаємодіяти з Timer, щоб точно контролювати час виконання різних подій. Наприклад, після атаки ворога гравець може тимчасово стати невразливим за допомогою Timer, що дозволяє йому уникнути подальших ушкоджень. Timer також може використовуватися для створення різних анімацій, таких як вибухи або спалахи світла, що додає грі динамічності та візуальної привабливості.

Усі ці класи разом створюють комплексний ігровий досвід, де кожен елемент гри ретельно продуманий і взаємодіє з іншими елементами. Це дозволяє створити захоплюючий і динамічний геймплей, де кожна дія гравця має свої наслідки, і кожен елемент гри впливає на загальний ігровий процес. Завдяки цьому, гравці можуть насолоджуватися глибоким і захоплюючим ігровим досвідом, де кожен рівень і кожна сцена ретельно сплановані та реалізовані з урахуванням всіх деталей.

Накінець розглянемо діаграму класів на рисунку 3.3.

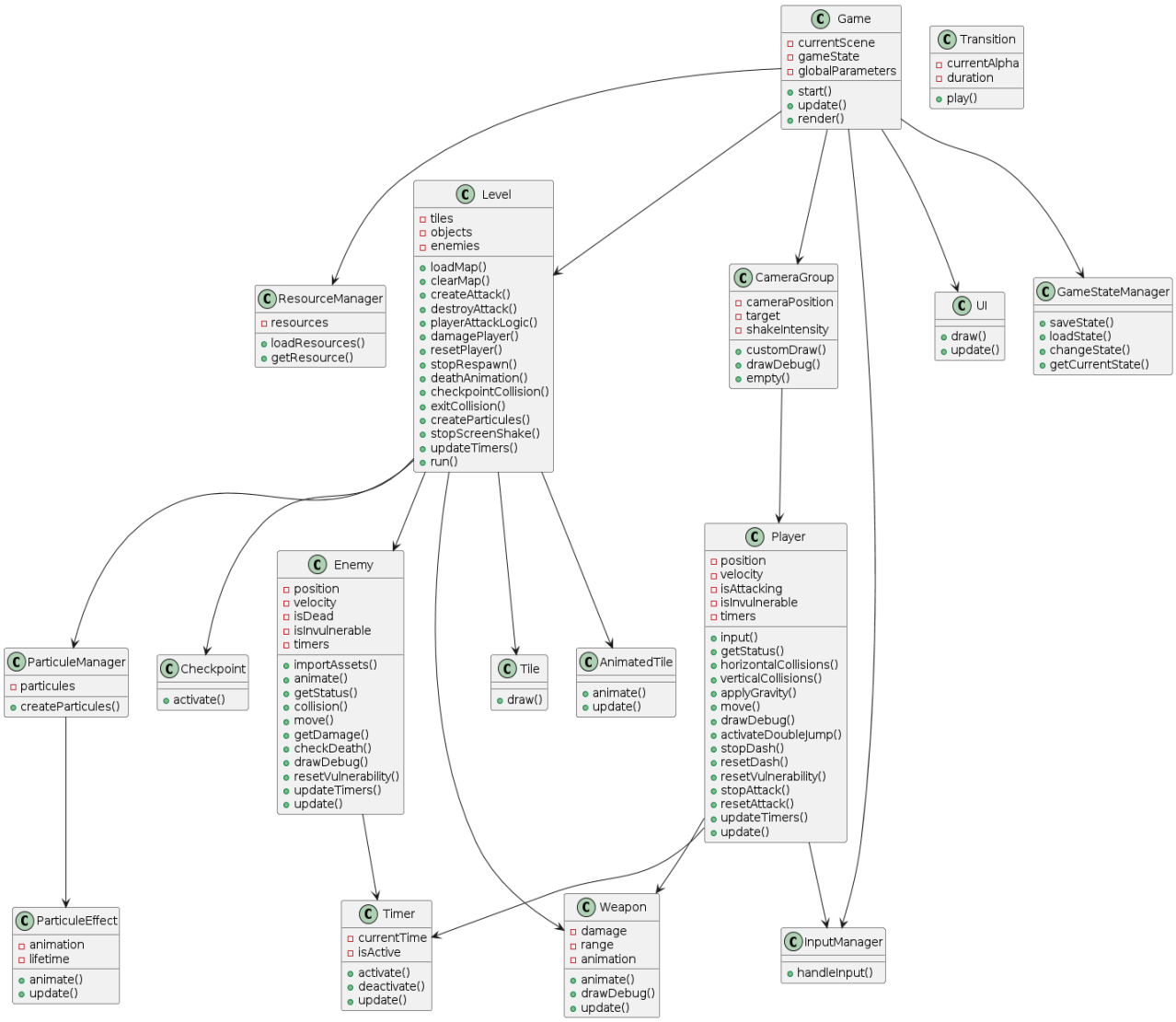

Рис. 3.11 – Діаграма класів гри

3.4 Тестування та відлагодження ігрової системи.

Першим кроком у тестуванні є визначення ключових аспектів гри, які потребують перевірки, що включає в себе перевірку основного ігрового функціоналу, взаємодії персонажа з навколишнім середовищем, колізій, анімацій, звукових ефектів, а також продуктивності гри. Основний ігровий функціонал охоплює всі дії, які може виконувати гравець, такі як переміщення, стрибки, атаки, збирання предметів та взаємодія з об'єктами. Важливо переконатися, що всі ці дії виконуються коректно і без затримок.

Для перевірки переміщення персонажа ми розробили тестовий рівень у Tiled, де гравець може рухатися в усіх напрямках і взаємодіяти з різними об'єктами. Ми використовували функції Pygame для обробки введення з клавіатури та миші, що дозволяє нам тестувати різні сценарії управління персонажем. Зокрема, ми перевіряємо, чи коректно працюють колізії зі стінами, чи можна стрибати через перешкоди та чи коректно відображаються анімації персонажа під час руху. Важливим аспектом тестування є також перевірка плавності руху, оскільки ривки та затримки можуть значно знизити задоволення від гри.

Рис. 3.12 - Тестовий рівень у Tiled, де проводиться тестування переміщення персонажа

Колізії є важливою частиною взаємодії персонажа з навколишнім середовищем. Вони забезпечують реалістичну поведінку об'єктів у грі та запобігають проходженню персонажа крізь стіни та інші перешкоди. Для тестування колізій ми створили серію тестів, де персонаж рухається вздовж стін, натрапляє на різні об'єкти та перевіряє їхні границі. Ми також

використовували функції Pygame для відображення рамок колізій, що дозволяє нам візуально переконатися в правильності їхньої роботи.

Рис. 3.13 - Тестування колізій, де видно рамки колізійних об'єктів

Анімації є важливою частиною візуальної складової гри. Вони роблять гру більш живою та динамічною. Для тестування анімацій ми перевіряли правильність їхнього відтворення в різних ситуаціях, таких як рух, атака, стрибок тощо. Ми використовували спрайти, створені в Paint, і перевіряли їхню зміну в залежності від дій персонажа. Особливу увагу ми приділяли плавності анімацій та відсутності помилок при їхньому відтворенні.

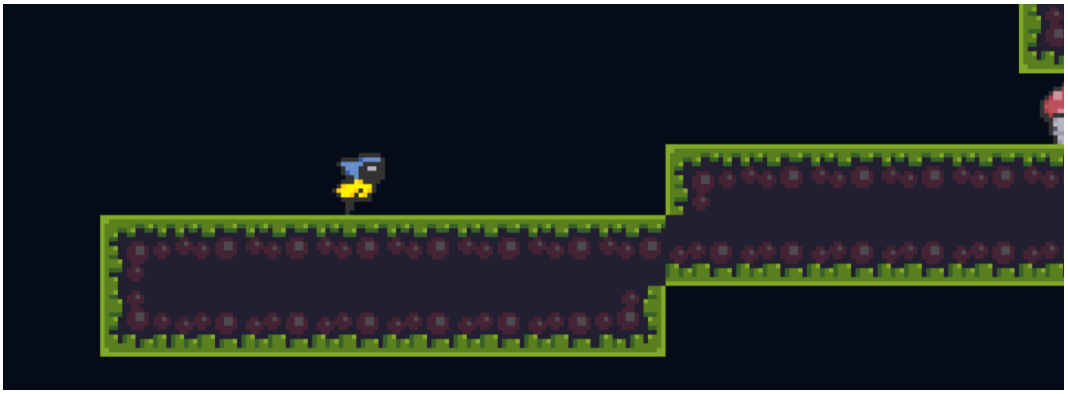

Рис. 3.14 - Анімації персонажа під час атаки та руху

Звукові ефекти додають грі атмосферності та допомагають гравцеві краще орієнтуватися в ігровому світі. Тестування звукових ефектів включає перевірку їхнього відтворення в потрібний момент, коректної гучності та відсутності затримок. Ми перевіряли звукові ефекти для різних дій персонажа, таких як атака, стрибок, збирання предметів тощо. Також ми тестували фонову музику на відповідність загальному стилю гри та її коректне відтворення протягом гри.

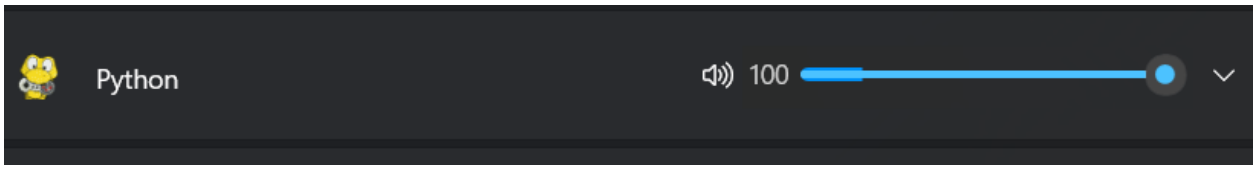

Рис. 3.15 - Перевірка звукових ефектів під час гри

РОЗДІЛ 4. ТЕХНІКО-ЕКОНОМІЧНА ЧАСТИНА

4.1 Маркетингове обґрунтування проєкту

У сучасному ігровому світі жанр Метроідванія користується великою популярністю, і щороку випускається безліч нових проектів у цьому стилі. Часто інвестори вкладають значні кошти, перевищуючи 50 тисяч доларів, у розробку таких ігор, сподіваючись на успіх та високий рівень завантажень. Проте реальність часто буває невтішною: навіть при значних інвестиціях багато таких ігор не можуть досягти і 2 тисячі завантажень. Це свідчить про високий рівень конкуренції та складність у завоюванні уваги гравців.

У цьому контексті наш проект, який розроблений як дипломна робота однією людиною, потребує об'єктивної оцінки його економічної ефективності. Основною метою розробки даної гри було не отримання прибутку, а демонстрація моїх технічних та творчих навичок. Проте, якщо розглядати можливість комерціалізації проекту, необхідно врахувати кілька важливих факторів:

Спочатку, слід визнати, що для досягнення успіху на ринку гра повинна бути більш розвиненою та мати унікальні характеристики, які відрізнятимуть її від численних конкурентів. Це може включати додаткові рівні, розширений сюжет, нові механіки гри, покращену графіку та звук. Важливим є також зворотний зв'язок від потенційних гравців, який допоможе визначити слабкі місця проекту та напрями для покращення.

4.2. Визначення трудомісткості розробки програмного продукту (ПП)

Термін розробки програмного продукту (ПП) залежить від обсягу інформаційної системи (ІС), складності розробки, кваліфікації персоналу і встановлених ринкових термінів. При оцінці трудомісткості розробки ПП використовується обсяг програмних засобів в тисячах умовних машинних

команд програми-аналога. Вибравши аналогічний програмний засіб (ПЗ), який має V0 умовних машинних команд. У цьому проєкті розробляється новий ПП, який відповідає аналогу ПЗ оптимізаційних розрахунків с V□= 7000 умовних машинних команд із трудомісткістю Тр = 350 люд/год. Трудомісткість розроблювального ПП визначається на підставі трудомісткості аналога з урахуванням складності розробки, ступеня новизни й використання в розробці стандартних модулів на підставі формул 4.1 – 4.4:

$$ТТЗ = ТР * L1 * КН \quad (4.1)$$

$$ТТП = ТР * L2 * КН \quad (4.2)$$

$$ТРП = ТР * L3 * КН * КТ \quad (4.3)$$

$$ТВН = ТР * L4 * КН \quad (4.4)$$

де: Тр – укрупнення норма часу на розробку аналога ПЗ, чол/год, що коректується поправочним коефіцієнтом, що враховує умови розробки ПЗ, тобто в умовах комп'ютера, Кн = 0.7, тобто:

$$ТР = 350 * 0.7 = 245 \text{ люд/год}$$

Lj – питома вага j-го етапу розробки (залежно від ступеня новизни й відповідних стадій):

$$L1 = 0{,}10;$$

$$L2 = 0{,}14;$$

$$L3 = 0{,}56;$$

$$L4 = 0{,}17.$$

КН – поправочний коефіцієнт, що враховує ступінь новизни, у цьому випадку КН = 0,7; КТ – поправочний коефіцієнт, що враховує ступінь використання в розробці типових програм КТ = 0,6.

Тоді:

$$ТТЗ = 245 * 0.10 * 0.7 = 17{,}5 \text{ (дні)}$$

$$ТТП = 245 * 0.14 * 0.7 = 24 \text{ (дні)}$$

$$ТРП = 245 * 0.56 * 0.7 * 0.6 = 57 \text{(дні)}$$

$$ТВН = 245 * 0.17 * 0.7 = 29 \text{ (дні)}$$

Тривалість розробки ПП у роках визначається за формулою 4.5:

$$T\text{ПП} = T\text{ТЗ} + T\text{ТП} + T\text{РП} + T\text{ВН} \quad (4.5)$$

де: ТПП – сумарна тривалість розробки, розрахуємо:

$$T\text{ПП} = 17{,}5 + 24 + 57 + 29 = 127{,}5 \text{ (дні)} = 0{,}364 \text{ (р.)}$$

4.2.2 Визначення ціни ПП

Оскільки ПП розглядається й створюється як продукція виробничотехнічного призначення, що допускає багаторазове тиражування й відчуження від безпосередніх розроблювачів, значить користуємося формулою 4.6:

$$Ц = С * К * Пр \quad (4.6)$$

де: С – витрати на розробку програмної продукції (кошторисна собівартість); К – коефіцієнт обліку витрат на виготовлення досвідченого зразка ПП як продукції виробничо-технічного призначення. Пр – нормативний прибуток, що розраховується по формулі 4.7:

$$П_Р = (С - С_М) * Рн / 100 \quad (4.7)$$

де: Рн– норматив рентабельності, 25%; См– матеріальні витрати, грн./вироб.

Витрати на розробку програмної продукції можуть бути представлені у вигляді кошторису витрат, що включає в себе наступні статті: 1. Матеріали. Витрати на матеріали визначаються по формулі 4.8:

$$С_М = К_{МР} * \sum Цi * V_i \quad (4.8)$$

де: $К_{МР}$ – коефіцієнт транспортно-заготівельних видатків;

Цi – ціна одиниці і-го матеріалу, грн.;

Vi – придбана кількість і-го матеріалу.

В таблиці 4.1 представлено витрати на матеріали.

Таблиця 4.1

Найменування Товару	Опис матеріалу	Кількість	Ціна за одиницю, грн.	Сума, грн.
Упаковка паперу	Папір офісний А4 80 г/м2	1	201	205
Флешнакопичувач	Флеш пам'ять USB Kingston Kyson 32GB	1	289	250
Картридж для принтеру	Картридж Canon PG-46 PIXMA Ink	1	519	530
Усього				985
КМР = 0.1				98.5
Разом:				1 083.5

2. Спеціальні устаткування.

Витрати, які пов’язані з використанням обчислювальної техніки, визначаються по формулі 4.9:

$$C^{ЕОМ} = t^{ЕОМ} * К_{И}{}^{ЕОМ} * Ц^{ЕОМ} * К_{Е}{}^{ЕОМ} * К_{БД}{}^{ЕОМ} \quad (4.9)$$

де:

$t^{ЕОМ}$ – час використання ЕОМ для розробки даного ПО, год (160);

$К_{И}{}^{ЕОМ}$ – поправочний коефіцієнт обліку часу використання ЕОМ (1,10);

$Ц_{ЕОМ}$ – ціна 1-ої години роботи ЕОМ, грн. (10);

КЕ ЕОМ – 1,0;

КБД ЕОМ – 1,0 (не використовується).

Тоді:

$$C_{ЕОМ} = 1083.5 * 1.10 * 10 = 1\ 191.85 \text{ грн.}$$

3. Основна заробітна плата

У статтю включається основна заробітна плата двох виконавців, безпосередньо зайнятих розробкою даного ПП (керівник, нормо контроль), з обліком їхнього посадового окладу (6800 та 12000 грн. відповідно) і часу участі в розробці. Розрахунок ведеться по формулі 4.10:

$$С_{ЗО}= \sum З_i * К_0 \, \tau_i \qquad (4.10)$$

де: $З_i$ – середньомісячний оклад і-го виконавця, грн.;

$Д_р$ – середня кількість робочих днів у місяці (20);

τ_i – трудомісткість робіт, виконуваних і-м виконавцем. Люд/дні.

Тоді:

$С_{ЗО}$= 6800 86 / 20 = 29 240 грн. (розробка)

$С_{ЗО}$= 12 000 * 0,1 * 31 / 20 = 1 860 грн. (розробка)

$С_{ЗО}$ = 26 581 + 1 690 = 31 100 грн. (розробка)

4. Додаткова заробітна плата. Розрахунок по формулі 4.11:

$$СЗД = С_{ЗО} * К_д \qquad (4.11)$$

де: КД – коефіцієнт відрахувань на додаткову заробітну плату (0,1).

$$СЗД = 31\ 100 * 0,1 = 3110 \text{ грн (загальне)}$$

5. Відрахування на соціальне страхування. У статті враховуються відрахування в бюджет соціального страхування по встановленому законодавством тарифу від суми основних й додаткової заробітної плати, тобто для виконання розрахунків треба використати формулу 4.12:

$$С_{СС} = К_{СС} * (С_{ЗО} + С_{ЗД}) \qquad (4.12)$$

де: КСС – коефіцієнт відрахувань на соціальне страхування (22%).

Тоді:

$$С_{СС} = 0,22 * (31100+ 3110) = 7528.4 \text{ грн 6.}$$

6. Накладні витрати

У статті враховуються витрати на загальногосподарські витрати, позавиробничі (комерційні) витрати й витрати на керування. Накладні витрати визначають у відсотковому відношенні до основної заробітної плати за формулою 4.13:

$$С_Н = К_Н * С_{ЗО} \quad (4.13)$$

де: КН – коефіцієнт накладних видатків (50%).

Тоді:

$$С_Н = 0,5 * 31100 = 15555 \text{ грн.}$$

Результати розрахунку кошторисної вартості ПП можна продемонструвати у вигляді таблиці в табл. 4.2.

Таблиця 4.2

Результати розрахунку кошторисної вартості ПП

Найменування статті	Значення собівартості, грн	Питома вага, %
Матеріали	1103.4	1.6
Спеціальне устаткування	9589.54	14.1
Основна заробітна плата	31100	45.7
Додаткова заробітна плата	3110	4.6
Відрахування на соціальне страхування	7528.40	11.1
Накладні витрати	15555	22.9
Разом	67992.84	100

Тепер можемо розрахувати формули описані вище:

$$С = С_М + С_{ЕОМ} + С_{ЗО} + С_{ЗД} + С_{СС} + С_Н = 67992.84 \text{ грн.}$$

$$ПР = (67992.84 – 1103.4) * 0.25 = 16722.36 \text{ грн.}$$

$$Ц = 1.1 * 67992.84 + 16722.36 = 91512.28 \text{ грн}$$

Таблиця 4.3.

Основні аспекти для покращення гри

Аспект	Опис	Можливі покращення
Геймплей	Основні механіки гри	Додавання нових рівнів та механік
Графіка	Візуальне оформлення	Покращення якості графіки, деталізація
Звук	Аудіосупровід	Вдосконалення звукових ефектів та музики
Сюжет	Історія гри	Розширення та поглиблення сюжету
Зворотний зв'язок	Відгуки гравців	Врахування побажань та зауважень користувачів

Далі, для виходу на ринок необхідно провести маркетингову кампанію, яка включатиме просування гри у соціальних мережах, на ігрових платформах та через різні канали комунікації. Маркетингова стратегія має бути орієнтована на цільову аудиторію та враховувати особливості ринку ігор Метроідванія.

Таблиця 4.4

Основні етапи маркетингової кампанії

Етап	Опис	Ціль
Дослідження ринку	Аналіз конкурентів та цільової аудиторії	Визначення ніші для гри
Соціальні мережі	Просування через Facebook, Instagram, Twitter	Залучення гравців та створення спільноти
Ігрові платформи	Розміщення гри на Steam, GOG, Epic Games	Розширення доступності гри
Реклама	Використання рекламних кампаній у Google Ads та інших платформах	Залучення нових користувачів

Тобто, якщо розглядати наш проект з точки зору комерціалізації, необхідно враховувати високу конкуренцію та складність у досягненні значного успіху на ринку. Ми, які зробили цю гру як дипломну роботу, маємо реальні можливості для вдосконалення проекту. Важливо приділити увагу покращенню геймплею, графіки, звуку та сюжету, а також розробити ефективну маркетингову стратегію. Лише після цих кроків можна сподіватися на комерційний успіх та значні завантаження гри.

Підсумовуючи, наш проект, створений як навчальна робота, має потенціал для комерційного успіху за умови його подальшого розвитку та вдосконалення. Оцінка економічної ефективності повинна враховувати не тільки поточні досягнення, але й перспективи для покращення та виходу на ринок.

4.3 Ризики та способи їх мінімізації

Розробка комп'ютерних ігор, особливо у жанрі Метроідванія, завжди пов'язана з певними ризиками, які можуть істотно вплинути на успіх проекту. Основні ризики, з якими може зіткнутися проект, включають:

- Технічні проблеми
- Затримки у розробці
- Недостатнє фінансування
- Висока конкуренція на ринку
- Зміна ринкових умов
- Людський фактор
- Захист інтелектуальної власності
- Вплив зовнішніх чинників

Розглянемо їх докладніше:

1) Технічні проблеми. Помилки у коді та проблеми з інтеграцією компонентів можуть значно вплинути на функціональність та стабільність гри. Для мінімізації цього ризику необхідно проводити регулярне тестування всіх

компонентів гри на всіх етапах розробки. Використання сучасних інструментів для управління проектом та контролю якості, таких як системи контролю версій (наприклад, Git) та платформи для автоматизованого тестування, допоможе швидко виявляти та виправляти помилки, забезпечуючи стабільність ігрового продукту.

2) Затримки у розробці. Відставання від графіка через неадекватне планування часу або непередбачувані обставини є серйозним ризиком для проекту. Для мінімізації цього ризику необхідно ретельно планувати всі етапи розробки, встановлювати реалістичні терміни та дотримуватися їх. Використання методологій управління проектами, таких як Agile або Scrum, дозволяє гнучко реагувати на зміни та оперативно вирішувати виникаючі проблеми, забезпечуючи ефективне управління часом та ресурсами. Регулярні зустрічі команди, звітність про виконані завдання та адаптація планів відповідно до реальних умов дозволяють уникнути серйозних затримок у розробці.

3) Недостатнє фінансування. Брак коштів може вплинути на реалізацію проекту. Для мінімізації цього ризику необхідно заздалегідь визначити бюджет проекту, враховуючи всі можливі витрати, такі як заробітна плата розробників, вартість обладнання та програмного забезпечення, витрати на маркетинг та просування гри. Також варто розглянути можливість залучення додаткових джерел фінансування, таких як краудфандинг, інвестори або гранти на розробку ігор. Створення детального фінансового плану та регулярний моніторинг витрат дозволять уникнути фінансових проблем та забезпечити стабільне фінансування проекту на всіх етапах його реалізації.

4) Висока конкуренція на ринку. Велика кількість подібних ігор може суттєво вплинути на успіх проекту. Для мінімізації цього ризику необхідно проводити ретельний аналіз ринку та конкурентів, визначати унікальні переваги своєї гри та активно просувати їх. Розробка унікальних ігрових механік, цікавого сюжету, привабливого візуального стилю та

якісного звукового оформлення допоможе виділити гру серед численних конкурентів. Ефективна маркетингова стратегія, орієнтована на цільову аудиторію, дозволить залучити увагу гравців та забезпечити високий рівень завантажень гри.

5) Зміна ринкових умов. Зміни в уподобаннях гравців, поява нових технологій або змін у законодавстві можуть створити нові виклики для розробників. Для мінімізації цього ризику необхідно бути гнучким та готовим адаптуватися до нових умов. Регулярний моніторинг ринку, аналіз тенденцій та своєчасне впровадження нових технологій дозволять залишатися конкурентоспроможними та швидко реагувати на зміни у зовнішньому середовищі.

6) Людський фактор. Зміни у складі команди розробників, конфлікти або низький рівень мотивації можуть негативно вплинути на хід проекту. Для мінімізації цього ризику необхідно створити сприятливі умови для роботи команди, забезпечити ефективну комунікацію та мотивацію співробітників. Регулярні зустрічі, командні тренінги, гнучкий графік роботи та стимулювання досягнень допоможуть підтримувати високий рівень залученості та продуктивності команди.

7) Захист інтелектуальної власності. Незаконне копіювання або використання гри без дозволу можуть завдати значних фінансових збитків. Для мінімізації цього ризику необхідно вживати заходів для захисту авторських прав, такі як реєстрація торгових марок, використання ліцензійних угод та інших правових механізмів.

8) Вплив зовнішніх чинників. Економічна нестабільність, політичні зміни або природні катастрофи можуть мати значний вплив на реалізацію проекту та його фінансові показники. Для мінімізації цього ризику необхідно мати резервні плани та гнучкість у прийнятті рішень. Створення фінансового резерву, страхування ризиків та регулярний аналіз зовнішнього середовища допоможуть бути готовими до непередбачуваних обставин.

Таблиця 4.5.

Основні ризики та способи їх мінімізації

Ризик	**Опис**	**Способи мінімізації**
Технічні проблеми	Помилки у коді, проблеми з інтеграцією	Регулярне тестування, сучасні інструменти
Затримки у розробці	Відставання від графіка	Планування та управління проектом
Недостатнє фінансування	Брак коштів	Пошук додаткових джерел фінансування
Конкуренція	Висока конкуренція на ринку	Аналіз ринку, унікальність продукту
Зміна ринкових умов	Зміни в уподобаннях споживачів	Гнучкість та адаптація до ринкових умов
Людський фактор	Зміни у складі команди, конфлікти	Сприятливі умови роботи, ефективна комунікація
Захист інтелектуальної власності	Незаконне копіювання, використання	Реєстрація торгових марок, ліцензійні угоди
Вплив зовнішніх чинників	Економічна нестабільність, політичні зміни	Резервні плани, страхування ризиків

Підсумовуючи, можна сказати, що успішна реалізація проекту з розробки двовимірної комп'ютерної гри жанру Метроідванія залежить від багатьох факторів. Ретельний аналіз можливих ризиків та розробка ефективних стратегій для їх мінімізації є необхідними кроками для забезпечення стабільного та успішного виконання проекту. Врахування технічних, фінансових, ринкових, людських та зовнішніх чинників дозволить знизити ризики та підвищити шанси на успіх проекту. Правильне планування, управління та гнучкість у прийнятті рішень допоможуть ефективно реагувати на виклики та забезпечити високий рівень якості кінцевого продукту.

РОЗДІЛ 5. ОХОРОНА ПРАЦІ

Охорона праці є невід'ємною складовою будь-якого проекту, оскільки вона спрямована на забезпечення безпеки та здоров'я працівників. У контексті розробки комп'ютерних ігор, де основна діяльність пов'язана з роботою за комп'ютером, особливу увагу слід приділити організації робочого місця, режиму праці та відпочинку, а також дотриманню правил техніки безпеки.

5.1 Організація робочого місця

Організація робочого місця розробників комп'ютерних ігор є основним фактором охорони праці. Робоче місце повинно бути обладнане таким чином, щоб забезпечити комфорт і зручність під час роботи, а також мінімізувати негативний вплив на здоров'я працівників. Важливими елементами організації робочого місця є:

1) Робоче місце та обладнання. Робоче місце повинно бути обладнане ергономічним стільцем з регулюванням висоти та кута нахилу спинки, що дозволяє підтримувати правильну осанку. Стіл повинен мати достатню площу для розміщення монітора, клавіатури, миші та інших необхідних предметів. Монітор повинен бути встановлений на рівні очей, щоб знизити навантаження на шию та очі.

2) Освітлення. Освітлення робочого місця має бути достатнім і рівномірним. Рекомендується використовувати комбіноване освітлення, що включає загальне та місцеве (настільну лампу). Світло повинно бути спрямоване так, щоб не створювати відблисків на екрані монітора.

3) Мікроклімат. Робоче місце повинно мати комфортні умови мікроклімату. Температура повітря повинна бути в межах 20-24°C, вологість – 40-60%. Необхіхно забезпечити регулярне провітрювання приміщення для підтримки свіжості повітря.

4) Організація простору. Робоче місце повинно бути організоване

таким чином, щоб забезпечити достатньо простору для руху та зручний доступ до всіх необхідних предметів. Розташування обладнання та меблів повинно мінімізувати необхідність часто нахилятися або повертатися.

5.2 Режим праці та відпочинку

Режим праці та відпочинку є ключом для підтримки високого рівня працездатності та здоров'я працівників.

Основні рекомендації щодо режиму праці та відпочинку включають:

1. Чергування роботи та відпочинку.

Рекомендується робити короткі перерви кожні 45-60 хвилин роботи за комп'ютером. Під час перерв слід виконувати прості фізичні вправи для зняття напруги з м'язів, а також вправи для очей для зменшення навантаження на зір.

2. Тривалість робочого дня.

Тривалість робочого дня не повинна перевищувати 8 годин. У разі необхідності роботи понаднормово, повинні бути передбачені додаткові перерви та компенсації.

3. Організація робочого графіка.

Робочий графік повинен враховувати біоритми працівників та забезпечувати достатньо часу для відпочинку та сну. Рекомендується уникати роботи у нічний час, оскільки це може негативно впливати на здоров'я та продуктивність працівників.

5.3 Дотримання правил техніки безпеки

Дотримання правил техніки безпеки є необхідним для запобігання нещасним випадкам та забезпечення безпеки працівників.

Основні правила техніки безпеки включають те, що:

1) Необхідно забезпечити правильне підключення та заземлення електричного обладнання. Розетки та електричні дроти повинні бути в

справному стані та відповідати вимогам безпеки. Забороняється використання несправного або пошкодженого електрообладнання.

2) Робоче місце повинно бути обладнане засобами пожежогасіння, такими як вогнегасники та протипожежні ковдри. Необхідно дотримуватися правил пожежної безпеки, зокрема, уникати перевантаження електричних мереж та правильно зберігати легкозаймисті матеріали.

3) Виконання рекомендацій щодо ергономіки робочого місця дозволяє запобігти розвитку захворювань опорно-рухового апарату та зору. Правильна організація робочого місця, дотримання режиму праці та відпочинку, а також регулярне виконання фізичних вправ сприяють збереженню здоров'я працівників.

ВИСНОВКИ

В першому розділі було досліджено жанр відеоігор Метроідванія, його історію та еволюцію, а також найвідоміші зразки цього жанру. Метроідванія поєднує в собі елементи пригодницьких ігор, платформерів та головоломок, при цьому надаючи особливий акцент на не лінійність світу та його поступове розкриття завдяки новим здібностям гравця.

Було визначенно, що цей жанр розпочав свій розвиток у 1980-х роках з виходом гри «Metroid» для Nintendo NES. У цій грі були уперше запропоновані нелінійний дизайн рівнів та принципи «розблокування», які згодом стали фундаментальними для жанру. Водночас зародження метроідванії пов'язано і з серією «Castlevania», зокрема із грою «Castlevania: Symphony of the Night», випущеною у 1997 році, яка досягла визначального успіху для жанру, додавши йому глибини за допомогою сюжету та атмосфери.

Сучасні представники жанру переважно зберігають його фундаментальні елементи, такі як нелінійність, дослідження світу та розблокування нових можливостей, продовжуючи розвивати та вдосконалювати його. Так, такі тайти як «Ori and the Blind Forest», «Hollow Knight», «Salt and Sanctuary» досягли визнання критиків і гравців та встановили нові стандарти якості. Проте створення якісної метроідванії залишається складним завданням, що вимагає глибокого дизайну та балансу.

Водночас жанр продовжує активно розвиватися та еволюціонувати, переходячи від двовимірних до тривимірних світів, а також збагачуючи ігровий процес новими механіками. Це демонструє його життєздатність і здатність як приваблювати нових розробників та аудиторію гравців, пропонуючи унікальний досвід дослідження та постійного прогресу гравця всередині складного ігрового світу.

У розділі 2 розглянуто основні аспекти розробки та реалізації концептуального документа для комп'ютерної гри жанру Метроідванія, створеної з використанням мови програмування Python та бібліотеки PyGame.

Розділ розпочинається з опису загальної характеристики проекту, метою якого є створення прототипу гри для освоєння основ програмування ігор. Було розроблено ігровий процес, який включає керування персонажем на двовимірному екрані з можливістю пересування, стрибків та взаємодії з простими об'єктами і ворогами.

Однією з важливих частин цього розділу є розробка концептуального документа, в якому викладено основні завдання проекту, такі як створення ігрового рушія на основі Python/PyGame, реалізація персонажа та його керування, додавання інтерактивних елементів, розробка ігрового процесу та проведення тестування і налагодження. У результаті цієї роботи було отримано працюючу демонстраційну гру, яка дозволяє гравцю керувати персонажем, переміщатися по двовимірному світу, взаємодіяти з об'єктами та боротися з ворогами. Гра орієнтована на ПК і підтримує управління за допомогою клавіатури або геймпада.

Нами було детально описано ігровий світ і сюжет, який розгортається навколо молодого чаклуна Мерлін та лицаря Галгамора, що об'єднали зусилля для боротьби з грибними чудовиськами, які тероризують мешканців лісу. Сюжет гри включає дослідження лісу, бій з ворогами та розгадування таємниць, щоб з'ясувати причину появи чудовиськ та звільнити ліс від прокляття.

Розробка геймплею та механік гри включала реалізацію основних елементів, таких як пересування персонажа, атака ворогів, збір предметів та переходи між рівнями. Механіки руху передбачають можливість бігу, стрибків, подвійного стрибка та швидкого рівня, що забезпечує динамічність ігрового процесу. Механіки бою дозволяють гравцю атакувати ворогів у різних напрямках, завдаючи їм шкоди та знищуючи їх. Взаємодія з ігровими об'єктами включає зіштовхування з перешкодами, використання контрольних точок та переходи між рівнями через спеціальні виходи.

Також було детально описано дизайн локацій та рівнів гри. Було розроблено три основні рівні: «Map Test», «Map Test2» та «Map Test3», які

включають різні види перешкод, ворогів та ігрових об'єктів. Кожен рівень має свої особливості та завдання, такі як досягнення кінцевої точки, уникнення ворогів та пасток, а також збирання предметів.

У розділі було здійснено вибір та обґрунтування програмних засобів реалізації проекту, а також докладно описано процес розробки графічної та програмної частин гри жанру Метроідванія. Важливим етапом стало обґрунтування вибору мови програмування Python та бібліотеки PyGame, які було обрано завдяки їхнім численним перевагам, таким як простота використання, велика спільнота розробників та широкий набір функціональних можливостей.

Програмна частина гри реалізована на основі об'єктно-орієнтованого підходу, що дозволило структурувати код у вигляді окремих класів, кожен з яких відповідає за конкретну функціональність, що забезпечило модульність, легкість у підтримці та можливість подальшого розширення гри. Використання таких об'єктів, як Surface та Sprite, спростило роботу з графічними ресурсами, дозволивши легко реалізувати анімації та рухомі об'єкти.

Для графічної частини було використано бібліотеку PyGame, а також графічний редактор Paint, що дозволило створити піксельну графіку, яка відповідає ретро-стилю гри. Редактор карт Tiled значно спростив процес створення ігрових рівнів та їх інтеграцію в гру. Використання тайл-сетів та об'єктів у Tiled дозволило швидко створювати великі ігрові простори з високим рівнем деталізації.

У процесі реалізації основного функціоналу коду було розроблено класи, які відповідають за різні аспекти ігрового процесу: керування гравцем, взаємодія з ворогами, обробка подій та колізій, а також відображення інтерфейсу користувача. Кожен клас має свої методи та властивості, що дозволяють ефективно виконувати поставлені завдання та забезпечувати динамічний ігровий процес.

Наприклад: Для управління гравцем була реалізована окрема клас Player, який відповідає за рухи, анімації та взаємодію з іншими об'єктами гри. Клас Player використовує модулі Pygame для обробки клавішних подій, завантаження та відтворення анімацій та обробки зіткнень з іншими об'єктами, що дозволяє створити гнучку та масштабовану систему управління гравцем, яку можна легко розширювати та модифікувати.

Для управління ворогами була створен клас Enemy, який відповідає за рухи, анімації та взаємодію з гравцем. Клас Enemy використовує схожі методи та принципи, що й клас Player, що забезпечує уніфікованість та зручність у розробці. Крім того, надалі, вороги можуть мати різні типи поведінки та анімацій, що дозволяє створювати різноманітні сценарії взаємодії з гравцем.

Частинки та ефекти були реалізовані за допомогою класу ParticuleManager, який відповідає за створення та управління частинками у грі. Частинки використовуються для створення візуальних ефектів, таких як вибухи, стрибки, удари тощо, що додає грі динамічності та забезпечує візуальне підкріплення дій гравця та ворогів.

Для управління рівнями та їх завантаження був створен клас Level, який відповідає за завантаження карт, розміщення об'єктів та управління взаємодією між ними. Клас Level використовує модуль pytmx для завантаження карт, створених у редакторі Tiled, та розміщення об'єктів на основі даних з цих карт, що дозволяє легко створювати та модифікувати рівні без необхідності змінювати код гри.

Інтерфейс користувача був реалізований за допомогою класу UI, яка відповідає за відображення елементів інтерфейсу, таких як здоров'я гравця, таймери та інші показники. Класа UI використовує методи Pygame для відображення зображень та тексту на екрані, що дозволяє створити зручний та інформативний інтерфейс для гравця.

Проведене тестування та відлагодження ігрової системи включало перевірку ключових аспектів гри, таких як переміщення персонажа, взаємодія з навколишнім середовищем, анімації, звукові ефекти та продуктивність гри.

Було виявлено та виправлено помилки, що забезпечило стабільну роботу гри та високий рівень якості кінцевого продукту. Тестування переміщення персонажа, колізій та анімацій дозволило переконатися в коректності їхньої роботи та плавності виконання.

Саме так було досягнуто основної мети проекту - створення працюючого прототипу гри, який демонструє основні механіки жанру та забезпечує захоплюючий ігровий досвід. Отримані результати свідчать про можливість подальшого розширення та вдосконалення гри, що дозволить створити повноцінний комерційний продукт. Вивчені методи та інструменти можуть бути використані для розробки інших ігрових проектів, забезпечуючи високу ефективність та якість роботи.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Arroyo, S., & Gonzalez, J. L. (2019). A retrospective on the Metroidvania genre: From Metroid to Ori and the Blind Forest. ACM Transactions on Interactive Intelligent Systems, 9(2-3), 1-35. https://doi.org/10.1145/3301275 (дата звернення: 14.05.2024).

2. Burger, J., Kwan, J., & Loprior, M. (2021). Level design in Metroidvanias: A spatial analysis and recommendations. ACM Transactions on Interactive Intelligent Systems, 11(1), 1-28. https://dl.acm.org/journal/tiis (дата звернення: 14.05.2024).

3. Rodriguez, E. (2020). Storytelling in Metroidvanias: Fragmentation, ambiguity, and immersion. In Narrative Across Interactive Media (pp. 125-148). MIT Press. https://doi.org/10.7551/mitpress/11337.003.0009 (дата звернення: 14.05.2024).

4. Plunkett, C. (2022, January 28). The enduring appeal of Metroidvanias. Polygon. https://www.polygon.com/22901679/metroidvania-games-appeal-analysis (дата звернення: 14.05.2024).

5. McEntire, C. (2019, March). Architecture of Metroidvania worlds. Game Developer Magazine, 26(2), 16-21. https://www.gdcvault.com/play/1026021/Architecture-of-Metroidvania-Worlds (дата звернення: 14.05.2024).

6. Nordin, D. (2018). Level design for non-linear room-based games. In Proceedings of the Game Development Conference (pp. 23-32). Association for Computing Machinery. https://doi.org/10.1145/3235765.3235785 (дата звернення: 14.05.2024).

7. Kauffeldt, A., & Wattiau, N. (2022). Creating a balanced experience in Metroidvania games using critical path regulation systems. IEEE Transactions on Games, 14(2), 162-169.

8. Thompson, J. (2020). Analyzing the atmosphere and aesthetics of the Metroidvania genre. Journal of Games Criticism, 4(1), 18-36.

https://gamescriticism.org/articles/thompson-4-1 https://gamescriticism.org/ (дата звернення: 14.05.2024).

9. Super Metroid. URL: https://en.wikipedia.org/wiki/Super_Metroid (дата звернення: 15.05.2024).

10. Super Mario Bros. Wonder. URL: https://www.nintendo.com/en-gb/Games/Nintendo-Switch-games/Super-Mario-Bros-Wonder-2404150.html (дата звернення: 15.05.2024).

11. Ender Magnolia Bloom in the Mist: Early access, story, gameplay, platforms. URL: https://www.oneesports.gg/gaming/ender-magnolia-early-access-platforms/ (дата звернення: 15.05.2024).

12. 10 ігор у жанрі метроідванія. URL: https://blog.allo.ua/ua/10-igor-u-zhanri-metroyidvaniya_2019-03-48/ (дата звернення: 15.05.2024).

13. Product-Specific Information. URL: https://www.jetbrains.com/business/documents/#product=rider (дата звернення: 15.05.2024).

14. Academy i., Language P. Python : Programming: Your Step By Step Guide To Easily Learn Python in 7 Days. Independently Published, 2017. 97 с.

15. Delisle M. PhpMyAdmin Starter. Packt Publishing, Limited, 2012.

16. Editor's Notepad. Sign Language and Linguistics. 2021. Т. 24, № 1. С. 1–3. URL: https://doi.org/10.1075/sll.00059.not (дата звернення: 16.05.2024).

17. Python / R. Toal та ін. Programming Language Explorations. 2016. С. 59–76. URL: https://doi.org/10.1201/9781315314334-4 (дата звернення: 16.05.2024).

18. Rossum G. v., Team P. D. The Python Language Reference: Release 3.6.4. 12th Media Services, 2018. 168 с.

19. Thompson M. J., Language P. Python Programming for Intermediates: Learn the Basics of Python in 7 Days!. Independently Published, 2018.

20. Архітектура та проектування програмного забезпечення. URL: http://dspace.wunu.edu.ua/jspui/bitstream/316497/24194/1/%D0%BE%D0%BF%D0%BE%D1%80%D0%BD%D0%B8%D0%B9%20%D0%BA%D0%BE%D0%

BD%D1%81%D0%BF%D0%B5%D0%BA%D1%82%20%D0%BB%D0%B5%D0%BA%D1%86%D1%96%D0%B9.pdf (дата звернення: 17.05.2024).

21. Georgieva R. GAME-BASED PROGRAMMING TEACHING FOR BEGINNERS IN PYGAME ZERO MODE – SAMPLE PYTHON TASKS. PART II – PYGAME LIBRARY. Mathematics and Informatics. 2023. Т. 66, № 3. С. 244–255. URL: https://doi.org/10.53656/math2023-3-3-gam (дата звернення: 17.05.2024).

22. Gravitational Dynamics in the Solar System: A Pygame Simulation. International Research Journal of Modernization in Engineering Technology and Science. 2023. URL: https://doi.org/10.56726/irjmets47595 (дата звернення: 17.05.2024).

23. Hunt J. StarshipMeteors pygame. Advanced Guide to Python 3 Programming. Cham, 2019. С. 141–162. URL: https://doi.org/10.1007/978-3-030-25943-3_13 (дата звернення: 17.05.2024).

24. Hunt J. StarshipMeteors Pygame. Advanced Guide to Python 3 Programming. Cham, 2023. С. 225–244. URL: https://doi.org/10.1007/978-3-031-40336-1_22 (дата звернення: 17.05.2024).

25. Introducing Pygame. Beginning Game Development with Python and Pygame. Berkeley, CA. С. 41–66. URL: https://doi.org/10.1007/978-1-4302-0325-4_3 (дата звернення: 17.05.2024).

26. Kinsley H., McGugan W. Introducing Pygame. Beginning Python Games Development. Berkeley, CA, 2015. С. 39–60. URL: https://doi.org/10.1007/978-1-4842-0970-7_3 (дата звернення: 17.05.2024).

27. Shinde P. N. Pygame: Develop Games using Python. International Journal for Research in Applied Science and Engineering Technology. 2021. Т. 9, № VI. С. 4442–4447. URL: https://doi.org/10.22214/ijraset.2021.35735 (дата звернення: 17.05.2024).

28. Song B. Game Development with Python Using Pygame. Proceedings of the International Conference on Information Economy, Data Modeling and Cloud Computing, ICIDC 2022, 17-19 June 2022, Qingdao, China, м. Qingdao, People's

Republic of China, 17–19 черв. 2022 р. 2022. URL: https://doi.org/10.4108/eai.17-6-2022.2322877 (дата звернення: 17.05.2024).

29. Watkiss S. Creating Computer Games. Beginning Game Programming with Pygame Zero. Berkeley, CA, 2020. С. 1–10. URL: https://doi.org/10.1007/978-1-4842-5650-3_1 (дата звернення: 17.05.2024).

30. Watkiss S. Pygame Zero. Beginning Game Programming with Pygame Zero. Berkeley, CA, 2020. С. 51–89. URL: https://doi.org/10.1007/978-1-4842-5650-3_3 (дата звернення: 17.05.2024).

31. Tiled | Flexible level editor. URL: https://www.mapeditor.org/ (дата звернення: 17.05.2024).

ДОДАТКИ

Додаток А

main.py:

```
import pygame

# Ініціалізація Pygame
pygame.init()

# Налаштування вікна
WINDOW_WIDTH = 800
WINDOW_HEIGHT = 600
window = pygame.display.set_mode((WINDOW_WIDTH, WINDOW_HEIGHT))
pygame.display.set_caption(«Головне меню»)

# Кольори
BLACK = (0, 0, 0)
WHITE = (255, 255, 255)
GRAY = (128, 128, 128)

# Шрифт
font = pygame.font.Font(None, 36)

# Функція для відображення кнопки
def draw_button(text, x, y, width, height, color, hover_color):
    mouse_pos = pygame.mouse.get_pos()
    button_rect = pygame.Rect(x, y, width, height)
    if button_rect.collidepoint(mouse_pos):
        pygame.draw.rect(window, hover_color, button_rect)
    else:
        pygame.draw.rect(window, color, button_rect)
    text_surface = font.render(text, True, WHITE)
    text_rect = text_surface.get_rect()
    text_rect.center = (x + width // 2, y + height // 2)
    window.blit(text_surface, text_rect)

# Функція для обробки подій меню
def menu_loop():
    running = True
    while running:
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                running = False
            elif event.type == pygame.MOUSEBUTTONDOWN:
                mouse_pos = pygame.mouse.get_pos()
                if play_button_rect.collidepoint(mouse_pos):
                    # Запуск основної гри
                    from src.game import Game
                    game = Game()
                    game.run()
                    running = False
                elif quit_button_rect.collidepoint(mouse_pos):
```

```
                        running = False
        # Відображення меню
        window.fill(BLACK)
        draw_button(«Зайти в гру», 200, 200, 400, 100, (0, 200,
0), (0, 255, 0))
        draw_button(«Вийти», 200, 400, 400, 100, (200, 0, 0),
(255, 0, 0))
        pygame.display.update()

    pygame.quit()

# Розміри кнопок
button_width = 400
button_height = 100

# Розташування кнопок
play_button_rect = pygame.Rect(200, 200, button_width,
button_height)
quit_button_rect = pygame.Rect(200, 400, button_width,
button_height)

# Запуск меню
print('Слава Україні!')
menu_loop()
```

weapon.py:

```
# Імпортуємо бібліотеку Pygame
import pygame

# Імпортуємо клас Player, константи BASE_DIR та LAYERS, а також
функцію import_folder
from src.player import Player
from src.settings import BASE_DIR, LAYERS
from src.support import import_folder

# Клас Weapon представляє зброю гравця
class Weapon(pygame.sprite.Sprite):
    def __init__(self, player: Player, direction: str, groups:
list[pygame.sprite.Group], z: int = LAYERS['main']):
        super().__init__(groups)
        direction_player = player.status.split('_')[0]

        # Анімація
        full_path = BASE_DIR / «graphics» / «player» /
f»{direction}_sword_effect»
        self.frames = import_folder(full_path)
        self.speed_animation = 4 * 3
        self.frame_index = 0

        # Ініціалізація
        self.image = self.frames[self.frame_index]
        self.z = z
        if direction == 'right':
```

```
            self.rect =
self.image.get_rect(topleft=player.rect.topright)
        elif direction == 'bottom':
            self.rect =
self.image.get_rect(topleft=player.rect.bottomleft)
        elif direction == 'top':
            self.rect =
self.image.get_rect(bottomleft=player.rect.topleft)
        else:
            self.rect =
self.image.get_rect(topright=player.rect.topleft)

    def animate(self, dt: float):
        # Анімує зброю
        self.frame_index += self.speed_animation * dt
        if self.frame_index >= len(self.frames):
            self.frame_index = 0
        self.image = self.frames[int(self.frame_index)]

    def draw_debug(self, display_surface: pygame.Surface,
offset: pygame.math.Vector2):
        # Відображає прямокутник для налагодження
        offset_rect = self.rect.copy()
        offset_rect.topleft -= offset
        pygame.draw.rect(display_surface, 'white', offset_rect,
3)

    def update(self, dt: float):
        # Оновлює анімацію зброї
        self.animate(dt)


UI.py:
# Імпортуємо бібліотеку Pygame
import pygame

# Імпортуємо клас Player та константу BASE_DIR
from src.player import Player
from src.settings import BASE_DIR

# Клас UI представляє інтерфейс користувача
class UI:
    def __init__(self):
        # Ініціалізація
        self.display_surface = pygame.display.get_surface()

        # Завантаження зображень для відображення здоров'я
        self.full_heart_surf = pygame.image.load(BASE_DIR /
«graphics» / «heart» / «hearts_hud.png»).convert_alpha()
        self.full_heart_surf =
pygame.transform.scale(self.full_heart_surf, (50, 50))

        self.empty_heart_surf = pygame.image.load(BASE_DIR /
```

```
«graphics» / «heart» / «no_hearts_hud.png»).convert_alpha()
        self.empty_heart_surf =
pygame.transform.scale(self.empty_heart_surf, (50, 50))

    def draw(self, player: Player):
        # Відображає здоров'я гравця у вигляді серцевин
        for i in range(player.max_health):
            if (i + 1) <= player.health:
                # Якщо серцевина повна, відображається повне
зображення
                full_heart_rect = pygame.Rect((10 * i) +
(self.full_heart_surf.get_width() * i), 10,

self.full_heart_surf.get_width(),
self.full_heart_surf.get_height())
                self.display_surface.blit(self.full_heart_surf,
full_heart_rect)
            else:
                # Якщо серцевина порожня, відображається порожнє
зображення
                empty_hearth_rect = pygame.Rect((10 * i) +
(self.empty_heart_surf.get_width() * i), 10,

self.empty_heart_surf.get_width(),
self.empty_heart_surf.get_height())
                self.display_surface.blit(self.empty_heart_surf,
empty_hearth_rect)
                self.display_surface.blit(self.empty_heart_surf,
empty_hearth_rect)


transition.py:
# Імпортуємо бібліотеку Pygame
import pygame

# Імпортуємо константи з модуля settings
from src.settings import SCREEN_WIDTH, SCREEN_HEIGHT, TARGET_FPS

# Клас Transition представляє анімацію переходу між рівнями
class Transition:
    def __init__(self, reset_player, stop_respawn):
        # Ініціалізація
        self.display_surface = pygame.display.get_surface()
        self.reset = reset_player
        self.stop = stop_respawn

        # Зображення для накладання
        self.image = pygame.Surface((SCREEN_WIDTH,
SCREEN_HEIGHT))
        self.color = 255
        self.speed = -2

    def play(self, dt: float):
```

```
        # Відтворює анімацію переходу
        self.color += self.speed * dt * TARGET_FPS
        if self.color <= 0:
            self.speed *= -1
            self.color = 0
            self.reset()
        if self.color > 255:
            self.color = 255
            self.speed = -2
            self.stop()

        self.image.fill((self.color, self.color, self.color))
        self.display_surface.blit(self.image, (0, 0),
special_flags=pygame.BLEND_RGBA_MULT)


timer.py:
import pygame.time

class Timer:
    def __init__(self, duration: int, func=None):
        # Тривалість таймера в мілісекундах
        self.duration = duration

        # Функція, яка буде викликана після закінчення таймера
        self.func = func

        # Час початку таймера
        self.start_time = 0

        # Прапорець, який вказує, чи таймер активний
        self.active = False

    def activate(self):
        # Активує таймер та встановлює час початку
        self.active = True
        self.start_time = pygame.time.get_ticks()

    def deactivate(self):
        # Деактивує таймер та скидає час початку
        self.active = False
        self.start_time = 0

    def update(self):
        # Оновлює стан таймера
        current_time = pygame.time.get_ticks()
        if current_time - self.start_time >= self.duration:
            # Якщо таймер закінчився, викликає функцію (якщо
вона задана)
            if self.func and self.start_time != 0:
                self.func()
            # Деактивує таймер
            self.deactivate()
```

tile.py:

```
import pygame
from src.settings import LAYERS, TILE_SIZE

# Клас Tile представляє базовий тайл
class Tile(pygame.sprite.Sprite):
    def __init__(self, pos: tuple[int, int],
                 groups: list[pygame.sprite.AbstractGroup],
                 surf: pygame.Surface =
pygame.Surface((TILE_SIZE, TILE_SIZE)),
                 z: int = LAYERS['main'],
                 alpha: int = 255):
        # Ініціалізація спрайту
        super().__init__(groups)
        self.image = surf
        self.image.set_alpha(alpha)
        self.rect = self.image.get_rect(topleft=pos)
        self.z = z

    def draw_debug(self, display_surface: pygame.Surface,
offset: pygame.math.Vector2):
        # Відображає прямокутник навколо тайлу для налагодження
        offset_rect = self.rect.copy()
        offset_rect.center -= offset
        pygame.draw.rect(display_surface, 'white', offset_rect,
3)

# Клас AnimatedTile представляє анімований тайл
class AnimatedTile(Tile):
    def __init__(self, pos: tuple[int, int], frames:
list[pygame.Surface],
                 groups: list[pygame.sprite.AbstractGroup],
speed_animation: int = 5, z: int = LAYERS['main'],
                 alpha: int = 255):
        # Анімація
        self.frames = frames
        self.frame_index = 0
        self.speed_animation = speed_animation

        # Ініціалізація тайлу
        super().__init__(
            pos=pos,
            surf=self.frames[self.frame_index],
            groups=groups,
            z=z
        )
        self.alpha = alpha

    def animate(self, dt: float):
        # Оновлює анімацію тайлу
        self.frame_index += self.speed_animation * dt
```

```
        if self.frame_index >= len(self.frames):
            self.frame_index = 0
        self.image = self.frames[int(self.frame_index)]
        self.image.set_alpha(self.alpha)

    def update(self, dt: float):
        self.animate(dt)

# Клас ExitTile представляє тайл, який є виходом на інший рівень
class ExitTile(Tile):
    def __init__(self, current_level: str, new_level: str,
                 pos: tuple[int, int],
                 width: int, height: int,
                 groups: list[pygame.sprite.AbstractGroup],
                 ):
        # Ініціалізація тайлу
        super().__init__(
            pos=pos,
            groups=groups)
        self.image = pygame.Surface((width, height))
        self.rect = self.image.get_rect(topleft=pos)

        self.current_level = current_level
        self.new_level = new_level

# Клас Checkpoint представляє контрольну точку на рівні
class Checkpoint(Tile):
    def __init__(self, level_name: str,
                 pos: tuple[int, int],
                 groups: list[pygame.sprite.AbstractGroup],
                 ):
        # Ініціалізація тайлу
        super().__init__(
            pos=pos,
            groups=groups)
        self.level_name = level_name


support.py:
from math import sin
from os import walk
from pathlib import Path

import pygame


def import_folder(path: Path) -> list[pygame.Surface]:
    ««««
    Імпортує всі зображення з вказаної директорії та повертає
список поверхонь pygame.

    Аргументи:
    path (Path): Шлях до директорії з зображеннями.
```

```
    Повертає:
    list[pygame.Surface]: Список поверхонь pygame, що містять
завантажені зображення.
    """
    surface_list = []

    for _, __, img_files in walk(path):
        for image in sorted(img_files):
            full_path = path / image
            image_surf =
pygame.image.load(full_path).convert_alpha()
            surface_list.append(image_surf)

    return surface_list


def wave_value():
    """
    Генерує значення прозорості для ефекту хвилі.

    Повертає:
    int: Значення прозорості (255 для непрозорого, 0 для
прозорого).
    """
    value = sin(pygame.time.get_ticks())
    if value >= 0:
        return 255
    else:
        return 0
```

settings.py:

```
from pathlib import Path

# Налаштування частоти кадрів
FPS = 10000  # Максимальна частота кадрів
TARGET_FPS = 60  # Цільова частота кадрів

# Режим відлагодження
DEBUG = False

# Налаштування екрану
TILE_SIZE = 64  # Розмір плитки (в пікселях)
SCREEN_WIDTH = 1280  # Ширина екрану
SCREEN_HEIGHT = 720  # Висота екрану
FULLSCREEN = False  # Повноекранний режим (True/False)

# Шлях до базової директорії
BASE_DIR = Path().resolve()

# Кольори
BG_COLOR = '#060C17'  # Колір фону
PLAYER_COLOR = '#C4F7FF'  # Колір гравця
```

```
TILE_COLOR = '#94D7F2'  # Колір плитки

# Налаштування камери
CAMERA_BORDERS = {
    'left': SCREEN_WIDTH // 2 - 50,  # Ліва межа камери
    'right': SCREEN_WIDTH // 2 + 50,  # Права межа камери
    'top': SCREEN_HEIGHT // 2 - 50,  # Верхня межа камери
    'bottom': SCREEN_HEIGHT // 2 + 50  # Нижня межа камери
}

# Шари
LAYERS = {
    'invisible': 0,  # Невидимий шар
    'water background': 1,  # Шар фону води
    'terrain': 2,  # Шар ландшафту
    'main': 3,  # Головний шар
    'water': 4  # Шар води
}
```

player.py:

```
import pygame
from src.settings import TARGET_FPS, BASE_DIR, LAYERS
from src.support import import_folder, wave_value
from src.tile import Tile
from src.timer import Timer


class Player(pygame.sprite.Sprite):
    def __init__(self, pos: tuple[int, int], group:
pygame.sprite.Group,
                 collision_sprites: pygame.sprite.Group,
create_attack, destroy_attack, create_particules,
                 joysticks: list[pygame.joystick.Joystick]):
        super().__init__(group)

        # Анімація
        self.animations = {
            'right_idle': [], 'left_idle': [],
            'right_run': [], 'left_run': [],
            'right_jump': [], 'left_jump': [],
            'right_fall': [], 'left_fall': [],
            'right_attack': [], 'left_attack': []
        }
        self.import_assets()
        self.status = 'right_idle'
        self.speed_animation = 4
        self.frame_index = 0
        self.alpha = 255

        # Налаштування
        self.image =
self.animations[self.status][self.frame_index]
        self.rect = self.image.get_rect(topleft=pos)
```

```
        self.z = LAYERS['main']
        self.joysticks = joysticks

        # Рух
        self.direction = pygame.math.Vector2()
        self.pos = pygame.math.Vector2(self.rect.topleft)
        self.speed = 8 * TARGET_FPS
        self.gravity = 0.8 * TARGET_FPS
        self.jump_speed = 20
        self.collision_sprites = collision_sprites
        self.on_floor = False
        self.can_double_jump = False
        self.can_dash = True
        self.can_attack = True

        # Статистика
        self.max_health = 3
        self.health = self.max_health
        self.damage = 10

        # Взаємодія
        self.vulnerable = True
        self.create_attack = create_attack
        self.destroy_attack = destroy_attack

        # Таймер
        self.timers = {
            'double jump': Timer(500,
self.activate_double_jump),
            'dash': Timer(100, self.stop_dash),
            'reset dash': Timer(500, self.reset_dash),
            'invulnerability': Timer(500,
self.reset_vulnerability),
            'attacking': Timer(300, self.stop_attack),
            'reset attack': Timer(500, self.reset_attack)
        }

        # Частинки гравця
        self.create_particules = create_particules

        # Імпорт звуків
        self.jump_sound = pygame.mixer.Sound(BASE_DIR / «sounds»
/ «jump.wav»)
        self.attack_sound = pygame.mixer.Sound(BASE_DIR /
«sounds» / «hit.wav»)
        self.hit_sound = pygame.mixer.Sound(BASE_DIR / «sounds»
/ «hit.wav»)

    def import_assets(self):
        # Імпортувати активи анімації для кожного типу анімації
        for animation in self.animations.keys():
            full_path = BASE_DIR / «graphics» / «player» /
animation
```

```
            self.animations[animation] =
import_folder(full_path)

    def animate(self, dt: float):
        # Оновити індекс кадру анімації
        self.frame_index += self.speed_animation * dt
        if self.frame_index >=
len(self.animations[self.status]):
            self.frame_index = 0
        self.image =
self.animations[self.status][int(self.frame_index)]

        # Ефект прозорості при отриманні шкоди
        if not self.vulnerable and not
self.timers['dash'].active:
            alpha = wave_value()
            self.image.set_alpha(alpha)
        else:
            self.image.set_alpha(self.alpha)

    def change_joysticks(self, joysticks:
list[pygame.joystick.Joystick]):
        # Змінити список підключених джойстиків
        self.joysticks = joysticks

    def input(self):
        # Обробка вводу з клавіатури або джойстика
        if not self.timers['dash'].active and not
self.timers['attacking'].active:
            if len(self.joysticks) == 0:
                keys = pygame.key.get_pressed()

                # Рух
                if keys[pygame.K_RIGHT]:
                    self.direction.x = 1
                    self.status = 'right'
                elif keys[pygame.K_LEFT]:
                    self.direction.x = -1
                    self.status = 'left'
                else:
                    self.direction.x = 0

                # Атака
                if keys[pygame.K_q] and self.can_attack:
                    self.timers['attacking'].activate()
                    self.speed_animation = 4 * 3
                    self.frame_index = 0
                    self.can_attack = False
                    self.direction = pygame.math.Vector2()
                    self.attack_sound.play()

                    if keys[pygame.K_UP]:
                        self.create_attack('top')
```

```
                    elif keys[pygame.K_DOWN]:
                        self.create_attack('bottom')
                    else:

self.create_attack(self.status.split('_')[0])

                # Стрибок
                if keys[pygame.K_SPACE] and (self.on_floor or
self.can_double_jump):
                    if self.on_floor:
                        self.timers['double jump'].activate()
                    self.can_double_jump = False
                    self.direction.y = -self.jump_speed
                    self.frame_index = 0
                    self.create_particules('before_jump',
self.rect.topleft)
                    self.jump_sound.play()

                # Рівінь
                if keys[pygame.K_RCTRL] and self.can_dash:
                    self.direction.y = 0
                    self.can_dash = False
                    self.timers['dash'].activate()
                    self.vulnerable = False
                    self.speed *= 4

            else:
                axis = self.joysticks[0].get_axis(0)
                if abs(axis) < 0.3:
                    axis = 0

                # Рух
                if axis > 0:
                    self.direction.x = axis
                    self.status = 'right'
                elif axis < 0:
                    self.direction.x = axis
                    self.status = 'left'
                else:
                    self.direction.x = 0

                # Атака
                if self.joysticks[0].get_button(2) and
self.can_attack:
                    self.timers['attacking'].activate()
                    self.speed_animation = 4 * 3
                    self.frame_index = 0
                    self.can_attack = False
                    self.direction = pygame.math.Vector2()
                    self.attack_sound.play()

                    if axis != 0:
```

```
self.create_attack(self.status.split('_')[0])
                    else:
                        axis = self.joysticks[0].get_axis(1)
                        if abs(axis) < 0.3:
                            axis = 0

                        if axis < 0:
                            self.create_attack('top')
                        elif axis > 0:
                            self.create_attack('bottom')

                # Стрибок
                if self.joysticks[0].get_button(0) and
(self.on_floor or self.can_double_jump):
                    if self.on_floor:
                        self.timers['double jump'].activate()
                    self.can_double_jump = False
                    self.direction.y = -self.jump_speed
                    self.frame_index = 0
                    self.create_particules('before_jump',
self.rect.topleft)
                    self.jump_sound.play()

                # Рівінь
                if self.joysticks[0].get_button(5) and
self.can_dash:
                    self.direction.y = 0
                    self.can_dash = False
                    self.timers['dash'].activate()
                    self.vulnerable = False
                    self.speed *= 4

    def get_status(self):
        # Визначити поточний стан анімації на основі руху та дій
        if self.direction.magnitude() == 0:
            self.status = self.status.split('_')[0] + '_idle'

        if self.direction.y < 0:
            self.status = self.status.split('_')[0] + '_jump'
        elif self.direction.y > 1:
            self.status = self.status.split('_')[0] + '_fall'
        else:
            if self.direction.x != 0:
                self.status = self.status.split('_')[0] + '_run'

        if self.timers['attacking'].active:
            self.status = self.status.split('_')[0] + '_attack'

    def horizontal_collisions(self):
        # Обробка горизонтальних зіткнень з іншими спрайтами
        for sprite in self.collision_sprites.sprites():
            if sprite.rect.colliderect(self.rect):
                if self.direction.x < 0:
```

```
                    self.rect.left = sprite.rect.right
                if self.direction.x > 0:
                    self.rect.right = sprite.rect.left
                self.pos.x = self.rect.x

    def vertical_collisions(self):
        # Обробка вертикальних зіткнень з іншими спрайтами
        for sprite in self.collision_sprites.sprites():  # type:
Tile
            if sprite.rect.colliderect(self.rect):
                if self.direction.y > 0:
                    self.rect.bottom = sprite.rect.top
                    self.direction.y = 0
                    self.on_floor = True
                    self.can_double_jump = False
                    if 'fall' in self.status:
                        self.create_particules('after_jump',
self.rect.topleft)
                if self.direction.y < 0:
                    self.rect.top = sprite.rect.bottom
                    self.direction.y = 0
                self.pos.y = self.rect.y

        if self.on_floor and self.direction.y != 0:
            self.on_floor = False

    def apply_gravity(self, dt: float):
        # Застосувати гравітацію до вертикального руху
        self.direction.y += self.gravity * dt
        self.pos.y += self.direction.y * dt * TARGET_FPS
        self.rect.y = round(self.pos.y)

    def move(self, dt: float):
        # Рух гравця
        self.pos.x += self.direction.x * self.speed * dt
        self.rect.x = round(self.pos.x)
        self.horizontal_collisions()

        self.apply_gravity(dt)
        self.vertical_collisions()

    def draw_debug(self, display_surface: pygame.Surface,
offset: pygame.math.Vector2):
        # Намалювати прямокутник для відлагодження
        offset_rect = self.rect.copy()
        offset_rect.topleft -= offset
        pygame.draw.rect(display_surface, 'white', offset_rect,
3)

    def activate_double_jump(self):
        # Активувати можливість подвійного стрибка
        self.can_double_jump = True
```

```
    def stop_dash(self):
        # Зупинити ефект рівня
        self.speed = 8 * TARGET_FPS
        self.timers['reset dash'].activate()
        self.vulnerable = True

    def reset_dash(self):
        # Скинути можливість рівня
        self.can_dash = True

    def reset_vulnerability(self):
        # Скинути вразливість гравця
        self.vulnerable = True

    def stop_attack(self):
        # Зупинити атаку
        self.timers['reset attack'].activate()
        self.speed_animation = 4
        self.destroy_attack()

    def reset_attack(self):
        # Скинути можливість атаки
        self.can_attack = True

    def update_timers(self):
        # Оновити всі таймери
        for timer in self.timers.values():  # type: Timer
            timer.update()

    def update(self, dt: float):
        # Головний метод оновлення стану гравця
        self.input()
        self.get_status()
        self.update_timers()

        self.move(dt)
        self.animate(dt)


particule.py:
import pygame

from src.settings import TARGET_FPS, BASE_DIR, LAYERS
from src.support import import_folder


class ParticuleManager:
    def __init__(self):
        # Словник, що містить списки зображень для різних типів
анімацій часток
        self.frames = {
            'before_jump': import_folder(BASE_DIR / «graphics» /
«particules» / «before_jump»),
```

```
            'after_jump': import_folder(BASE_DIR / «graphics» /
«particules» / «after_jump»),
            'mushroom_death': import_folder(BASE_DIR /
«graphics» / «enemies» / «mushroom» / «death»),
            'player_death': import_folder(BASE_DIR / «graphics»
/ «player» / «death»)
        }

    def create_particules(self, animation_type: str, pos:
tuple[int, int], group: pygame.sprite.Group):
        # Отримати список зображень для заданого типу анімації
        animation_frames = self.frames[animation_type]
        # Створити екземпляр ParticuleEffect з заданими
параметрами
        ParticuleEffect(pos, animation_frames, group)


class ParticuleEffect(pygame.sprite.Sprite):
    def __init__(self, pos: tuple[int, int], animation_frames:
list[pygame.Surface], group: pygame.sprite.Group):
        super().__init__(group)

        # Анімація
        self.frames = animation_frames
        self.frame_index = 0
        self.animation_speed = 0.15 * TARGET_FPS

        # Налаштування
        self.image = self.frames[self.frame_index]
        self.rect = self.image.get_rect(topleft=pos)
        self.z = LAYERS['main']

    def animate(self, dt: float):
        # Оновити індекс кадру анімації
        self.frame_index += self.animation_speed * dt
        if self.frame_index >= len(self.frames):
            # Якщо анімація завершена, видалити частку
            self.kill()
        else:
            # Інакше, оновити зображення частки
            self.image = self.frames[int(self.frame_index)]

    def update(self, dt: float):
        # Оновити анімацію частки
        self.animate(dt)
```

level_data.py:

```
LEVELS = {
    'map_test': {
        'map_test_2': (3008, 384)
    },
    'map_test_2': {
        'map_test': (704, 832),
```

```
        'map_test_3': (2304, 512),
        'exit': (128, 576)
    },
    'map_test_3': {
        'map_test_2': (2304, 512),
        'exit': (128, 576)
    }
}


level.py:
import sys
import pygame
from pytmx import load_pygame

from src.UI import UI
from src.camera import CameraGroup
from src.enemy import Enemy
from src.level_data import LEVELS
from src.particule import ParticuleManager
from src.player import Player
from src.settings import TILE_SIZE, BG_COLOR, BASE_DIR, DEBUG,
LAYERS
from src.support import import_folder
from src.tile import Tile, AnimatedTile, ExitTile, Checkpoint
from src.timer import Timer
from src.transition import Transition
from src.weapon import Weapon

class Level:
    def __init__(self, level_name, joysticks:
list[pygame.joystick.Joystick]):
        self.display_surface = pygame.display.get_surface()
        self.joysticks = joysticks
        self.level_name = level_name
        self.exits = []
        self.all_sprites = CameraGroup()
        self.collision_sprites = pygame.sprite.Group()
        self.collider_sprites = pygame.sprite.Group()
        self.enemy_sprites = pygame.sprite.Group()
        self.checkpoint_sprites = pygame.sprite.Group()
        self.exit_sprites = pygame.sprite.Group()
        self.last_checkpoint = None
        self.current_attack = None

        self.timers = {
            'screen shake': Timer(300, self.stop_screen_shake),
            'player death': Timer(1500, self.death_animation)
        }
        self.screen_shake = False
        self.player = Player((128, 576), self.all_sprites,
self.collision_sprites, self.create_attack,
                             self.destroy_attack,
```

```
self.create_particules, self.joysticks)
        self.respawn = False
        self.current_level = level_name
        self.load_map(self.current_level)
        self.ui = UI()
        self.particule_manager = ParticuleManager()
        self.transition = Transition(self.reset_player,
self.stop_respawn)

    def load_map(self, level_name, x: int = None, y: int =
None):
        self.clear_map()
        tmx_data = load_pygame(f'data/{level_name}.tmx')
        self.current_level = level_name

        if x is not None and y is not None:
            self.player.pos.x = x
            self.player.pos.y = y

        for x, y, surf in
tmx_data.get_layer_by_name('Terrain').tiles():
            Tile((x * TILE_SIZE, y * TILE_SIZE),
[self.all_sprites, self.collision_sprites], surf,
LAYERS['terrain'])

        water_frames = import_folder(BASE_DIR / «graphics» /
«water»)
        for x, y, surf in
tmx_data.get_layer_by_name('Water').tiles():
            AnimatedTile((x * TILE_SIZE, y * TILE_SIZE),
water_frames, [self.all_sprites], z=LAYERS['water'], alpha=150)

        for obj in tmx_data.get_layer_by_name('Enemies'):
            if obj.name == 'Collider':
                Tile((obj.x, obj.y), [self.collider_sprites],
z=LAYERS['invisible'])
            if obj.type == 'Enemy':
                Enemy(obj.name, (obj.x, obj.y),
[self.all_sprites, self.enemy_sprites], self.collider_sprites,
                      self.create_particules)

        for obj in tmx_data.get_layer_by_name('Interaction'):
            if obj.name == 'Checkpoint':
                Checkpoint(self.current_level, (obj.x, obj.y),
[self.checkpoint_sprites])

        for obj in tmx_data.get_layer_by_name('Exit'):
            ExitTile(self.current_level, obj.name, (obj.x,
obj.y), obj.width, obj.height,
                     [self.all_sprites, self.exit_sprites])

    def change_joysticks(self, joysticks:
list[pygame.joystick.Joystick]):
```

```
        self.joysticks = joysticks
        self.player.change_joysticks(self.joysticks)

    def clear_map(self):
        self.all_sprites.empty()
        self.collision_sprites.empty()
        self.collider_sprites.empty()
        self.enemy_sprites.empty()
        self.checkpoint_sprites.empty()
        self.exit_sprites.empty()
        if self.current_attack:
            self.current_attack.kill()

    def create_attack(self, direction: str):
        self.current_attack = Weapon(self.player, direction,
[self.all_sprites])

    def destroy_attack(self):
        if self.current_attack:
            self.current_attack.kill()
        self.current_attack = None

    def player_attack_logic(self):
        if self.current_attack:
            collision_sprites =
pygame.sprite.spritecollide(self.current_attack,
self.enemy_sprites, False)
            if collision_sprites:
                for target_sprite in collision_sprites:
                    target_sprite.get_damage(self.player)

    def damage_player(self):
        if self.player.vulnerable:
            collision_sprites =
pygame.sprite.spritecollide(self.player, self.enemy_sprites,
False)
            if collision_sprites:
                for _ in collision_sprites:
                    self.player.health -= 1
                    self.player.vulnerable = False

self.player.timers['invulnerability'].activate()
                    self.screen_shake = True
                    self.timers['screen shake'].activate()

                    if self.player.health <= 0:

self.particule_manager.create_particules('player_death',
self.player.rect.topleft,

self.all_sprites)
                        self.player.kill()
                        self.timers['player death'].activate()
```

```
    def reset_player(self):
        self.load_map(self.last_checkpoint.level_name)
        x = self.last_checkpoint.rect.x
        y = self.last_checkpoint.rect.y
        self.player = Player((x, y), self.all_sprites,
self.collision_sprites, self.create_attack,
                             self.destroy_attack,
self.create_particules, self.joysticks)

    def stop_respawn(self):
        self.respawn = False

    def death_animation(self):
        self.respawn = True

    def checkpoint_collision(self):
        collision_sprites =
pygame.sprite.spritecollide(self.player,
self.checkpoint_sprites, False)
        if collision_sprites:
            for checkpoint in collision_sprites:
                self.last_checkpoint = checkpoint

    def exit_collision(self):
        collision_sprites =
pygame.sprite.spritecollide(self.player, self.exit_sprites,
False)
        if collision_sprites:
            for exit_sprite in collision_sprites:
                if exit_sprite.new_level == 'exit':
                    self.load_map('map_test', 128, 576)
                else:
                    x, y =
LEVELS[exit_sprite.new_level][exit_sprite.current_level]
                    self.load_map(exit_sprite.new_level, x, y)

    def create_particules(self, animation_type: str, pos:
tuple[int, int]):
        self.particule_manager.create_particules(animation_type,
pos, self.all_sprites)

    def stop_screen_shake(self):
        self.screen_shake = False

    def update_timers(self):
        for timer in self.timers.values():
            timer.update()

    def run(self, dt: float):
        self.damage_player()
        self.player_attack_logic()
        self.checkpoint_collision()
```

```
        self.exit_collision()

        self.all_sprites.update(dt)
        self.display_surface.fill(BG_COLOR)
        self.all_sprites.custom_draw(self.player,
self.screen_shake)
        self.ui.draw(self.player)

        self.update_timers()

        if self.respawn:
            self.transition.play()
```

game.py:

```
import sys
import pygame

# Імпортуємо клас Level
from src.level import Level
# Імпортуємо налаштування екрану та FPS
from src.settings import SCREEN_WIDTH, SCREEN_HEIGHT, FPS,
FULLSCREEN

# Клас Game представляє головний цикл гри
class Game:
    def __init__(self):
        # Ініціалізація Pygame
        pygame.init()
        # Створюємо вікно гри
        self.screen = pygame.display.set_mode((SCREEN_WIDTH,
SCREEN_HEIGHT))
        # Якщо FULLSCREEN встановлено, переходимо в
повноекранний режим
        if FULLSCREEN:
            self.screen =
pygame.display.set_mode((self.screen.get_width(),
self.screen.get_height()),

pygame.FULLSCREEN)
        # Встановлюємо заголовок вікна
        pygame.display.set_caption('')
        # Створюємо об'єкт для відстеження FPS
        self.clock = pygame.time.Clock()

        # Ініціалізація джойстиків
        pygame.joystick.init()
        # Створюємо список підключених джойстиків
        self.joysticks = [pygame.joystick.Joystick(i) for i in
range(pygame.joystick.get_count())]

        # Створюємо об'єкт рівня гри
        self.level = Level('map_test', self.joysticks)
```

```
    def run(self):
        while True:
            # Обчислюємо дельту часу (dt) для плавного руху
            dt = self.clock.tick(FPS) / 1000

            # Обробляємо події
            for event in pygame.event.get():
                # Якщо подія - вихід з гри
                if event.type == pygame.QUIT:
                    # Вимикаємо джойстики та виходимо з гри
                    pygame.joystick.quit()
                    pygame.quit()
                    sys.exit()

                # Якщо підключено новий джойстик
                if event.type == pygame.JOYDEVICEADDED:
                    # Оновлюємо список джойстиків
                    self.joysticks =
[pygame.joystick.Joystick(i) for i in
range(pygame.joystick.get_count())]
                    # Повідомляємо рівень про зміну джойстиків
                    self.level.change_joysticks(self.joysticks)
                # Якщо від'єднано джойстик
                if event.type == pygame.JOYDEVICEREMOVED:
                    # Оновлюємо список джойстиків
                    self.joysticks =
[pygame.joystick.Joystick(i) for i in
range(pygame.joystick.get_count())]
                    # Повідомляємо рівень про зміну джойстиків
                    self.level.change_joysticks(self.joysticks)

            # Оновлюємо рівень гри
            self.level.run(dt)
            # Оновлюємо вікно гри
            pygame.display.update()
```

enemy.py:

```
from random import randint
import pygame

# Імпортуємо класи та функції з інших файлів
from src.player import Player
from src.settings import BASE_DIR, LAYERS, TARGET_FPS
from src.support import import_folder, wave_value
from src.timer import Timer

# Клас Enemy представляє ворога в грі
class Enemy(pygame.sprite.Sprite):
    def __init__(self, monster_name: str, pos: tuple[int, int],
groups: list[pygame.sprite.Group],
                 collider_sprites: pygame.sprite.Group,
create_particules):
        super().__init__(groups)
```

```
        self.monster_name = monster_name

        # Анімація
        self.animations = {
            'right_run': [], 'left_run': []
        }
        self.import_assets(monster_name)
        self.status = 'right_run'
        self.speed_animation = 6
        self.frame_index = 0

        # Налаштування
        self.image =
self.animations[self.status][self.frame_index]
        self.rect = self.image.get_rect(topleft=pos)
        self.z = LAYERS['main']

        # Рух
        self.pos = pygame.math.Vector2(self.rect.topleft)
        self.speed = randint(3, 5) * TARGET_FPS

        # Таймери
        self.timers = {
            'invulnerability': Timer(350,
self.reset_vulnerability)
        }

        # Статистика
        self.health = 50
        self.vulnerable = True

        # Зіткнення
        self.collider_sprites = collider_sprites

        # Частинки гравця
        self.create_particules = create_particules

    # Імпортуємо активи (зображення) для ворога
    def import_assets(self, name: str):
        path = BASE_DIR / «graphics» / «enemies» / name

        for animation in self.animations.keys():
            self.animations[animation] = import_folder(path /
animation)

    # Анімуємо ворога
    def animate(self, dt: float):
        self.frame_index += self.speed_animation * dt
        if self.frame_index >=
len(self.animations[self.status]):
            self.frame_index = 0
        self.image =
self.animations[self.status][int(self.frame_index)]
```

```
        # Удар
        if not self.vulnerable:
            alpha = wave_value()
            self.image.set_alpha(alpha)
        else:
            self.image.set_alpha(255)

    # Визначаємо статус ворога (рух праворуч чи ліворуч)
    def get_status(self):
        if self.speed > 0:
            self.status = 'right_run'
        else:
            self.status = 'left_run'

    # Перевіряємо зіткнення з іншими спрайтами
    def collision(self):
        if pygame.sprite.spritecollide(self,
self.collider_sprites, False):
            self.speed *= -1

    # Рухаємо ворога
    def move(self, dt: float):
        if self.vulnerable:
            self.pos.x += self.speed * dt
            self.rect.x = round(self.pos.x)
            self.collision()

    # Наносимо шкоду ворогу від гравця
    def get_damage(self, player: Player):
        if self.vulnerable:
            self.health -= player.damage
            self.vulnerable = False
            self.timers['invulnerability'].activate()

    # Перевіряємо, чи ворог мертвий
    def check_death(self):
        if self.health <= 0:
            self.create_particules(f'{self.monster_name}_death',
self.rect.topleft)
            self.kill()

    # Відображаємо прямокутник навколо ворога (для налагодження)
    def draw_debug(self, display_surface: pygame.Surface,
offset: pygame.math.Vector2):
        # Малюємо прямокутник
        offset_rect = self.rect.copy()
        offset_rect.topleft -= offset
        pygame.draw.rect(display_surface, 'white', offset_rect,
3)

    # Скидаємо вразливість ворога
    def reset_vulnerability(self):
```

```
            self.vulnerable = True

        # Оновлюємо таймери
        def update_timers(self):
            for timer in self.timers.values():  # type: Timer
                timer.update()

        # Оновлюємо стан ворога
        def update(self, dt: float):
            self.get_status()
            self.update_timers()
            self.check_death()

            self.move(dt)
            self.animate(dt)
```

camera.py:

```
from random import randint
import pygame

from src.player import Player
from src.settings import CAMERA_BORDERS, LAYERS
from src.tile import Tile


class CameraGroup(pygame.sprite.Group):
    def __init__(self):
        ««««
        Ініціалізація класу CameraGroup, який є підкласом
pygame.sprite.Group.
        Цей клас використовується для керування камерою та
відображення спрайтів на екрані.
        ««««
        super().__init__()
        self.display_surface = pygame.display.get_surface()
        self.offset = pygame.math.Vector2(100, 300)

        # Встановлення меж камери
        cam_left = CAMERA_BORDERS['left']
        cam_top = CAMERA_BORDERS['top']
        cam_width = self.display_surface.get_size()[0] -
(cam_left + CAMERA_BORDERS['right'])
        cam_height = self.display_surface.get_size()[1] -
(cam_top + CAMERA_BORDERS['bottom'])

        self.camera_rect = pygame.Rect(cam_left, cam_top,
cam_width, cam_height)

    def custom_draw(self, player: Player, screen_shake: bool):
        ««««
        Функція для відображення спрайтів на екрані з
урахуванням положення камери та ефекту тремтіння екрану.
```

```
        Args:
            player (Player): Об'єкт класу Player, який
представляє гравця.
            screen_shake (bool): Булеве значення, яке вказує, чи
потрібно застосувати ефект тремтіння екрану.
        """
        # Отримання положення камери
        if player.rect.left < self.camera_rect.left:
            self.camera_rect.left = player.rect.left
        if player.rect.right > self.camera_rect.right:
            self.camera_rect.right = player.rect.right
        if player.rect.top < self.camera_rect.top:
            self.camera_rect.top = player.rect.top
        if player.rect.bottom > self.camera_rect.bottom:
            self.camera_rect.bottom = player.rect.bottom

        # Зміщення камери
        self.offset = pygame.math.Vector2(
            self.camera_rect.left - CAMERA_BORDERS['left'],
            self.camera_rect.top - CAMERA_BORDERS['top']
        )

        # Ефект тремтіння екрану
        offset_screen_shake = pygame.math.Vector2()
        if screen_shake:
            offset_screen_shake.x = randint(-4, 4)
            offset_screen_shake.y = randint(-4, 4)

        # Відображення спрайтів
        for layer in LAYERS.values():
            for sprite in self.sprites():  # type: Tile
                if sprite.z == layer:
                    offset_pos = sprite.rect.topleft -
self.offset + offset_screen_shake
                    self.display_surface.blit(sprite.image,
offset_pos)

    def draw_debug(self):
        """
        Функція для відображення відладчика (debug) для
спрайтів, які мають метод draw_debug.
        """
        # Зміщення камери
        self.offset = pygame.math.Vector2(
            self.camera_rect.left - CAMERA_BORDERS['left'],
            self.camera_rect.top - CAMERA_BORDERS['top']
        )

        for sprite in self.sprites():
            if hasattr(sprite, 'draw_debug'):
                sprite.draw_debug(self.display_surface,
self.offset)
```

```
    def empty(self):
        ««««
        Функція для видалення всіх спрайтів з групи, які не
мають атрибуту 'input'.
        ««««
        for sprite in self.sprites():
            if not hasattr(sprite, 'input'):
                sprite.kill()


map_test.tmx:
<?xml version=«1.0» encoding=«UTF-8»?>
<map version=«1.10» tiledversion=«1.10.2»
orientation=«orthogonal» renderorder=«right-down» width=«50»
height=«17» tilewidth=«64» tileheight=«64» infinite=«0»
nextlayerid=«6» nextobjectid=«54»>
 <tileset firstgid=«1» source=«tilesets/terrain.tsx»/>
 <tileset firstgid=«73» source=«tilesets/Water.tsx»/>
 <tileset firstgid=«77» source=«tilesets/Enemies.tsx»/>
 <layer id=«1» name=«Terrain» width=«50» height=«17»>
  <data encoding=«csv»>
14,14,14,14,14,14,14,14,14,14,14,14,14,14,14,14,14,14,14,14,14,14,14,1
4,14,14,14,14,14,14,14,14,14,14,14,14,14,14,14,14,14,14,14,14,14,14,14
,14,14,14,14,14,14,14,
4,26,26,26,26,26,26,26,26,26,26,26,26,26,26,26,26,26,26,26,26,26,26,26
,26,26,26,26,26,26,6,14,14,14,14,14,14,14,14,14,14,14,14,14,14,14,14,1
4,14,14,14,14,14,14,
15,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,13,14,4
,26,26,26,26,26,26,26,26,26,26,26,26,26,26,26,26,26,26,26,26,26,6,
15,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,13,14,1
5,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,13,
15,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,25,26,2
7,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,25,
15,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
15,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
15,0,0,0,0,0,0,0,0,0,1,2,3,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,3
,0,0,0,0,0,0,0,0,0,0,0,0,0,1,2,2,2,2,2,
15,0,0,0,0,0,0,0,0,0,0,25,26,27,0,0,0,0,2,2,2,2,0,0,0,0,0,0,0,1,2,
14,15,0,0,0,0,0,0,0,0,0,0,0,0,25,26,6,14,14,14,
15,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,13,14,1
4,27,0,0,0,0,0,1,2,2,2,45,0,0,0,0,25,26,26,6,
28,2,2,3,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,13,14,1
5,0,0,0,0,0,0,13,14,4,27,0,0,0,0,0,0,0,0,0,13,
14,14,14,15,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,43,44,44,44,44
,30,14,15,0,0,0,0,0,0,0,25,26,27,0,0,0,0,0,0,0,0,0,0,13,
14,14,14,15,0,0,0,0,0,0,0,1,2,3,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,13,1
4,15,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,13,
14,14,14,28,2,2,2,2,2,2,2,30,14,15,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1
3,14,15,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,13,
14,14,14,14,14,14,14,14,14,14,14,14,14,28,2,2,2,73,2,2,2,2,2,2,2,2,
2,2,2,2,30,14,28,2,2,2,3,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,13,
```

```
14,14,14,14,14,14,14,14,14,14,14,14,14,14,14,14,73,14,14,14,14,1
4,14,14,14,14,14,14,14,14,14,14,14,14,28,2,2,2,2,2,2,2,2,2,2,2,2
,2,2,30,
14,14,14,14,14,14,14,14,14,14,14,14,14,14,14,14,73,14,14,14,14,1
4,14,14,14,14,14,14,14,14,14,14,14,14,14,14,14,14,14,14,14,14,14
,14,14,14,14,14,14,14
</data>
 </layer>
 <layer id=«3» name=«Water» width=«50» height=«17» locked=«1»>
  <data encoding=«csv»>
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,73,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,73,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,73,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,73,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,73,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,73,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,73,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,73,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,73,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,73,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,73,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,73,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,73,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,73,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
</data>
 </layer>
 <objectgroup id=«4» name=«Enemies» locked=«1»>
  <object id=«8» name=«Collider» x=«1408» y=«640» width=«64»
height=«64»/>
  <object id=«9» name=«Collider» x=«1792» y=«640» width=«64»
height=«64»/>
  <object id=«11» name=«mushroom» type=«Enemy» gid=«77» x=«1536»
y=«704» width=«64» height=«64»/>
```

```
  <object id=«13» name=«Collider» x=«1792» y=«832» width=«64»
height=«64»/>
  <object id=«14» name=«Collider» x=«1088» y=«832» width=«64»
height=«64»/>
  <object id=«22» name=«mushroom» type=«Enemy» gid=«77» x=«1212»
y=«898» width=«64» height=«64»/>
  <object id=«35» name=«mushroom» type=«Enemy» gid=«77» x=«2880»
y=«960» width=«64» height=«64»/>
  <object id=«36» name=«mushroom» type=«Enemy» gid=«77» x=«2624»
y=«960» width=«64» height=«64»/>
  <object id=«37» name=«Collider» x=«2176» y=«896» width=«64»
height=«64»/>
  <object id=«38» name=«Collider» x=«3136» y=«896» width=«64»
height=«64»/>
  <object id=«42» name=«mushroom» type=«Enemy» gid=«77» x=«1284»
y=«898» width=«64» height=«64»/>
  <object id=«43» name=«Collider» x=«1089» y=«448» width=«58»
height=«64»/>
  <object id=«44» name=«Collider» x=«1344» y=«451» width=«64»
height=«64»/>
  <object id=«45» name=«mushroom» type=«Enemy» gid=«77» x=«1113»
y=«510» width=«64» height=«64»/>
  <object id=«46» name=«mushroom» type=«Enemy» gid=«77» x=«448»
y=«831» width=«64» height=«64»/>
  <object id=«47» name=«Collider» x=«641» y=«771» width=«64»
height=«64»/>
  <object id=«48» name=«Collider» x=«188» y=«769» width=«64»
height=«64»/>
 </objectgroup>
 <objectgroup id=«2» name=«Interaction» locked=«1»>
  <object id=«34» name=«Checkpoint» x=«128» y=«576» width=«64»
height=«64»/>
 </objectgroup>
 <objectgroup id=«5» name=«Exit»>
  <object id=«39» name=«map_test_2» x=«3136» y=«320» width=«64»
height=«128»/>
 </objectgroup>
</map>


map_test_2.tmx:
<?xml version=«1.0» encoding=«UTF-8»?>
<map version=«1.10» tiledversion=«1.10.2»
orientation=«orthogonal» renderorder=«right-down» width=«50»
height=«17» tilewidth=«64» tileheight=«64» infinite=«0»
nextlayerid=«11» nextobjectid=«35»>
 <tileset firstgid=«1» source=«tilesets/terrain.tsx»/>
 <tileset firstgid=«73» source=«tilesets/Enemies.tsx»/>
 <layer id=«1» name=«Terrain» width=«50» height=«17»>
  <data encoding=«csv»>
0,0,0,0,0,0,0,0,14,14,14,14,14,14,14,14,14,14,14,14,14,14,14,14,14,
14,14,14,14,14,14,14,14,14,14,14,14,14,14,14,14,14,14,14,14,0,0,
0,0,0,0,
```

0,0,0,0,0,0,0,0,14,14,14,14,14,14,14,14,14,14,14,14,14,14,14,14,
14,14,26,26,26,26,26,26,26,26,26,26,26,26,26,26,26,14,14,14,0,0,
0,0,0,0,
0,0,0,0,0,0,0,0,14,14,14,14,14,14,14,14,14,14,14,14,14,14,14,14,
14,27,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,13,14,14,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,14,14,14,14,14,14,14,14,14,14,14,14,14,14,14,14,
27,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,13,14,14,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,14,14,14,14,14,14,14,14,14,14,14,14,14,14,26,27,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,13,14,14,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,14,14,14,14,14,14,14,14,14,14,14,14,14,27,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,13,14,14,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,14,14,14,14,14,14,14,14,4,26,26,26,27,0,0,0,0,0,
0,0,0,0,0,1,2,2,2,2,2,2,2,3,0,13,14,14,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,14,14,14,4,26,26,26,26,27,0,0,0,0,0,0,0,0,0,0,
0,0,1,14,14,14,14,26,26,26,26,27,0,13,14,14,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,14,14,14,15,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
25,26,26,26,27,0,0,0,0,0,0,13,14,14,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,14,14,14,15,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,13,14,14,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,14,14,14,15,0,0,0,0,0,0,0,0,0,0,0,0,0,1,2,2,2,2,
2,2,2,2,2,2,3,0,0,0,0,13,14,14,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,14,26,26,27,0,0,0,0,0,1,2,2,2,2,2,2,2,26,26,26,2
6,26,26,26,26,26,26,26,27,0,0,0,0,13,14,14,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,14,0,0,0,0,0,0,0,0,25,26,26,26,26,26,26,27,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,13,14,14,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,14,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,13,14,14,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,14,2,2,2,2,2,2,3,0,0,0,0,0,0,0,0,0,0,1,2,2,3,0,0
,0,0,1,2,2,2,2,2,2,30,14,14,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,14,14,14,14,14,14,14,28,2,2,2,2,2,2,2,2,2,2,30,1
4,14,28,2,2,2,2,30,14,14,14,14,14,14,14,14,14,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,14,14,14,14,14,14,14,14,14,14,14,14,14,14,14,14,
14,14,14,14,14,14,14,14,14,14,14,14,14,14,14,14,14,14,14,14,0,0,
0,0,0,0
</data>
</layer>
<layer id=«10» name=«Water» width=«50» height=«17»>
<data encoding=«csv»>
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

```
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
</data>
 </layer>
 <objectgroup id=«3» name=«Enemies»>
  <object id=«4» name=«mushroom» type=«Enemy» gid=«73» x=«2304»
y=«896» width=«64» height=«64»/>
  <object id=«5» name=«Collider» x=«2112» y=«832» width=«64»
height=«64»/>
  <object id=«6» name=«Collider» x=«2624» y=«832» width=«64»
height=«64»/>
  <object id=«7» name=«mushroom» type=«Enemy» gid=«73» x=«2500»
y=«900» width=«64» height=«64»/>
  <object id=«8» name=«mushroom» type=«Enemy» gid=«73»
x=«1773.33» y=«894» width=«64» height=«64»/>
  <object id=«22» name=«Collider» x=«1918.67» y=«833.333»
width=«64» height=«64»/>
  <object id=«23» name=«Collider» x=«1597.33» y=«832» width=«64»
height=«64»/>
  <object id=«24» name=«Collider» x=«2180» y=«897.333»
width=«64» height=«64»/>
  <object id=«25» name=«Collider» x=«1856» y=«900» width=«64»
height=«64»/>
  <object id=«26» name=«Collider» x=«1665.33» y=«896» width=«64»
height=«64»/>
  <object id=«27» name=«Collider» x=«957.333» y=«898.667»
width=«64» height=«64»/>
  <object id=«28» name=«mushroom» type=«Enemy» gid=«73» x=«1088»
y=«960» width=«64» height=«64»/>
  <object id=«29» name=«mushroom» type=«Enemy» gid=«73»
x=«1921.33» y=«640» width=«64» height=«64»/>
  <object id=«30» name=«mushroom» type=«Enemy» gid=«73»
x=«2109.33» y=«642.667» width=«64» height=«64»/>
  <object id=«31» name=«Collider» x=«1858.67» y=«576» width=«64»
height=«64»/>
  <object id=«32» name=«Collider» x=«2170.67» y=«577.333»
```

```
width=«64» height=«64»/>
 </objectgroup>
 <objectgroup id=«2» name=«Player»>
  <object id=«3» x=«186» y=«-188»/>
  <object id=«34» name=«Checkpoint» x=«702.667» y=«837.333»
width=«64» height=«64»/>
 </objectgroup>
 <objectgroup id=«5» name=«Exit»>
  <object id=«9» name=«map_test» x=«576» y=«768» width=«64»
height=«128»/>
  <object id=«33» name=«map_test_3» x=«2298.67» y=«513.333»
width=«64» height=«128»/>
 </objectgroup>
 <objectgroup id=«6» name=«Interaction»/>
</map>

map_test_3.py:
<?xml version=«1.0» encoding=«UTF-8»?>
<map version=«1.10» tiledversion=«1.10.2»
orientation=«orthogonal» renderorder=«right-down» width=«50»
height=«17» tilewidth=«64» tileheight=«64» infinite=«0»
nextlayerid=«11» nextobjectid=«50»>
 <tileset firstgid=«1» source=«tilesets/terrain.tsx»/>
 <tileset firstgid=«73» source=«tilesets/Enemies.tsx»/>
 <layer id=«1» name=«Terrain» width=«50» height=«17»>
  <data encoding=«csv»>
14,14,14,14,14,14,14,14,14,14,14,14,14,14,14,14,14,14,14,14,14,1
4,14,14,14,14,14,14,14,14,14,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,
14,14,26,26,26,26,26,26,26,26,26,26,26,26,26,26,26,26,26,26,26,2
6,26,26,14,14,14,14,14,14,14,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,
14,15,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,25,14,14,14,14
,14,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
14,15,0,0,0,0,0,50,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,13,14,14,14
,14,14,14,14,14,14,14,14,14,14,14,14,14,14,14,14,0,0,0,0,0,0,
14,15,37,38,39,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,13,14,14,
14,14,14,14,14,14,14,14,14,14,14,14,14,14,14,14,14,14,14,14,0,0,0,0,0,0,
14,15,49,50,51,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,13,14,14,
14,14,14,14,14,14,14,14,14,14,14,14,14,14,14,14,14,14,14,14,0,0,0,0,0,0,
14,15,61,62,63,0,0,0,0,37,38,39,0,0,0,0,0,0,0,0,0,0,0,0,0,0,25,26,
26,6,14,14,14,14,14,14,14,14,14,14,14,14,14,14,14,14,14,14,0,0,0,0,0,0,
14,15,0,0,0,0,0,0,0,0,49,50,51,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,25,
26,6,14,14,14,14,26,26,26,26,26,26,6,14,14,0,0,0,0,0,0,
14,15,0,0,0,0,0,0,0,0,61,62,63,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,25,26,26,26,27,0,0,0,0,0,0,0,13,14,14,0,0,0,0,0,0,
14,15,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,13,14,14,0,0,0,0,0,0,
14,15,0,0,0,0,0,0,0,0,0,0,0,0,37,38,39,0,0,0,0,0,0,0,0,1,2,2,2,2
,2,2,2,2,2,2,3,0,0,0,0,13,14,14,0,0,0,0,0,0,
14,15,0,0,0,0,0,0,0,0,0,0,0,0,49,50,51,0,0,0,0,0,0,0,0,25,26,26,
26,26,26,26,26,26,26,26,27,0,0,0,0,13,14,14,0,0,0,0,0,0,
14,15,0,0,0,0,0,0,0,0,0,0,0,0,61,62,63,0,0,0,0,0,0,0,0,17,0,0,0,
```

```
0,0,0,0,0,0,0,0,0,0,0,0,13,14,14,0,0,0,0,0,0,
14,15,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,13,14,14,0,0,0,0,0,0,
14,14,2,2,2,2,2,2,2,2,2,2,2,2,2,3,0,0,0,0,0,0,0,0,0,0,1,2,2,3,0,
0,0,0,1,2,2,2,2,2,2,30,14,14,0,0,0,0,0,0,
14,14,14,14,14,14,14,14,14,14,14,14,14,14,14,28,2,2,2,2,2,2,2,2,
2,2,30,14,14,28,2,2,2,2,30,14,14,14,14,14,14,14,14,14,0,0,0,0,0,
0,
14,14,14,14,14,14,14,14,14,14,14,14,14,14,14,14,14,14,14,14,14,1
4,14,14,14,14,14,14,14,14,14,14,14,14,14,14,14,14,14,14,14,14,14
,14,0,0,0,0,0,0
</data>
 </layer>
 <layer id=«10» name=«Water» width=«50» height=«17»>
  <data encoding=«csv»>
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
</data>
 </layer>
 <objectgroup id=«3» name=«Enemies»>
  <object id=«4» name=«mushroom» type=«Enemy» gid=«73» x=«2304»
```

```
y=«896» width=«64» height=«64»/>
  <object id=«5» name=«Collider» x=«2112» y=«832» width=«64»
height=«64»/>
  <object id=«6» name=«Collider» x=«2624» y=«832» width=«64»
height=«64»/>
  <object id=«7» name=«mushroom» type=«Enemy» gid=«73» x=«2500»
y=«900» width=«64» height=«64»/>
  <object id=«8» name=«mushroom» type=«Enemy» gid=«73»
x=«1773.33» y=«894» width=«64» height=«64»/>
  <object id=«22» name=«Collider» x=«1918.67» y=«833.333»
width=«64» height=«64»/>
  <object id=«23» name=«Collider» x=«1597.33» y=«832» width=«64»
height=«64»/>
  <object id=«24» name=«Collider» x=«2180» y=«897.333»
width=«64» height=«64»/>
  <object id=«25» name=«Collider» x=«1856» y=«900» width=«64»
height=«64»/>
  <object id=«26» name=«Collider» x=«1665.33» y=«896» width=«64»
height=«64»/>
  <object id=«27» name=«Collider» x=«957.333» y=«898.667»
width=«64» height=«64»/>
  <object id=«28» name=«mushroom» type=«Enemy» gid=«73» x=«1088»
y=«960» width=«64» height=«64»/>
  <object id=«30» name=«mushroom» type=«Enemy» gid=«73»
x=«2109.33» y=«642.667» width=«64» height=«64»/>
  <object id=«31» name=«Collider» x=«1858.67» y=«576» width=«64»
height=«64»/>
  <object id=«32» name=«Collider» x=«2170.67» y=«577.333»
width=«64» height=«64»/>
 </objectgroup>
 <objectgroup id=«2» name=«Player»/>
 <objectgroup id=«5» name=«Exit»>
  <object id=«49» name=«Exit» x=«194» y=«130» width=«64»
height=«128»/>
 </objectgroup>
 <objectgroup id=«6» name=«Interaction»/>
</map>
```