

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНИХ НАУК

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи освітнього ступеня «бакалавр»
за спеціальністю 122 «Комп'ютерні науки»
(освітня програма «Комп'ютерна графіка та розробка ігор»)
на тему:

«Механіки ігрових процесів в слайдскроллерах з
елементами платформінгу»

Виконав студент групи КН-19-2
ГАЛАС Тимур Тимурович

Керівник роботи:
СУГОНЯК Інна Іванівна

Рецензент:
ПЕТРОСЯН Руслан Валерікович

Житомир – 2023

ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНИХ НАУК

ЗАТВЕРДЖУЮ

Завідувач кафедри комп'ютерних наук

Марина ГРАФ

«05» січня 2023 р.

ЗАВДАННЯ

на кваліфікаційну роботу освітнього ступеня «бакалавр»
за спеціальністю 122 «Комп'ютерні науки»
(освітня програма «Комп'ютерна графіка та розробка ігор»)

Здобувач вищої освіти: **ГАЛАС Тимур Тимурович**

Керівник роботи: **СУГОНЯК Інна Іванівна**

Тема роботи: **«Механіки ігрових процесів в слайдскроллерах з елементами платформінгу»,**

затверджена наказом закладу вищої освіти від **«05» січня 2023 р., №10/с**

Термін здачі закінченої роботи: **«23» червня 2023 р.**

Вихідні дані роботи: об'єктом дослідження є гра в жанрі сайдскроллер екшн з елементами платформінгу. Предметом дослідження є двигун Unity, платформа розміщення публікацій, її побудова на сучасних технологіях розробки.

Консультанти випускної кваліфікаційної роботи із зазначенням розділів, що їх стосуються:

Розділ	Консультант	Дата	
		Завдання видав	Завдання прийняв
1	Сугоняк І.І	02.02.2023р.	02.02.2023р.
2	Сугоняк І.І	03.04.2023р.	03.04.2023р.
3	Сугоняк І.І	15.05.2023р.	15.05.2023р.

Календарний план

№ з/п	Назва етапів випускної кваліфікаційної роботи	Термін виконання етапів роботи	Примітка
1	Постановка задачі. Пошук, огляд та аналіз аналогічних розробок. Формулювання технічного завдання. Опрацювання літературних джерел.	2 лютого 2023 – 3 квітня 2023	Виконано
2	Проектування структури системи	3 квітня 2023 – 12 травня 2023	Виконано
3	Написання програмного коду	12 травня 2023 – 10 червня 2023	Виконано
4	Оформлення пояснювальної записки	12 червня 2023 – 16 червня 2023	Виконано

Здобувач вищої освіти **Тимур ГАЛАС**
Керівник **Інна СУГОНЯК**

РЕФЕРАТ

Пояснювальна записка до кваліфікаційної роботи на тему «Механіки ігрових процесів в слайдскроллерах з елементами платформінгу» складається з переліку умовних скорочень, вступу, трьох розділів, висновків, списку використаної літератури та додатку.

Текстова частина викладена на **47** сторінках друкованого тексту.

Пояснювальна записка має **31** сторінки додатків. Список використаних джерел містить **20** найменування і займає **2** сторінки. В роботі наведено **11** рисунків. Загальний обсяг роботи – **88** сторінок.

У першому розділі були визначені основні вимоги та критерії до проекту, були оглянуті альтернативні ігри, було обрано стек технологій для розробки.

У другому розділі було проведено проектування гри та аналіз механік.

У третьому розділі був розроблено ігрові механіки, додано аудіо-візуальну складову гри, був описаний код головних механік гри.

Висновок містить в собі результати виконаної роботи по розробці гри в жанрі сайдскролер.

					КН.КР.Б – 122 – 23 – ПЗ			
Змн.	Арк.	№ докум.	Підпис	Дата				
Розроб.	Т.Т.ГАЛАС				Механіки ігрових процесів в сайдскроллерах з елементами платформінгу Пояснювальна записка	Літ.	Арк.	Аркушів
Керівник	І.І.СУГОНЯК						3	89
Рецензент	Р.В. ПЕТРОСЯН					Житомирська політехніка, група КН-19-2		
Н. контр.								
Зав. каф.	М.С. ГРАФ							

ABSTRACT

The explanatory note to the qualification work on the topic "Game Mechanics in Side-scrolling Platformers" consists of a list of abbreviations, introduction, three chapters, conclusions, a list of references and an additions.

The text part is set out on 47 pages of printed text.

The explanatory note has 31 pages of appendices. The list of references includes 20 titles and occupies 2 pages. The paper contains 11 pictures. The total volume of the work is 88 pages.

The first chapter defines the main requirements and criteria for the project, reviews alternative games, and selects the technology stack for development.

The second chapter focuses on game design and mechanics analysis.

The third chapter involves the development of game mechanics, the addition of audio-visual components to the game, and a description of the code for the main game mechanics.

The conclusion presents the results of the work done on the development of a side-scrolling platformer game.

					КН.КР.Б – 122 – 23 – ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		4

ЗМІСТ

РЕФЕРАТ	3
ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ	7
ВСТУП	8
РОЗДІЛ 1. АНАЛІЗ НАПРЯМКІВ ТА ПЕРЕГЛЯД ТЕХНОЛОГІЙ ДЛЯ	10
СТВОРЕННЯ ГРИ У 2D СТИЛІ	10
1.1 Особливості реалізації ігрового додаток типу «сайдскроллер»	10
1.2 Аналіз існуючих технологій реалізації ігрових додатків	13
1.3 Обґрунтування вибору інструментальних засобів розробки	17
1.4 Постановка задачі та технічне завдання	18
Висновки до першого розділу	18
РОЗДІЛ 2. ПРОЕКТУВАННЯ СИСТЕМИ ТА СЕРЕДОВИЩА.	
ОЗНАЙОМЛЕННЯ З ІГРОВИМИ МЕХАНІКАМИ	20
2.1 Опис сценарію та структури	20
2.2 Опис геймплейного циклу	24
2.3 Бойова система	26
2.4 Інші механіки, варті зазначення	29
Висновки до другого розділу	32
РОЗДІЛ 3. РОЗРОБКА ТА ПРОГРАМУВАННЯ САЙДСКРОЛЛЕР ГРИ	34
3.1 Розробка механік, пов'язаних з ігровим персонажем	34
3.2 Розробка інших скриптів та механік	44
3.3 Розробка інших аспектів гри	51
Висновки до третього розділу	53
ВИСНОВКИ	55
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	56
ДОДАТОК	58

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

HP - health point, показник здоров'я.

SP - stamina point, показник витривалості.

JSON - JavaScript Object Notation

.TXT - розширення текстового файлу

Лор — це фонові інформація про світ гри, його історію, культуру, традиції та інші деталі.

SFX - звукові ефекти.

MFx - музика та музичні ефекти..

Бектрекінг - повернення на вже вивчені гравцем локації.

Хаб - сленг в ігровій спільноті, який використовують для назви безпечної для гравця локації, де можна спілкуватися, купляти, продавати та займатися іншими речами в безпеці..

Quality of life features - в контексті відеоігор стосується функцій або аспектів ігрового дизайну, які гарантують, що гравці мають плавний ігровий процес і не вигорають так швидко.

Souls-like - це жанр комп'ютерних ігор, що виник як результат популярності серії ігор Dark Souls, розробленої компанією FromSoftware.

ГГ - Головний Герой / Головна Героїня.

					КН.КР.Б – 122 – 23 – ПЗ	Арк.
						7
Змн.	Арк.	№ докум.	Підпис	Дата		

ВСТУП

У сучасному світі комп'ютерні ігри займають вагоме місце в розвагах і розвитку суспільства. Розмаїття жанрів та інноваційність геймплею роблять геймерське середовище всецільнішим та захоплюючим. Один із цікавих жанрів, що завоював велику популярність, є слайдскроллер з елементами платформінгу.

Слайдскроллери є особливо захопливими для гравців завдяки своїй неперервності та динамічності. У цих іграх гравець контролює героя, який автоматично рухається вперед по рівню. Він повинен уникати перешкод, збирати бонуси та завершувати рівні, одночасно підкорюючи висоти платформінгу. Саме в поєднанні цих елементів лежить основна цікавість та виклик для гравця. Дослідження в цьому питанні потребують основні аспекти геймплею цього жанру, враховуючи вплив механік на загальний досвід гри, взаємодію гравця з оточенням та способи досягнення успіху

Метою кваліфікаційної роботи є аналізі технологій ігрових механік в слайдскроллерах з елементами платформінгу та реалізація механіки ігрових процесів для ігри.

Завданнями кваліфікаційної роботи є:

- аналіз напрямків та технологій для створення гри у 2d стилі;
- проектування системи ігрових механіками для слайдскроллер;
- розробка визуального середовща для упраління користуваче ігровими меаніками;
- розробка механік бойвої системи та моханік ігрового персонажу;
- розрозка механік та інхйї скриптів другорядних функцій ігри.

Предмет дослідження є процес реалізації взаємодії користувача та ігрового додатку типу «слайдскроллер» на основі ігрових механік

Об'єкт дослідження: технології реалізації шгрових механік взаємодії користувача з ігрою для різних типів задач

В роботі використано методи об'єктно-орієнтованого проектування та програмування, методи візуальної розробки додатків, методи та технології підтримки взаємодії користувача з ігровими додатками

					КН.КР.Б – 122 – 23 – ПЗ	Арк.
						9
Змн.	Арк.	№ докум.	Підпис	Дата		

РОЗДІЛ 1. АНАЛІЗ НАПРЯМКІВ ТА ПЕРЕГЛЯД ТЕХНОЛОГІЙ ДЛЯ СТВОРЕННЯ ГРИ У 2D СТИЛІ

1.1 Особливості реалізації ігрового додаток типу «сайдскроллер»

Сайдскролер - це тип комп'ютерних ігор, в яких геймплей відбувається в горизонтальному напрямку, і гравець пересувається від лівого краю екрану до правого (або навпаки). У таких іграх графічний простір зазвичай ширший, ніж екран, тому екран прокручується в бік, щоб показати різні ділянки гри. Гравець бачить тільки обмежену область, яка поміщається на екрані, тоді як решта гри знаходиться за межами видимості.

Особливості реалізації сайдскролерів можуть варіюватися залежно від платформи та движка гри, але основні принципи такі:

1. Екранна прокрутка: Гра відображає лише обмежену частину рівня на екрані, яка зазвичай пересувається вгору, вниз, вліво або вправо по мірі руху гравця. Нові об'єкти та ділянки рівня постійно з'являються з одного краю екрана, тоді як ті, що виходять за межі, видаляються.

2. Анімація персонажа: Головний герой або ігровий об'єкт пересувається горизонтально або вертикально по екрану. Він може мати анімацію ходьби, стрибків, атаки тощо. Також інші об'єкти гри можуть бути анімовані.

3. Колізії та фізика: Для взаємодії героя з оточенням, реалізуються механізми колізій та фізики. Це означає, що герой може зіштовхуватись зі стінами, перешкодами, ворогами або іншими об'єктами в грі.

4. Події та вороги: Гра може містити різноманітні події, такі як ворожі атаки, збір предметів, вирішення головоломок тощо. Вороги можуть рухатись, атакувати або мати свої власні патерни поведінки.

5. Паралакс: Щоб створити відчуття глибини та пластичності, сайдскролери часто використовують ефект паралаксу. Це означає, що об'єкти на передньому плані рухаються швидше, ніж об'єкти на задньому плані, що надає ілюзію глибини.

					КН.КР.Б – 122 – 23 – ПЗ	Арк.
						10
Змн.	Арк.	№ докум.	Підпис	Дата		

6. Управління: Гравці можуть керувати героєм за допомогою клавіатури, геймпада або інших контролерів. Управління може включати ходьбу, біг, стрибки, атаки та взаємодію з об'єктами в грі.

Ці особливості залежать від конкретної гри, розробників і використовуваного движка, але загальні принципи сайдскролерів залишаються схожими.

У даній грі ми збираємося використати основу ігор Souls-like. Термін "Souls-like" використовується для опису ігор, які відтворюють певні ключові елементи геймплею, настрою та стилістики Dark Souls.

Основні риси жанру Souls-like включають:

Висока вимогливість: Souls-like ігри відомі своєю складністю. Вони часто вимагають від гравця майстерності, стратегічного мислення та точного виконання. Вороги можуть бути сильними, ловкими та вимагати доброго розуміння бойової системи.

Система битви: Souls-like ігри пропонують унікальну систему битви, яка поєднує реалістичність і стратегічність. Гравець може виконувати атаки, блокувати, ухилятися від ударів та використовувати спеціальні рухи. Часто акцент робиться на вагомих рішеннях та розумному використанні ресурсів.

Великий світ і нелінійність: Souls-like ігри відомі своїми просторами, деталізованими світами, які гравцеві дають велику свободу дослідження. Зазвичай гра пропонує нелінійну структуру, де гравець може вибирати різні шляхи та вирішувати завдання в будь-якому порядку.

Система перевірки пам'яті: В Souls-like іграх зазвичай використовується система контролю точок збереження (checkpoint) або бонфаєрів, які слугують місцями відновлення гравця після смерті. При смерті гравець втрачає частину накопиченої гри, але має можливість повернутись до місця своєї смерті для повернення втраченого.

Атмосфера і загадковість: Souls-like ігри відомі своїм особливим настроєм, містичністю та загадковістю. Сюжет може бути заплутаним, а історія та

					КН.КР.Б – 122 – 23 – ПЗ	Арк.
						11
Змн.	Арк.	№ докум.	Підпис	Дата		

глибина світу розкриваються через фрагменти тексту, діалоги з персонажами та оточуючі знахідки.

Загалом, жанр Souls-like передає складність, темну атмосферу, велику свободу гравця та вимогливий геймплей, які стали визнакою серії Dark Souls та її наслідників.

У нашому додатку ми прагнемо забезпечити гравця набором інструментів, які дозволять йому насолодитись захоплюючою історією, під час якої він буде досліджувати і відчувати задоволення від геймплею.

Акцент історії буде зроблено на геймовому світі та увагу приділено лору, який зацікавить гравця й зробить йому віру в його життєздатність.

Геймплей буде фокусуватись на проходженні просторових головоломок та сутичках з ворогами. Глибока система переміщення надасть гравцю інструменти для їх вирішення.

Бойова система буде базуватися на іграх жанру Souls-Like, що дозволить гравцю відчути реалістичність та стратегічність бою. Особливу увагу приділиться вагомим рішенням та розумному використанню ресурсів.

Звукове супроводження гри має на меті передати атмосферу локацій та ще глибше занурити гравця у світ гри.

Музичний супровід буде відтворювати тон та емоції кожної локації. Усі музичні треки будуть створені в оркестровому стилі, щоб передати атмосферу середньовіччя та темного фентезі.

Звукові ефекти будуть сконцентровані на передачі відчуття вагомості та живості головного героя. Будуть присутні звуки кожної дії персонажа, від руху до використання предметів.

Усі вищеописані елементи створять умови, за яких гравець зможе насолодитись захоплюючою історією, пов'язаною з викликами, які ставляться перед ним у геймплеї. У жанра ігор Souls-like зростає популярність, оскільки багато сучасних ігор намагаються спростити вхідні бар'єри, щоб залучити більше потенційних гравців. В результаті геймери, які вважають себе середніми

					КН.КР.Б – 122 – 23 – ПЗ	Арк.
						12
Змн.	Арк.	№ докум.	Підпис	Дата		

або важкими гравцями, відчувають дефіцит ігор, призначених спеціально для них.

1.2 Аналіз існуючих технологій реалізації ігрових додатків

Аналіз аналогів програмного продукту є важливим етапом у постановці задачі розробки гри у 2D стилі. Дослідження існуючих аналогів дозволяє виявити їхні сильні та слабкі сторони, визначити можливі прогалини на ринку, а також отримати інспірацію та ідеї для поліпшення власного продукту. Основні завдання аналізу аналогів програмного продукту включають:

Вивчення існуючих ігор у 2D стилі: проведення огляду та детального аналізу існуючих ігор, що відповідають задуманому жанру та цільовій аудиторії. Дослідження їхнього геймплею, графіки, звукового оформлення та інших ключових аспектів.

Визначення сильних і слабких сторін аналогів: ідентифікація основних переваг та недоліків існуючих ігор у 2D стилі. Врахування факторів, що можуть бути важливими для цільової аудиторії, таких як геймплейна глибина, графічний стиль, інноваційність тощо.

Вивчення рецензій та оцінок: аналіз відгуків гравців, рецензій, оцінок та коментарів про існуючі аналоги. Виявлення позитивних та негативних аспектів, які були відзначені споживачами.

Ідентифікація недоліків на ринку: виявлення прогалин або недоліків, які можуть бути заповнені новим продуктом. Аналіз сильних сторін конкурентів, щоб визначити області, де можна виявити перевагу.

Запозичення ідей та інновацій: пошук ідей, рішень та інновацій, які можуть бути використані для покращення власного продукту. Виявлення особливих функцій чи підходів, які привернули увагу гравців у схожих іграх.

Аналіз аналогів програмного продукту допомагає зрозуміти конкурентне середовище, ідентифікувати свої конкурентні переваги та надати цінну інформацію для подальшого проектування та розробки гри у 2D стилі.

1) Salt and Sanctuary є 2D екшн-рольовою грою, що поєднує в собі елементи платформінгу та грайндера. Гра відзначається хорошою механікою

					КН.КР.Б – 122 – 23 – ПЗ	Арк.
						13
Змн.	Арк.	№ докум.	Підпис	Дата		

бою, розмаїттям зброї та навичками, а також деталізованим темним світом і викликів, що постійно збільшуються. Salt and Sanctuary надає гравцям велику свободу досліджувати світ гри, виконувати завдання та розвивати свого героя.[1]



Рис.1.1. Гра Salt and Sanctuary

Гра Salt and Sanctuary отримала позитивні відгуки за свою атмосферу та ігровий дизайн, проте є кілька потенційних недоліків, які деякі гравці можуть відчувати:

1. Візуальна естетика: Гра використовує піксельне зображення, що може не всім сподобатись. Деякі гравці можуть відчувати, що візуальна стилістика обмежує виразність гри та затемнює деякі деталі.

2. Контроль та рухова механіка: Деякі гравці можуть відчувати, що контролювання персонажа в Salt and Sanctuary не є настільки точним або респонсивним, як вони б хотіли. Це може призвести до відчуття незручності під час битви та маневрування.

3. Велика складність та вимогливість: Salt and Sanctuary відома своїм високим рівнем складності та потребою в ретельному плануванні та виконанні стратегій. Це може виявитись викликом для менш вправних гравців або тих, хто шукає більш недвозначний досвід гри.

4. Відсутність мультиплеєра: Гра не має офіційної можливості грати у режимі спільної гри або мультиплеєра. Це може бути розчаруванням для гравців, які б хотіли досліджувати світ гри разом з друзями.

					КН.КР.Б – 122 – 23 – ПЗ	Арк.
						14
Змн.	Арк.	№ докум.	Підпис	Дата		

2) Hollow Knight - це метроїдванія-платформер, який вражає своїм візуальним стилем та майстерністю виконання. Гра пропонує незабутню атмосферу, масштабні локації для дослідження, вражаючу графіку та анімацію. Вона відома своїми складними битвами з босами, глибокими загадками та інтригуючим сюжетом. Hollow Knight також використовує механіку збору та вдосконалення різноманітних зброї та навичок.[2]



Рис.1.2. Гра Hollow Knight

Гра Hollow Knight отримала велику популярність і хвали за свій атмосферний світ і глибокий геймплей, але все ж таки є кілька потенційних недоліків, які деякі гравці можуть відчувати:

Висока складність: Hollow Knight відома своєю високою складністю, особливо в битвах зі босами та дослідженні вимогових місць. Це може стати викликом для менш вправних гравців або тих, хто шукає більш розслабленого досвіду гри.

Відсутність явного направлення: Гра дає гравцеві велику свободу в дослідженні світу Hollow Knight, але водночас може бути неясним щодо основних цілей та напрямку. Деякі гравці можуть відчувати себе загубленими або збентеженими через брак чіткого провідника.

Значна затримка перед збереженням: У Hollow Knight відсутня можливість зберігати гру в будь-якому місці. Гравцям потрібно знайти

					КН.КР.Б – 122 – 23 – ПЗ	Арк.
						15
Змн.	Арк.	№ докум.	Підпис	Дата		

спеціальні пункти збереження, що може бути проблемою, якщо гравець не має достатньо часу для продовження гри або хоче вийти.

Велика кількість бектрекінгу: У грі можуть виникати ситуації, коли гравцеві доводиться повертатись до вже досліджених місць для досягнення нових цілей або виконання завдань. Це може викликати відчуття повторності та втоми у деяких гравців.

3) Cuphead - це унікальна 2D гра, яка стилізована під мультфільм 1930-х років. Вона відзначається ретро-графікою, захоплюючими бос-битвами та швидким, вимогливим геймплеєм. Гра пропонує гравцям випробувати свою спритність та реакцію в різноманітних випробуваннях, які вимагають точного планування і виконання. Cuphead також має кооперативний режим, що дозволяє гравцям спільно боротися з викликами гри.[3]

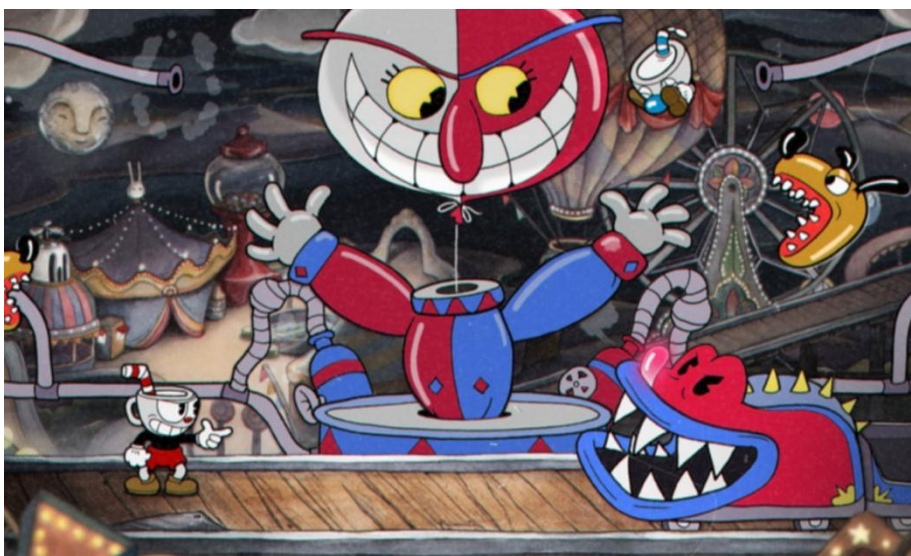


Рис.1.3. Гра Cuphead

Хоча гра Cuphead отримала значну популярність і визнання за свій унікальний стиль та викликів, які вона ставить перед гравцем, все ж є кілька потенційних недоліків, з якими можуть зіткнутись деякі гравці:

1. Висока складність: Cuphead відома своїм надзвичайно високим рівнем складності. Це може виявитись проблемою для деяких гравців, особливо для тих, хто шукає більш розслабленого та менш вимогливого досвіду гри.

2. Необхідність великої кількості повторних спроб: У грі Cuphead потрібно багато тренувань та повторних спроб, щоб пройти складні рівні та

					КН.КР.Б – 122 – 23 – ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		16

перемогти босів. Це може викликати відчуття фрустрації та втоми у деяких гравців.

3. Можливість відчуття монотонності: У грі Cuphead багато рівнів мають схожий стиль та атмосферу, що може привести до відчуття монотонності у деяких гравців. Брак різноманітності в геймплеї та локаціях може зменшити захоплення гри на довгострокову перспективу.

4. Брак режиму спільної гри: Cuphead не має офіційного режиму спільної гри для багатьох гравців, що може бути розчаруванням для тих, хто шукає можливість грати разом з друзями або онлайн.

Важливо враховувати, що деякі з недоліків цих ігор можуть бути частиною загального досвіду гри та створювати виклик, який деякі гравці знаходять захоплюючим. Однак вони також можуть бути факторами, які впливають на задоволення від гри у деяких випадках.

1.3 Обґрунтування вибору інструментальних засобів розробки

При виборі інструментів та засобів для реалізації проекту гри у 2D стилі з елементами платформінгу, Unity є одним із найпопулярніших та потужних варіантів. Unity - це інтегроване середовище розробки (IDE), яке надає широкі можливості для створення гри, включаючи підтримку 2D графіки, фізики, анімації та інших важливих функцій.[4] Ось кілька причин, чому Unity може бути вдалим вибором для вашого проекту:

Підтримка 2D графіки: Unity надає розширені можливості для створення і редагування 2D графіки. Ви можете створювати та редагувати спрайти, реалізовувати анімацію, керувати шарами та здійснювати інші графічні налаштування.[5]

Фізика та колізії: Unity має вбудовану фізичну систему, яка дозволяє створювати реалістичну поведінку об'єктів у грі. Ви можете встановлювати правила колізій, сили тяжіння, реалізовувати рухи та інші фізичні ефекти.[6]

Мови програмування: Unity підтримує кілька мов програмування, включаючи C# і JavaScript (UnityScript). Ви можете вибрати мову, з якою вам комфортно працювати та яка відповідає вашим потребам.[7]

					КН.КР.Б – 122 – 23 – ПЗ	Арк.
						17
Змн.	Арк.	№ докум.	Підпис	Дата		

Спільнота та ресурси: Unity має велику спільноту розробників, яка надає безліч ресурсів, документацію, пакети активів та підтримку. Ви можете знайти відповіді на свої питання, вирішити проблеми та навчитися новим технікам через цю спільноту.[8]

Мультиплатформеність: Unity дозволяє розробляти гри для різних платформ, включаючи ПК, мобільні пристрої, консолі та інші. Це дає можливість досягти широкої аудиторії та покрити різні пристрої.[9]

З урахуванням цих переваг, використання Unity може бути ефективним вибором для реалізації вашого проекту гри у 2D стилі з елементами платформінгу. Знання Unity, спільнота розробників та наявність різноманітних ресурсів допоможуть прискорити розробку та досягти бажаних результатів[10].

1.4 Постановка задачі та технічне завдання

Ми встановили перед собою такі головні задачі:

1. Розробка системи переміщення: створення реагуючої системи механіки руху, щоб гравець міг пересуватися по рівнях та подолати перешкоди.
2. Розробка системи бойових механік: створення цікавої та різноманітної системи бою, яка змушує гравця швидко приймати важливі рішення.
3. Розробка системи інвентаризації: створення зручного механізму для збирання предметів на різних локаціях та їх використання для полегшення геймплею або покращення головного героя.
4. Розробка системи діалогів з персонажами: надання гравцеві можливості спілкуватися з персонажами в головному хабі, які просуватимуть сюжет, розкриватимуть фонову історію та надаватимуть допомогу в подорожі.
5. Реалізація інших механік, які не входять в жодну з категорій: створення додаткових механік, що покращать геймплей гравця та quality of life features.

Висновки до першого розділу

В цьому розділі ми розглянули постановку задачі дипломної роботи, провели аналіз аналогів програмного продукту та визначили вибір інструментів та засобів для реалізації проекту.

					КН.КР.Б – 122 – 23 – ПЗ	Арк.
						18
Змн.	Арк.	№ докум.	Підпис	Дата		

Під час аналізу аналогів, таких як Salt and Sanctuary, Hollow Knight та Cuphead, ми з'ясували, що гри у 2D стилі з елементами платформінгу можуть відрізнятися за своїм геймплеєм, графікою, атмосферою та механіками. Ці аналоги слугують важливим джерелом натхнення для створення власної гри.

Вибір Unity як основного інструменту для реалізації проекту гри у 2D стилі є обґрунтованим через його потужність, підтримку 2D графіки та фізики, розширені можливості програмування, наявність спільноти розробників та ресурсів.

Загальний висновок полягає в тому, що гра у 2D стилі з елементами платформінгу є захопливим та цікавим жанром, що має великий потенціал для творчості та інновацій.

					КН.КР.Б – 122 – 23 – ПЗ	Арк.
						19
Змн.	Арк.	№ докум.	Підпис	Дата		

РОЗДІЛ 2. ПРОЕКТУВАННЯ СИСТЕМИ ТА СЕРЕДОВИЩА. ОЗНАЙОМЛЕННЯ З ІГРОВИМИ МЕХАНІКАМИ

2.1 Опис сценарію та структури

Давайте поговоримо про фантастичний світ гри, де головні події відбуваються у середньовічній фентезійній епосі. Головний континент цього світу називається Кілла, і на ньому знаходяться чотири основні країни:

1. Імперія Місяця - це королівство, де люди є домінуючою расою, а інші раси відчують пригнічення. Імперія займає найбільшу територію і вважається найсильнішою в економічному та військовому плані. У цій країні головну роль відіграє церква Іруніла, яка посідає провідне місце у пропаганді расизму проти інших рас.

2. Арандар - це держава ельфів, яка охоплює тільки територію Лісу Лаена. Ельфи живуть в ізоляції та уникають контактів з іншими країнами та расами.

3. Гек - це країна дворфів, розташована у Горах Геку, від яких вона отримала свою назву. Головні галузі промисловості та торгівлі в цій країні пов'язані з видобуванням руди. Міста дворфів розташовані під землею, в скелях, тому вони вважаються неприступними.

4. Легіон Вагну - це країна демонів, що знаходиться у кам'яній пустелі на півдні. В цій країні єдиним законом є закон сили, тому вона розділена на багато кланів, які ведуть між собою війни за територію та цінні ресурси. Коли з'являється клан, який здобуває перевагу над іншими, він оголошує Хрестовий похід на території інших країн, щоб досягти своїх цілей. Під час цього походу інші країни об'єднуються в альянс, щоб протистояти їхньому нападу.

Назва гри "NameLess", яка передає всю суть головного героя. Головною героїнею гри є Жнець помсти, яка втратила свою пам'ять в обмін на отриману силу. Після смерті, один з богів обрав її своїм пророком, передавши їй свою силу, але взамін вимагав її спогади. Головною метою гравця та самої Жнеці є знайти спосіб відновити її пам'ять. Для досягнення цього, вона повинна досліджувати різні місця, де можуть бути вказівки щодо досягнення цілей, а також здобувати корисну інформацію від інших персонажів.

					КН.КР.Б – 122 – 23 – ПЗ	Арк.
						20
Змн.	Арк.	№ докум.	Підпис	Дата		

Давайте поговоримо про деяких персонажів, з якими можна взаємодіяти:

1. Даріель - це кузнець з міста Ван, який допомагатиме головній героїні поліпшувати її клинок, використовуючи міфрилову руду як плату за послуги.

2. Вермеїл - вона є володаркою алхімічної лавки. Вона згодна поліпшувати флягу зцілення головної героїні в обмін на рідку квітку Суно. Вермеїл використовує магію ілюзій, щоб приховати свій справжній зовнішній вигляд, оскільки вона є Висшим вампіром, якому хотілося б побачити світ, де вампіри та люди можуть бути друзями. Гравець помітить ілюзію, але не зможе змінити цю ситуацію до тих пір, поки не відправиться до замку Волкихар та не знайде там картину, на якій буде зображена справжня Вермеїл.

3. Жнець полювання - це інший Жнець, схожий на головну героїню. Він недовірливий, закритий та надзвичайно обережний. Жнець полювання запропонував угоду Жнецю помсти: вона збирає докази про злочини церкви Іруніла, а він, в свою чергу, допомагає їй відновити пам'ять. Раніше він був вірним послідовником церкви та навіть мав титул паладина, але був зраджений і вбитий через ненадібність. Після цього він став Жнецем і поклав свою мету знищити цю організацію. Протягом гри, він поступово почне довіряти головній героїні та зігріватиметься до неї.

Таким чином, персонажі гри виконують важливу роль у прогресі головної сюжетної лінії та допомагають гравцеві в досягненні його цілей.

Unity використовує мову програмування C# як свою основну мову, і для реалізації системи гравців використовується класова структура. У проекті ми маємо головний клас `PlayerClass`, який містить змінні, необхідні для роботи інших скриптів, таких як `PlayerMovement`, `FightBehaviour`, `PlayerInv`, `PlayerChara`. Це дозволяє нам розбити код на кілька частин, що реалізують конкретні типи механік. Такий підхід полегшує подальше читання і розуміння коду.

					КН.КР.Б – 122 – 23 – ПЗ	Арк.
						21
Змн.	Арк.	№ докум.	Підпис	Дата		

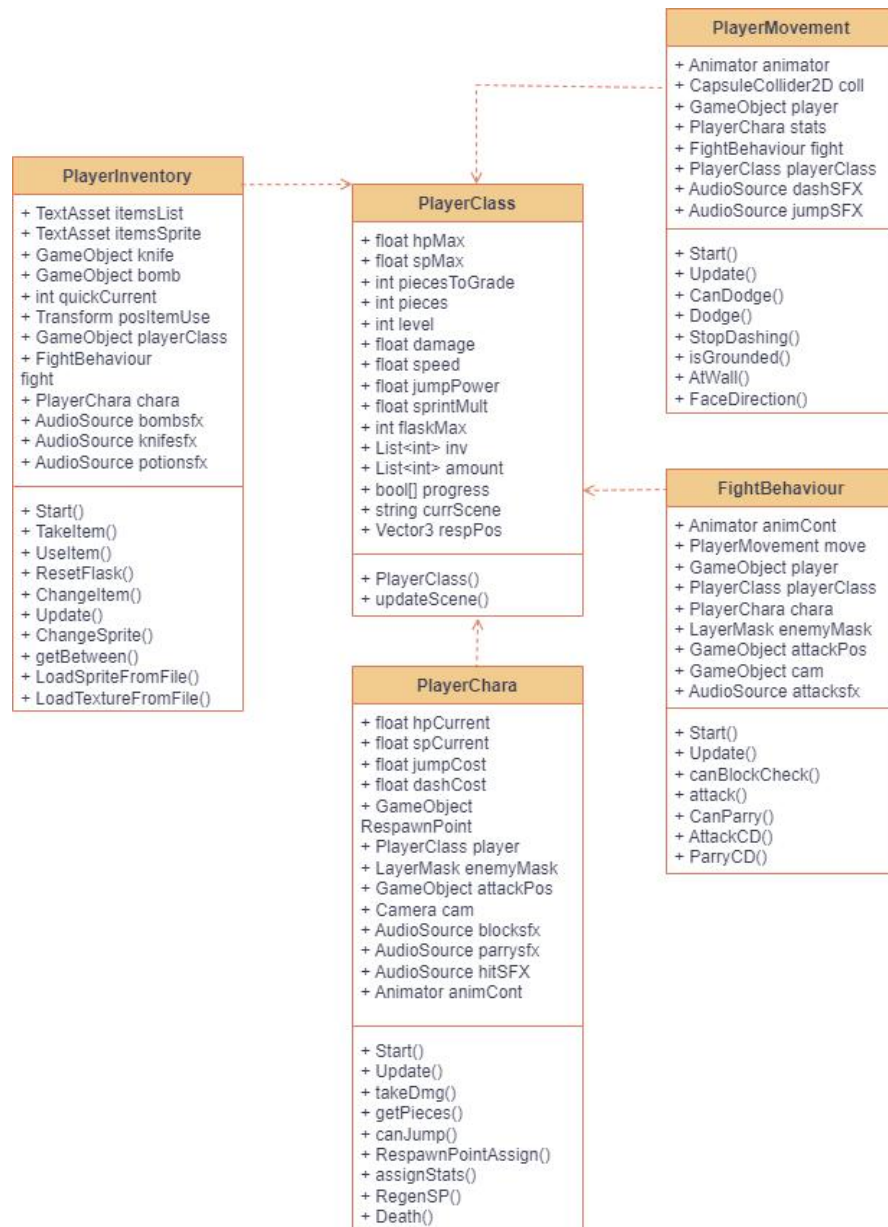


Рис.2.1 Діаграма класів головного героя

У нашій роботі ми вирішили використати модульну систему. Модульна система в програмуванні відноситься до організації програмного коду у відокремлені модулі або компоненти, які виконують конкретні функції або завдання. Кожен модуль представляє собою логічно самостійну одиницю, яка може бути розроблена, тестована та використана окремо або в поєднанні з іншими модулями для створення складніших програм.

Основна ідея модульної системи полягає в тому, щоб розбити програму на менші, більш керовані частини, які легше розуміти, підтримувати і

перевикористовувати. Кожен модуль може мати свою внутрішню реалізацію та інтерфейс, який визначає, як інші модулі можуть взаємодіяти з ним.

Переваги модульної системи включають:

1. Модульність: Код розділяється на окремі модулі, що сприяє зручності управління та розробці. Кожен модуль може бути реалізований та тестований незалежно, а також легко замінений або модифікований без впливу на інші частини програми.

2. Перевикористання: Модулі можуть бути використані в різних проектах або в одному проекті більше одного разу, що сприяє ефективному використанню ресурсів та забезпечує єдність у функціональності.

3. Зрозумілість: Ізольовані модулі полегшують розуміння коду. Кожен модуль відповідає за конкретну функціональність, що спрощує відладку та підтримку.

4. Командна робота: Коли кілька розробників працюють над проектом, модульна структура дозволяє їм працювати над окремими модулями незалежно один від одного, зменшуючи конфлікти та сприяючи паралельному розвитку.

Модульна система допомагає створювати більш організовані, ефективні та легко збережені програми, сприяючи швидкому розвитку та підтримці кодової бази.

На прикладі механіки переміщення між рівнями, ми можемо побачити що весь код розбитий на дрібні самостійні частини, які працюють незалежно один від одного.

					КН.КР.Б – 122 – 23 – ПЗ	Арк.
						23
Змн.	Арк.	№ докум.	Підпис	Дата		

Name	Date modified	Type	Size
ConfScript.cs	6/22/2023 3:41 PM	C# Source File	2 KB
ConfScript.cs.meta	6/22/2023 3:41 PM	META File	1 KB
exitTravel.cs	6/22/2023 3:41 PM	C# Source File	1 KB
exitTravel.cs.meta	6/22/2023 3:41 PM	META File	1 KB
Lvl_btnSRC.cs	6/22/2023 3:41 PM	C# Source File	2 KB
Lvl_btnSRC.cs.meta	6/22/2023 3:41 PM	META File	1 KB
Shrine_btnSRC.cs	6/22/2023 3:41 PM	C# Source File	3 KB
Shrine_btnSRC.cs.meta	6/22/2023 3:41 PM	META File	1 KB
TravelButton.cs	6/22/2023 3:41 PM	C# Source File	1 KB
TravelButton.cs.meta	6/22/2023 3:41 PM	META File	1 KB
TravelPanel.cs	6/22/2023 5:13 PM	C# Source File	5 KB
TravelPanel.cs.meta	6/22/2023 3:41 PM	META File	1 KB

Рис.2.2. Приклад модульної системи

2.2 Опис геймплейного циклу

У проєкті дуже важливо гарно розуміти, як саме буде функціонувати гра. Необхідно встановлювати чіткі пріоритети та мати чітке уявлення про те, яким чином гра повинна працювати.

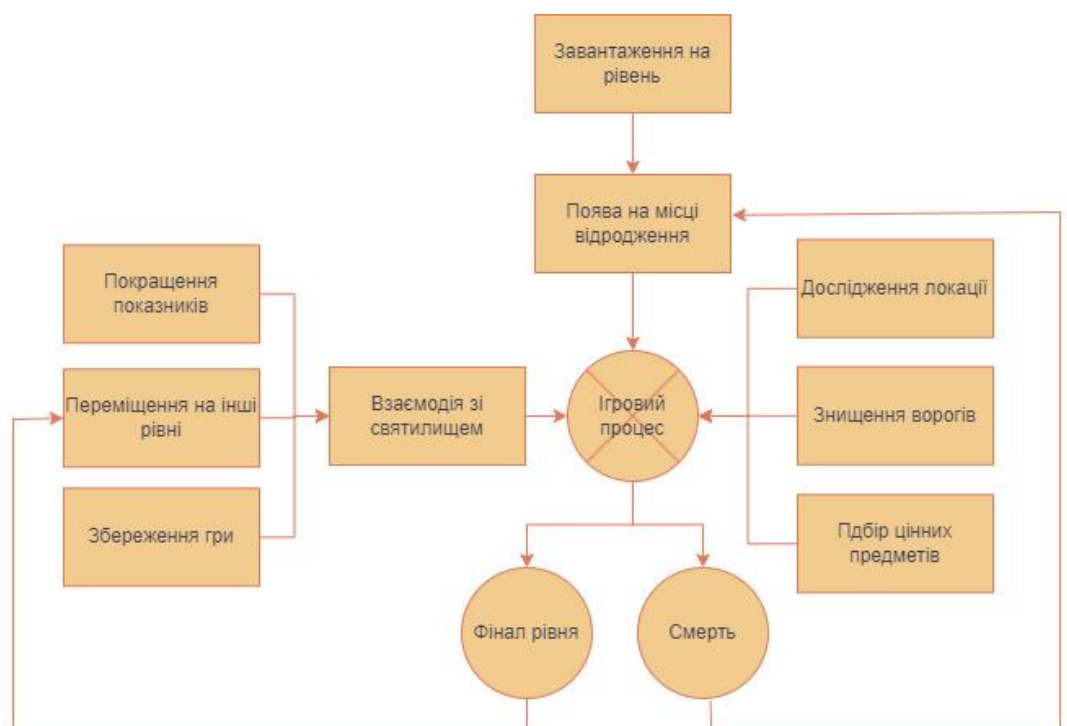


Рис.2.3 Блок-схема ігрового циклу

Ця блок-схема представляє основний хід геймплею залежно від дій гравця в грі. Нижче наведено опис кожного елемента блок-схеми:

- Гравець: Початкова точка, з якої розпочинається гра.
- Відпочинок(сбереження): Гравець може відпочити або зберегти свій прогрес у грі на цій точці.
- Вдосконалення показників за монети: Гравець може покращити свої характеристики або здібності, використовуючи зібрані монети.
- Переміщення на інші рівні: Гравець може перейти на наступний рівень гри після виконання певних умов.
- Смерть: Якщо гравець помер, він повинен повторити гравецький процес з попередньої точки відпочинку або іншої визначеної точки.
- Ігровий процес: Основний хід гри, де гравець взаємодіє з оточуючим світом, вбиває НПів для отримання предметів і монет, а також досліджує локації для знаходження корисних предметів.
- Фінал рівня: Гравець досягає фінальної точки рівня, що означає успішне завершення цього рівня.

Ця блок-схема відображає цикл геймплею, включаючи здобуття ресурсів, переміщення гравця, виконання завдань та досягнення фінальних точок, а також можливість відпочити та зберегти прогрес.

Основою цієї 2D гри є рух, який відіграє ключову роль у її розробці. Швидкість та маневреність є важливими аспектами у цьому типі гри. Швидке переміщення дозволяє користувачеві ефективно взаємодіяти з ворогами та швидко досліджувати гру.

У даному проекті користувач буде мати такі можливості:

1.Швидке переміщення по мапі: гравець може швидко рухатися по ігровій карті, що дозволяє ефективно переміщатися між різними областями.

2.Прижок для переміщення між платформами: гравець може здійснювати стрибки, щоб пересуватися між різними платформами або об'єктами на мапі.

					КН.КР.Б – 122 – 23 – ПЗ	Арк.
						25
Змн.	Арк.	№ докум.	Підпис	Дата		

3.Деш (довгий стрибок): гравець може виконувати довгі стрибки, що дозволяють йому переміститися на великі відстані або перескочити ворогів.

Ці компоненти руху достатні для комфортного пересування по грі, а головне, що гра не буде відчуватися повільною. Це створить враження швидкого темпу. Далі показано діаграму, яка демонструє, як ці можливості будуть відтворені у грі:

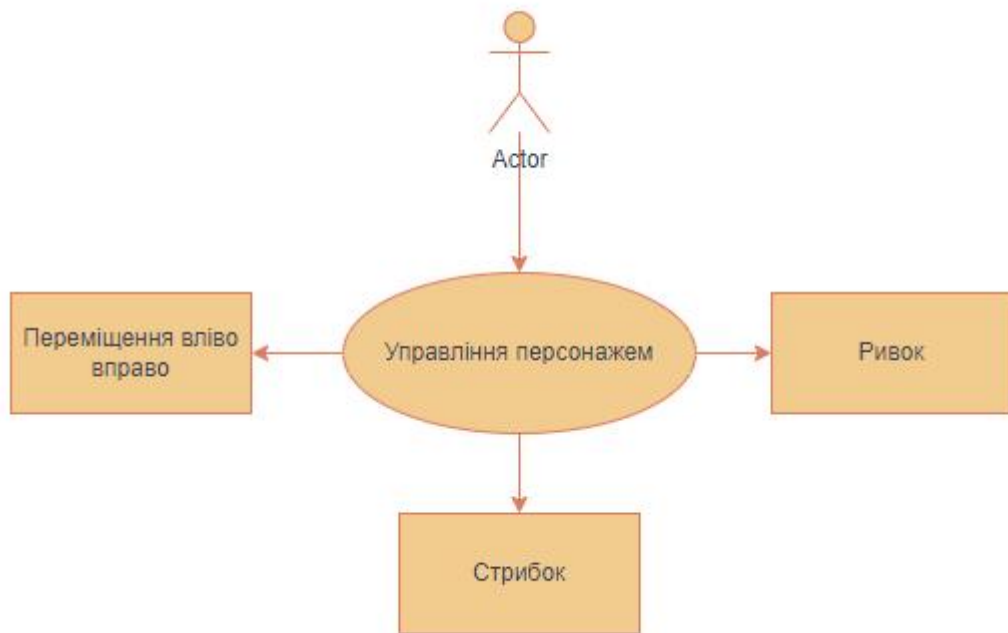


Рис.2.4. Діаграма механік переміщення

2.3 Бойова система

У грі в жанрі souls-like, бойова система зосереджена на ухиленнях та парируванні. Гравець може виконувати удари, а також матиме можливість використовувати блок і парирування. Ось як працюють ці елементи:

ЛКМ (ліва кнопка миші): Використовується для здійснення удару.

ПКМ (права кнопка миші): Використовується для блоку. Гравець може встановити захисну позицію, натиснувши праву кнопку миші. Це дозволяє гравцю повністю відбити атаку противника, якщо він вчасно натисне кнопку блоку.

СКМ (середня кнопка миші): Використовується для парирування. Гравець може активувати режим парирування, натиснувши середню кнопку миші. Це дозволяє гравцю синхронізувати свою атаку з атакою противника і відхилити її, створивши можливість для контратаки.

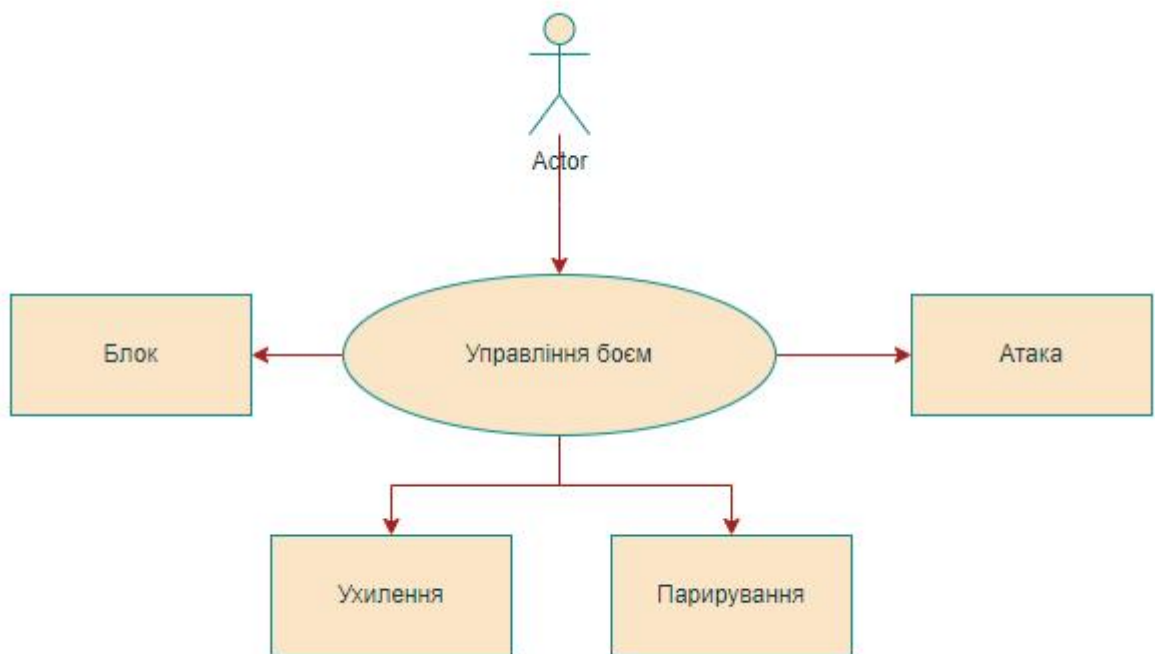


Рис.2.5 Схема з бойовими механіками

Механіка, пов'язана з ворогами, центрується на їхній основній меті - вбивстві гравця. Вороги будуть мати здатність здійснювати атаки, на які гравець зможе реагувати двома способами: відпарюванням або ухиленням.

Відпарювання: Гравець матиме можливість відпарювати атаки ворогів, використовуючи відповідні техніки парирування. Це означає, що гравець зможе точно реагувати на атаку ворога, співставляючи свою власну атаку або блокування з атакою противника.

Ухилення: Гравець також матиме можливість ухилятися від атак ворогів, швидко рухаючись вбік або ухиляючись від удару. Це дозволить гравцеві уникнути отримання ушкоджень, пересуваючись у безпечні позиції та зберігаючи можливість для контратаки.

Обидва варіанти - відпарювання і ухилення - дають гравцю стратегічні варіанти взаємодії з ворогами. Гравець повинен аналізувати атаки ворогів,

вчасно реагувати та обирати найефективнішу тактику, щоб пережити та перемогти ворогів.

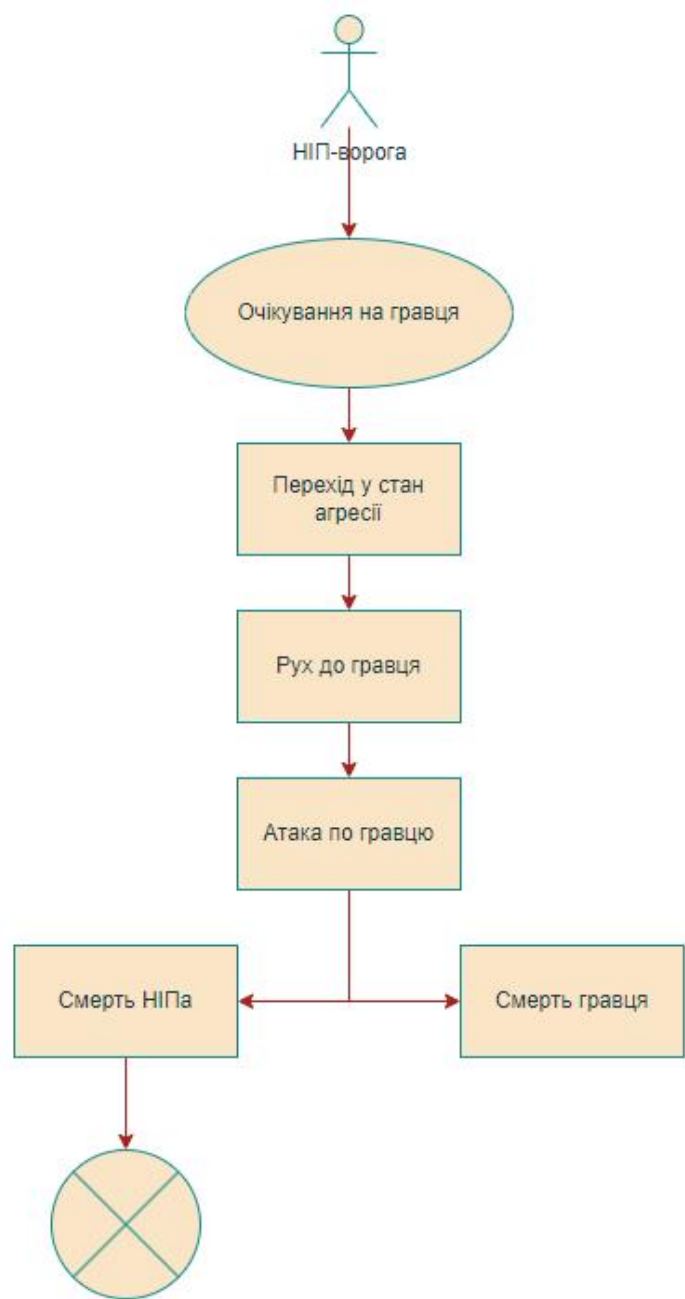


Рис.2.6. Схема поведінки ворога

Далі приведено діаграму класу, за допомогою якого було реалізовано ворогів у грі.



Рис.2.7. Діаграма класа ворогів

2.4 Інші механіки, варті зазначення

Відновлення здоров'я: Гравець може відновити своє здоров'я, відпочиваючи у спеціальних відведених місцях. Це дозволяє гравцеві відновити втрачене здоров'я та підготуватися до подальшого бою.

Лікувальне зілля: Гравець може збирати лікувальне зілля, яке наповнює склянки. Це зілля можна використовувати під час бою, щоб швидко відновити частину втраченого здоров'я. Гравець повинен стратегічно використовувати зілля відповідно до ситуації, зберігаючи їх для найбільш критичних моментів.

					КН.КР.Б – 122 – 23 – ПЗ	Арк.
						29
Змн.	Арк.	№ докум.	Підпис	Дата		

Смерть і відродження: У випадку смерті головного героя, він повертається на місце останнього відпочинку. Однак, він втрачає половину накопичених монет як покарання за поразку. Крім того, всі вороги, яких гравець вже переміг, відновлюються у своїх визначених місцях. Це створює виклик і додає напруження до гри, оскільки гравець повинен бути обережним і стратегічно підходити до кожного бою, збираючи монети та уникати повторної смерті.

Ці бойові механіки створюють динамічну гру, в якій гравець повинен балансувати між ризиком і винагородою, використовуючи різні способи відновлення здоров'я та керуючи своїми ресурсами для ефективного прогресування у грі.

Діалоги з НІП-ами у мирних локаціях:

Таверна: У таверні гравець може зустріти персонажа, який буде направляти героя по подальшим його завданням. Він може розповісти про історію країни, події, що відбулися раніше, або надати корисні поради щодо подальшого прогресування у грі. Гравець може задавати запитання, щоб дізнатися більше деталей і поглибити своє розуміння світу гри.

Крамниця: У крамниці гравець може спілкуватися з торговцем та покращити лікувальне зілля. Торговці можуть надати додаткову інформацію про предмети, їх властивості та застосування.

Кузня: У кузні гравець може зустріти коваля, який може покращити його зброю. Вони можуть надати інформацію про різні види покращень, матеріали, необхідні для апгрейду. Гравець може отримати корисні поради щодо покращення свого екіпірування та бойової ефективності.

Ці діалоги дозволяють гравцеві глибше зануритися в світ гри, розширити свої знання про лор та отримати корисні поради від НІП-ів, що може вплинути на його подальшу геймплей стратегію і взаємодію з оточуючим світом.

Покращення:

Так, під час гри гравець матиме можливість заробляти монети, перемагаючи ворогів. Зібрані монети можна буде використовувати для

					КН.КР.Б – 122 – 23 – ПЗ	Арк.
						30
Змн.	Арк.	№ докум.	Підпис	Дата		

покращення характеристик головного героя. Наприклад, гравець може підвищити його здоров'я або витривалість.

Покращення характеристик дозволить гравцеві стати сильнішим і ефективнішим у бою, збільшити шанси на виживання та подолання складних ворожих перешкод. Кожне покращення буде вимагати витрати певної кількості монет, що створить стимул для гравця збирати їх та приймати стратегічні рішення щодо використання накопичених ресурсів.

Втрата половини невикористаних монет після смерті гравця створює ризик та напруження, спонукаючи його до обережності та уникання небезпеки. Це стимулює гравця грати обережно та стратегічно, оскільки втрата монет може вплинути на його здатність до покращення характеристик та подальший прогрес у грі.

Загалом, система покращень та втрати монет додає грі глибину, ризик та розвиток персонажа, стимулюючи гравця до активної гри та стратегічних рішень.

Звуки та музика у грі:

Звуковий стиль у грі гратиме важливу роль у створенні атмосфери та занурення гравця у середньовічний світ. Він буде відображати атмосферу села або міста середньовічного періоду, з використанням відповідних звуків.

Також, спеціальні ефекти, пов'язані з боєм, відтворюватимуться за допомогою звуків ударів, блокування та парирування. Ці звуки створять реалістичну атмосферу битви і додадуть динаміки до геймплею.

Щодо музики, вона також відповідатиме середньовічному сеттінгу гри. Оркестрові та духові мотиви підкреслять епічність та таємничість гри. Музика буде відтворюватися на задньому плані, допомагаючи створити належну атмосферу під час розвідування середньовічних локацій та ведення боїв.

В цілому, звуковий дизайн та музика гри допоможуть створити незабутню атмосферу та поглибити враження від геймплею.

Ігровий цикл - це поняття, яке використовується в геймдизайні та відноситься до структури та послідовності елементів, що повторюються у

					КН.КР.Б – 122 – 23 – ПЗ	Арк.
						31
Змн.	Арк.	№ докум.	Підпис	Дата		

відеоігрі. Це основна одиниця геймплею, яка складається з послідовності дій та подій, що повторюються, ізольовано одна від одної.

Ігровий цикл гри зображено на рис.2.5. Ми можемо бачити, що гравцю буде чим зайнятися під час його проходження.



Рис.2.8 Ігровий цикл проходження гри

Висновки до другого розділу

Розділ 2 гри включає такі важливі елементи, як геймплей, бойові механіки, діалоги, покращення та звуки/музика. Основна мета цього розділу полягає в описі основних функцій та характеристик, які гравець зустрине під час гри.

У геймплеї гравець матиме можливість переміщатися, виконувати різні дії, такі як прижок, присідання і ривок, а також здійснювати бойові дії. Бойова система буде зосереджена на ухиленнях, парируванні та атаках, що вимагатиме від гравця стратегічного мислення та реакції.

Діалоги з НІП-ами у мирних локаціях допоможуть гравцеві дізнатися більше про лор світу гри та отримати зачіпки до його основної мети - помсти. Це розширить гральний світ і поглибить іммерсію гравця.

Покращення характеристик головного героя за допомогою зароблених монет створить стимул для гравця збирати ресурси та розвивати свого персонажа. Втрата половини монет після смерті підвищить ризик та напруження, стимулюючи обережну та стратегічну гру.

Звуки та музика гри будуть відтворювати середньовічну атмосферу, використовуючи духові та оркестрові мотиви, а також відповідні звуки оточення та боїв. Це додасть глибини та реалізму гральному досвіду.

Розділ 2 створює основу для захопливої та іммерсивної гри, де гравець матиме можливість розвивати свого персонажа, виконувати завдання,

взаємодіяти з НПП-ами та відчувати атмосферу середньовічного світу. Цей розділ визначає ключові механіки та елементи гри, які роблять її унікальною та захоплюючою для гравців.

					КН.КР.Б – 122 – 23 – ПЗ	Арк.
						33
Змн.	Арк.	№ докум.	Підпис	Дата		

РОЗДІЛ 3. РОЗРОБКА ТА ПРОГРАМУВАННЯ САЙДСКРОЛЛЕР ГРИ

3.1 Розробка механік, пов'язаних з ігровим персонажем

Розробка модульного коду для керування гравцем є важливим етапом розробки гри. Створення класу гравця (PlayerClass) з необхідними змінними та параметрами дозволяє ефективно керувати і відстежувати стан головного персонажа.

У класі PlayerClass можуть бути збережені дані, такі як урон, кількість монет, координати на сцені та інші важливі характеристики. Це дозволяє легко відстежувати та зберігати прогрес гравця протягом гри (див.Додаток А).

Важливою перевагою модульної системи є зручність в розробці та налагодженні коду. Кожен модуль або скрипт виконує свою конкретну функцію, що спрощує виявлення й виправлення помилок та проблемних ділянок. Такий підхід полегшує командну роботу та сприяє більшій гнучкості у розробці.

Розділення змінних на регіони з назвами скриптів є хорошою практикою у розробці гри. Це допомагає підтримувати порядок та організованість у коді, а також полегшує подальше користування та зміну цих змінних.

Конструктор в класі PlayerClass використовується для створення об'єкта гравця з заданими початковими значеннями. Це може включати у себе ініціалізацію змінних, налаштування параметрів та будь-які інші дії, які необхідні для початкового налаштування гравця.

Метод updateScene відповідає за оновлення поточної сцени гри. Він отримує назву поточної сцени та виконує необхідні дії в залежності від цієї сцени. Цей метод може бути викликаний в основному циклі гри для постійного оновлення сцени та взаємодії гравця з навколишнім середовищем.

Використання конструктора та методу updateScene дозволяє ефективно управляти головним персонажем та його взаємодією з грою. Вони допомагають зберігати актуальну інформацію про стан гравця та поточну сцену, що є важливим для правильної роботи гри та забезпечення плавного геймплею.

Перейдемо до скрипту PlayerMovement.cs, який відповідає за переміщення гравця. Зауважу, що використання класу PlayerClass для

					КН.КР.Б – 122 – 23 – ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		34

збереження необхідних змінних є хорошою практикою, оскільки це дозволяє уникнути дублювання даних та спростити управління змінними в різних скриптах (див. Додаток Б).

У скрипті PlayerMovement.cs можуть бути виконані такі дії, пов'язані з переміщенням персонажа:

Отримання вхідних даних: Скрипт може відслідковувати вхід користувача, такі як натискання клавіші переміщення. Він може отримувати ці дані через Input систему та інші методи вводу.

Обчислення переміщення: За допомогою отриманих вхідних даних та налаштувань (наприклад, швидкість персонажа), скрипт може обчислити переміщення гравця відповідно до обраного напрямку.

Переміщення персонажа: За допомогою функцій переміщення, таких як Translate або MovePosition, скрипт може змістити позицію персонажа на сцені відповідно до обчисленого переміщення.[11]

Обробка колізій: Скрипт може також виконувати обробку колізій з іншими об'єктами або перешкодами на сцені. Це може включати відскок від стін, зупинку персонажа при зіткненні з іншими об'єктами тощо.

Так, використання методу Start для пошуку необхідних ігрових об'єктів є доброю практикою з точки зору оптимізації гри. Метод Start викликається один раз при початку роботи скрипта, тому пошук ігрових об'єктів виконується лише один раз під час ініціалізації скрипту (Лістинг 3.1).

Це дозволяє зменшити навантаження на процесор шляхом уникнення постійного пошуку об'єктів під час виконання скрипта.[12] Замість цього, після пошуку об'єкти зберігаються в змінних, що дозволяє звертатися до них напрямку без необхідності повторного пошуку.

Такий підхід є ефективним, особливо у великих іграх з багатьма об'єктами, де постійний пошук може призвести до затримок та падіння продуктивності.

					КН.КР.Б – 122 – 23 – ПЗ	Арк.
						35
Змн.	Арк.	№ докум.	Підпис	Дата		

```

Event function 2 BrownyOFF +1
private void Start()
{
    coll = GetComponent<CapsuleCollider2D>();
    player = GameObject.FindWithTag("Player");
    playerClass = GetComponent<PlayerClass>();
    sprite = GetComponent<SpriteRenderer>();
    animator = GetComponent<Animator>();
    rb = GetComponent<Rigidbody2D>();
    stats = GetComponent<PlayerChara>();
    fight = GetComponent<FightBehaviour>();
    groundCheck = GameObject.Find("groundCheck");
}

```

Лістинг.3.1. Start()

Аби не описувати кожен рядок коду зупинимося лише на самому головному та цікавому. Метод Update спрацьовує кожен кадр гри, отже він повинен містити в собі лише головне та не працювати, коли це не потрібно (Лістинг 3.2). Викликаючи return, коли це нам необхідно, ми економимо ресурси програми, що дає нам додатковий бонус до продуктивності продукту[13].

```

void Update()
{
    if (stats.isDead || stats.isStunned || stats.inDialogue || fight.isParry || fight.isAttacking )
    {
        rb.velocity = Vector2.zero;
        return;
    }

    if (GameObject.FindWithTag("MainCamera").GetComponent<CameraFollow>().inPause)
    {
        return;
    }

    if (isDashing && AtWall())
    {
        rb.velocity = Vector2.zero;
        return;
    }
}

```

Лістинг.3.2. Update()

Використання `GetAxisRaw("Horizontal")` дозволяє отримати значення для переміщення гравця вліво або вправо. Значення 1 відповідає натисканню

					КН.КР.Б – 122 – 23 – ПЗ	Арк.
						36
Змн.	Арк.	№ докум.	Підпис	Дата		

клавiші D (рух вправо), а значення -1 - клавiші A (рух вліво). Це дозволяє зручно керувати гравцем в грі (Лістинг 3.3).[14]

Щодо стрибка та ухилення, використання 'GetKeyDown' допомагає визначити, коли гравець натиснув відповідну клавiшу. Результат можна зберегти у змінну типу 'bool'. За допомогою базового умовного оператора 'if else', ми можемо викликати відповідні методи або виконувати дії відповідно до натискання клавiші.

Ці підходи дозволяють точно та ефективно керувати рухом гравця та виконувати різні дії за вказаними клавiшами.

```
var inputX:float = Input.GetAxisRaw("Horizontal");  
var jumpInput:bool = Input.GetKeyDown(KeyCode.W);  
var dodgeInput:bool = Input.GetKeyDown(KeyCode.Space);
```

Лістинг.3.3. Input

Для переміщення персонажів у грі використовується компонент 'Rigidbody2D', який відповідає за фізичну поведінку об'єктів. Змінюючи значення 'velocity' цього компонента, ми можемо змінювати швидкість руху об'єкту[15].

Для горизонтального руху, ми використовуємо змінну 'inputX', яку отримали від гравця. Множення цієї змінної на швидкість дозволяє встановити відповідну швидкість руху вліво або вправо. Позитивне значення 'inputX' вказує на рух вправо, а негативне - на рух вліво, що дозволяє коректно виконувати рух у відповідних напрямках (Лістинг 3.4).

Для стрибка ми змінюємо координату 'y' вектора 'velocity'. Тут не потрібно враховувати негативне значення, оскільки стрибок завжди відбувається у позитивному напрямку. Зміна значення 'y' дозволяє персонажу піднятися у повітря.

За допомогою вищезгаданих механік зміни 'velocity' та використання відповідних значень для кожного випадку руху (горизонтальний або вертикальний), ми можемо контролювати рух гравця та інших персонажів у грі.

					КН.КР.Б – 122 – 23 – ПЗ	Арк.
						37
Змн.	Арк.	№ докум.	Підпис	Дата		

```

if (!fight.isBlocking)
{
    rb.velocity = new Vector2(x:inputX * playerClass.speed, rb.velocity.y);
    if (jumpInput && isGrounded() && stats.canJump())
    {
        jumpSFX.Play();
        rb.velocity = new Vector2(rb.velocity.x, y:playerClass.jumpPower);
        stats.spCurrent -= stats.jumpCost;
    }

    if (dodgeInput && CanDodge())
    {
        Dodge();
    }
}

```

Лістинг.3.4. Velocity management

У скрипті PlayerInventory.cs здійснюється управління інвентарем гравця. Інвентар представлений двома списками: `inv` і `amount` (див.Додаток Д).

Список `inv` містить ID предметів, які знаходяться в інвентарі гравця. ID може бути унікальним ідентифікатором або посиланням на конкретний предмет у грі.

Список `amount` відображає кількість кожного предмету, що зберігається в інвентарі.[16] Кількість предметів відповідає елементу зі списку `inv` за відповідним індексом.

Цей скрипт відповідає за додавання, видалення та використання предметів в інвентарі гравця. Він забезпечує можливість підбирати нові предмети, збільшувати їх кількість, використовувати їх для різних дій або взаємодіяти з іншими скриптами у грі.

У методі Update() скрипта PlayerInventory.cs реалізовано використання предметів за допомогою панелі швидкого доступу (Лістинг 3.5). Панель швидкого доступу розташована внизу екрана і містить активний предмет, який гравець може використовувати шляхом натискання клавіші Q.

В коді, використовується функція Input.GetAxis("Mouse ScrollWheel"), яка дозволяє отримати значення прокрутки колеса миші. Це значення використовується для зміни активного предмета на панелі швидкого доступу.

					КН.КР.Б – 122 – 23 – ПЗ	Арк.
						38
Змн.	Арк.	№ докум.	Підпис	Дата		

Коли гравець прокручує колесо миші, виконується перевірка на зміну значення прокрутки. Якщо значення позитивне, то активний предмет змінюється на наступний в списку предметів, якщо значення від'ємне, то активний предмет змінюється на попередній.

Після цього, коли гравець натискає клавішу Q, виконується перевірка активного предмета. Якщо активний предмет існує і може бути використаний, виконується відповідна дія, пов'язана з використанням цього предмета (Лістинг 3.6).

Цей підхід дозволяє гравцю зручно та швидко використовувати предмети з панелі швидкого доступу, змінюючи їх активний стан за допомогою колеса миші та використовуючи їх за допомогою клавіші Q.

```
void Update()
{
    if (GameObject.FindWithTag("MainCamera").GetComponent<CameraFollow>().inPause)
    {
        return;
    }

    if (Input.GetKeyDown(KeyCode.I))
    {
        Debug.Log(playerClass.inv);
    }
    else if (Input.GetKeyDown(KeyCode.Q) && !fight.isAttacking && !fight.isBlocking)
    {
        UseItem(quickCurrent);
    }
    else if (Input.GetAxis("Mouse ScrollWheel") > 0f ) // forward
    {
        ChangeItem(n:1);
        ChangeSprite();
    }
    else if (Input.GetAxis("Mouse ScrollWheel") < 0f ) // backwards
    {
        ChangeItem(n:-1);
        ChangeSprite();
    }
}
```

Лістинг.3.5. PlayerInventory update()

```

public void UseItem(int index)
{
    var tmpID:int = playerClass.inv.IndexOf(index);
    if (playerClass.inv[index] == 2 && playerClass.amount[index] > 0)
    {
        chara.hpCurrent += 25;
        potionsfx.Play();
        playerClass.amount[index]--;
    }
    else if (playerClass.inv[index] == 3 && playerClass.amount[index] > 0)
    {
        var tmppos = new Vector2(posItemUse.position.x, posItemUse.position.y);
        GameObject knife_throw = Instantiate(knife, (Vector3)tmppos, Quaternion.identity);
        knivesfx.Play();
        playerClass.amount[index]--;
    }
    else if (playerClass.inv[index] == 4 && playerClass.amount[index] > 0)
    {
        var tmppos = new Vector2(posItemUse.position.x, posItemUse.position.y);
        GameObject bomb_throw = Instantiate(bomb, (Vector3)tmppos, Quaternion.identity);
        playerClass.amount[index]--;
        bombsfx.Play();
    }
    ChangeSprite();
}

```

Лістинг.3.6. PlayerInventory UseItem()

Давайте розглянемо механіки бою. Гравець має можливість використовувати атаку, блокування, ухилення та парирування. Ці різноманітні механіки дають гравцеві велику свободу у виборі свого стилю гри. Атакуючи, гравець може завдати шкоди ворогам та убити їх. Захищаючись блокуванням, гравець може знищити ворожі атаки. Ухиляючись від ворогів, гравець може потрапити за їх спину та нанести шкоду. Парирування - це ризикований стиль, але з великою винагородою. Якщо гравець успішно парирує атаку ворога, він може нанести контратаку, завдаючи подвійну шкоду. За все це відповідає скрипт FightBehaviour.cs (див.Додаток В).

Для реалізації затримки атаки було використано coroutine - затримка виконання коду. (Лістинг 3.7) Це дозволяє зробити так, щоб нанесення урону були синхронізоване з анімацією атаки. Також використовується `OverlapCircleAll`, який зберігає в собі всі об'єкти в певному радіусі перед

					КН.КР.Б – 122 – 23 – ПЗ	Арк.
						40
Змн.	Арк.	№ докум.	Підпис	Дата		

гравцем в слою ворогів, що дає можливість наносити удари по декільком ворогам одразу.

Додатково, для визначення ворогів, з якими гравець взаємодіє, використовується метод `OverlapCircleAll`. Цей метод шукає всі об'єкти в певному радіусі перед гравцем, які належать до шару ворогів. Це дає можливість гравцеві наносити удари одразу по декільком ворогам, які знаходяться у цьому радіусі[17].

```
public IEnumerator attack()
{
    attacksfx.Play();
    animCont.SetTrigger(name: "Attack");
    isAttacking = true;
    canAttack = false;
    yield return new WaitForSeconds(0.25f);
    timeBtwAttack = startTimeBtwAttack;
    Collider2D[] enemyToDamage = Physics2D.OverlapCircleAll((Vector2)attackPos.transform.position, attackRange, (int)enemyMask);
    for (int i = 0; i < enemyToDamage.Length; i++)
    {
        if (!enemyToDamage[i].GetComponent<EnemyClass>().isDead)
        {
            enemyToDamage[i].GetComponent<EnemyClass>().TakeDamage(playerClass.damage);
        }
    }
    isAttacking = false;
    StartCoroutine(routine: AttackCD());
}
```

Лістинг.3.7. Attack script

Блокування та парирування реалізовано шляхом зміни анімацій та змінних стану гравця. Ці механіки дозволяють гравцеві захищатися від ворожих атак та уникати отримання ушкоджень. Процес блокування включає зміну анімаційного ефекту та встановлення відповідної змінної стану, що вказує, що гравець знаходиться в режимі блокування. Парирування також передбачає зміну анімацій та змінних стану, щоб гравець відтворював відповідну анімацію ухилення та був в стані парирування.

Далі: менеджмент показників гравця, такі як здоров'я та стаміна. Це критичні показники для гравця, оскільки якщо в нього закінчиться здоров'я - він помре, а стаміна - не зможе ухилитися, стрибати та блокувати (див.Додаток Г).

					КН.КР.Б – 122 – 23 – ПЗ	Арк.
						41
Змн.	Арк.	№ докум.	Підпис	Дата		

Механіка смерті гравця в грі, інспірованої жанром Souls-like, є важливою частиною геймплею. При смерті гравця відбуваються деякі наслідки, які додають виклику та реалізму в грі[18].

Одним з наслідків є втрата половини накопичених монет гравцем. Це стимулює гравця бути обережним та розсудливим, оскільки втрата монет може вплинути на його прогрес і можливості покращити персонажа.

Після смерті, екран затемнюється та з'являється напис про смерть, що додає настрою гри і підкреслює важливість виживання. Гравець переміщується до останнього святилища, де він відпочивав, що стає моментом відпочинку та підготовки до наступної спроби.

Додатково, всі вороги, яких гравець вже здолав, спавняться знову після смерті. Це створює виклик та можливість повторного зіткнення з вже відомими ворогами, дозволяючи гравцеві вдосконалювати свої навички та стратегію (Лістинг 3.8).

Усі ці елементи механіки смерті гравця додають грі виклик, напруження та посилюють відчуття досягнення та прогресу під час успішного програвання гри.

```
public IEnumerator Death()
{
    deathCouroutine = true;
    isDead = true;
    animCont.SetBool(name: "isDead", value: true);
    cam.GetComponent<CameraFollow>().YouDied.SetActive(true);
    camSRC.BlackScreenTransparency(a:1);
    animCont.SetBool(name: "isDead", value: false);
    yield return new WaitForSeconds(3f);
    assignStats();
    player.pieces /= 2;
    gameObject.transform.position = RespawnPoint.transform.position;
    cam.GetComponent<CameraFollow>().YouDied.SetActive(false);

    levelCreateSRC.GetComponent<LevelCreate>().foreachEnemyFindDestroy();
    levelCreateSRC.GetComponent<LevelCreate>().foreachCycle(levelCreateSRC.GetComponent<LevelCreate>().Enemy, levelCreateSRC.g
    isDead = false;
    deathCouroutine = false;
    camSRC.BlackScreenTransparency(a:0);
}
```

Лістинг.3.8. Death()

Отримання урону гравцем є важливим аспектом геймплею. Коли гравець отримує удар від ворога, відбувається передача двох змінних у метод: нанесений урон (dmg) та тип атаки (type). Типи атак визначаються різними

					КН.КР.Б – 122 – 23 – ПЗ	Арк.
						42
Змн.	Арк.	№ докум.	Підпис	Дата		

параметрами, але ми зосередимося на їх обробці пізніше, коли розглядатимемо скрипт поведінки ворогів. Наразі ми зосередимося на основній логіці обробки отриманого урону (Лістинг.3.9)[19].

За допомогою умовної конструкції if-else ми перевіряємо тип атаки та стан гравця, щоб визначити, чи потрібно нанести урон гравцеві. Залежно від різних умов, можуть бути виконані різні дії, такі як зменшення здоров'я гравця, виклик анімацій отримання удару, зміна стану гравця та інші додаткові ефекти. Це дозволяє реалістично відтворити взаємодію між гравцем та ворогами, а також забезпечує можливість реалізувати різні механіки, такі як блокування, ухилення або парирування ударів.

Використання умовної конструкції if-else дозволяє гнучко контролювати реакцію гравця на отриманий урон та визначати подальші наслідки цього удару. В залежності від типу атаки та стану гравця можуть відбуватися різні події та зміни в геймплеї, що робить бій більш цікавим та варіативним для гравця.

Ця логіка дозволяє контролювати взаємодію гравця з ворогами, реалістично відтворювати вплив ударів та стимулює гравця шукати оптимальні стратегії для виживання та перемоги[20].

```
public bool takeDmg(float dmg, int type)
{
    if(GameObject.FindWithTag("Player").GetComponent<PlayerMovement>().isDashing && (type == 0 || type == 1)) // if dodging && onlyDodge attack or default attack
        return true;

    if(GameObject.FindWithTag("Player").GetComponent<PlayerMovement>().isDashing && type == 2) // if dodging && onlyParry attack
    {
        hpCurrent -= dmg;
        return false;
    }

    if (fight.isBlocking && type == 0 && !fight.isParry) // if block default attack
    {
        spCurrent -= dmg;
        blocksfx.Play();
        return false;
    }

    if (fight.isParry && (type == 0 || type == 2)) // if parry default or parryOnly attack
    {
        parrysfx.Play();
        StopCoroutine(routine:fight.CanParry());
        return true;
    }

    animCont.SetTrigger(name:"Hurt"); // others
    hpCurrent -= dmg;
    hitSFX.Play();
    return false;
}
```

Лістинг.3.9. TakeDMG()

3.2 Розробка інших скриптів та механік

Механіка ворогів була реалізовано аналогічно як і гравця, тільки методи якого знаходяться в самому класі, а в скрипті поведінки буде лише сама поведінка та виклик необхідних методів (див.додаток Ж).

Метод атаки ворога реалізовано з можливістю завдавати як звичайні, так і спеціальні атаки. Використовуючи функцію `Random.Next`, ми генеруємо випадкове число в діапазоні від 0 до 100 і перевіряємо його значення. Залежно від цього значення, ворог може виконати спеціальну атаку. Якщо "кидок костей" успішний, активується індикатор, який показує гравцеві, як саме йому необхідно реагувати на цю атаку. Це додає елемент випадковості до бою та змушує гравця адаптуватися до ворога (Лістинг 3.10).

Сама атака ворога реалізована аналогічно до атаки гравця: за допомогою функції `OverlapCircleAll` ми отримуємо масив об'єктів в шарі гравця, а потім викликаємо функцію `takeDmg` гравця, передаючи в неї значення урону та типу атаки. Атаки можуть бути звичайними, від яких можна лише ухилитися, або парируються.

Після завершення атаки ми запускаємо корутину, яка створює паузу між атаками, щоб забезпечити відповідний ритм бою та можливість гравцю реагувати на наступну атаку ворога.

Цей підхід додає додаткову випадковість та непередбачуваність до бою, роблячи його більш захоплюючим та вимагаючим від гравця стратегічного мислення та швидкої реакції.

					КН.КР.Б – 122 – 23 – ПЗ	Арк.
						44
Змн.	Арк.	№ докум.	Підпис	Дата		

```

public IEnumerator Attack()
{
    // 0 - default attack
    // 1 - only dodge
    // 2 - only parry
    isAttacking = true;
    var dice = rng.Next(0, 100);
    if (canOnlyDodge && 0 <= dice && dice <= 10) // onlyDodge
    {
        animController.SetTrigger(name: "attackSpecial");
        // Indicator HERE
        indicatorDodge.SetActive(true);
        var tmpColor = indicatorDodge.GetComponent<SpriteRenderer>().color;
        tmpColor.a = 100f;
        indicatorDodge.GetComponent<SpriteRenderer>().color = tmpColor;
        yield return new WaitForSeconds(attackDelay - 0.2f);
        // Indicator FLASH
        tmpColor.a = 255f;
        indicatorDodge.GetComponent<SpriteRenderer>().color = tmpColor;
        yield return new WaitForSeconds(0.2f);
        Collider2D[] colls = Physics2D.OverlapCircleAll((Vector2) attackPoss.transform.position, range, (int) playerLayer);
        foreach (var i:Collider2D in colls)
        {
            if (i.GetComponent<PlayerChara>().takeDmg(dmg, type:1))
            {
                StartCoroutine(routine: StunCd());
            }
        }
        indicatorDodge.SetActive(false);
    }
}

```

Лістинг.3.10. EnemyClass Attack()

У скрипті EnemyBehavior ми відповідаємо за контроль поведінки ворога. У методі Start ми отримуємо необхідні компоненти та об'єкти, необхідні для правильної роботи ворога (див. Додаток 3).

У методі Update ми визначаємо відстань до гравця за допомогою функції Vector2.Distance і перевіряємо, чи ця відстань є достатньою для руху ворога. Якщо відстань менша або рівна певному значенню, ворог починає рухатися в бік гравця (див. Додаток 3 ряд. 19-43).

Коли ворог підійшов на достатню відстань для атаки, ми викликаємо метод атаки, щоб ворог завдавав шкоду гравцеві.

Цей підхід дозволяє ворогові активно рухатися в напрямку гравця і атакувати його, коли відстань є оптимальною. Це створює реалістичне враження поведінки ворогів і робить битви з ними цікавими та вимагаючими від гравця.

					КН.КР.Б – 122 – 23 – ПЗ	Арк.
						45
Змн.	Арк.	№ докум.	Підпис	Дата		

Камера в грі реалізована таким чином, що плавно слідує за рухами гравця, що додає більшу іммерсію та приємніші візуальні враження. Для досягнення цього ефекту використовується функція `Vector3.SmoothDamp`.

`Vector3.SmoothDamp` - це функція, яка виконує плавний перехід від одного значення до іншого протягом певного часу. В даному випадку, ми використовуємо цю функцію для плавного руху камери за гравцем.

Параметри функції включають поточне положення камери, цільове положення (положення гравця), швидкість руху камери та час, за який має бути виконаний перехід. За допомогою цих параметрів функція автоматично обчислює нове положення камери на кожному кадрі, що дозволяє плавно слідувати за гравцем навіть при зміні його руху (Лістинг 3.11).

Цей підхід додає динаміки та комфорту гравцю, оскільки камера не робить різких рухів і приємно адаптується до рухів.

```
private void Update()
{
    if (GameObject.FindWithTag("Player").GetComponent<PlayerChara>().isDead)
    {
        noDamping = true;
        return;
    }
    else if(noDamping && cam.transform.position != target.position)
    {
        noDamping = false;
        transform.position = target.position;
    }
    Vector3 movePos = target.position + offset;
    transform.position = Vector3.SmoothDamp(current:transform.position, target:movePos, ref velocity, damping);
    if (Input.GetKey(KeyCode.P) && !inPause)
    {
        PauseStart();
    }
}
```

Лістинг.3.11.Camera Damp

Затронимо збереження даних. Збереження даних у вигляді .txt файлів є зручним способом організації та збереження ігрової інформації. Кожен файл може містити конкретний тип предмету, його назву та іншу необхідну інформацію (Лістинг 3.12).

Використання .txt файлів для збереження даних має декілька переваг. По-перше, це зручність редагування. Завдяки такому формату даних, ви можете

					КН.КР.Б – 122 – 23 – ПЗ	Арк.
						46
Змн.	Арк.	№ докум.	Підпис	Дата		

легко змінювати вміст файлів, додавати нову інформацію чи видаляти стару. Це дає вам гнучкість у налаштуванні гри та зміні ігрових параметрів без необхідності перекомпіляції чи зміни вихідного коду.

Крім того, збереження даних у .txt файлах дозволяє легко розширювати ігровий контент. Ви можете створювати нові файли для нових типів предметів або додавати нову інформацію до існуючих файлів, що дозволяє вам швидко вводити зміни та оновлення в гру.

Для роботи з файлами ви можете використовувати вбудовані функції та класи мови програмування, такі як читання/запис до файлу, розбиття на рядки та обробка даних. При взаємодії з гравцем у методі діалогу в грі, ви можете читати вміст файлу, знаходити потрібну інформацію та використовувати її для створення діалогових опцій та інших елементів гри.

Загалом, використання .txt файлів для збереження даних є зручним та гнучким підходом, який дозволяє вам легко керувати ігровою інформацією та розширювати її в майбутньому.

itemsID.txt	
1	0:shard
2	1:flower
3	2:flask
4	3:knife
5	4:bomb
6	5:document
7	

Лістинг.3.12.зберігання даних

Далі - реалізація рівні. Оскільки гра наслідує ідеї Souls-like ігор, то просто поставити Prefab ворога на сцені нам не підходить. Також у гри є система збереження, тому перед початком гри потрібно завантажити необхідну інформацію і вже від неї працювати. Скрипт LevelCreate саме цим і займається.

У скрипті LevelCreate ми реалізуємо завантаження необхідних об'єктів та створення ворогів на сцені. На початку гри ми завантажуюмо всі необхідні Prefab'и та отримуємо масив позицій, де вороги мають бути розміщені. Далі ми

					КН.КР.Б – 122 – 23 – ПЗ	Арк.
						47
Змн.	Арк.	№ докум.	Підпис	Дата		

проходимо цей масив та за допомогою функції Instantiate створюємо ворогів, використовуючи відповідні Prefab'и. Цей підхід дозволяє нам гнучко контролювати розміщення ворогів на різних рівнях та забезпечує можливість швидко змінювати їх кількість та типи (Лістинг 3.13).

```
void Awake()
{
    Player = GameObject.FindWithTag("Player");
    Enemy = Resources.Load(path: "Enemy") as GameObject;

    MushRoom = Resources.Load(path: "MushRoomEnemy") as GameObject;
    Goblin = Resources.Load(path: "GoblinEnemy") as GameObject;
    Skeleton = Resources.Load(path: "SkeletonEnemy") as GameObject;
    FlyingEye = Resources.Load(path: "EyeEnemy") as GameObject;

    PlayerPos = GameObject.FindWithTag("playerPos");

    shrines = GameObject.FindGameObjectsWithTag("shrine");

    if (SceneManager.GetActiveScene().name != "lvl_hub")
    {
        foreachCycle(MushRoom, MushRoomPos);
        foreachCycle(Goblin, GoblinPos);
        foreachCycle(Skeleton, SkeletonPos);
        foreachCycle(FlyingEye, FlyingEyePos);
    }

    if (PlayerPrefs.GetInt(key: "newGame") != 0) //load game[...]
    else[...] //new game
}
```

Лістинг.3.13.LevelCreate Awake()

Механіка збереження реалізована за допомогою класу SaveClass, який має таку саму структуру, як і клас PlayerClass. Це дозволяє нам легко передавати інформацію між ними. SaveClass містить змінні, які потрібно зберегти, а також конструктор, який дозволяє створювати об'єкти цього класу з необхідними значеннями. Такий підхід дозволяє нам зручно та ефективно зберігати та відновлювати ігровий прогрес гравця (Лістинг 3.14).

```

public class SaveClass
{
    PlayerStats

    ProgresVars

    Frequently called 1 usage Tymur
    public SaveClass(PlayerClass player){...}
}

```

Лістинг.3.14. SaveClass.cs

Файл SaveSystem.cs відповідає за завантаження та збереження гравцевого прогресу. В ньому містяться два методи: SavePlayer та LoadPlayer, які відповідають за збереження та завантаження гравцевих даних відповідно.

Перш ніж зберігати або завантажувати дані, ми визначаємо шлях до файлу збереження. Це робиться за допомогою змінної "path", яка містить Application.persistentDataPath. Використання цього шляху гарантує, що файли будуть знаходитися в одному місці на будь-якому пристрої, уникнувши проблем з пошуком файлів.

Потім ми створюємо новий BinaryFormatter та FileStream. Це дозволяє нам серіалізувати (для збереження) або десеріалізувати (для завантаження) дані у форматі бінарного коду. Це забезпечує надійне збереження і недоступність для редагування даних гравця.

У методі SavePlayer ми серіалізуємо об'єкт класу SaveClass та зберігаємо його у файл. Аналогічно, у методі LoadPlayer ми десеріалізуємо дані з файлу та повертаємо об'єкт класу SaveClass.

Таким чином, завдяки цьому скрипту ми можемо зручно та безпечно зберігати та завантажувати ігровий прогрес гравця (Лістинг 3.15).

					КН.КР.Б – 122 – 23 – ПЗ	Арк.
						49
Змн.	Арк.	№ докум.	Підпис	Дата		

```

public class SaveSystem : MonoBehaviour{

    ⚡ Frequently called  📄 2 usages  👤 Tymur
    public static void SavePlayer(PlayerClass player)
    {
        Debug.Log(message: "SavePlayer");
        BinaryFormatter formatter = new BinaryFormatter();
        string path = Application.persistentDataPath + "/player.json";
        FileStream stream = new FileStream(path, FileMode.Create);

        SaveClass data = new SaveClass(player);

        formatter.Serialize(stream, data);
        stream.Close();
    }

    📄 2 usages  👤 Tymur
    public static SaveClass LoadPlayer()
    {
        string path = Application.persistentDataPath + "/player.json";
        if (File.Exists(path))
        {
            BinaryFormatter formatter = new BinaryFormatter();
            FileStream stream = new FileStream(path, FileMode.Open);

            SaveClass data = formatter.Deserialize(stream) as SaveClass;

            stream.Close();

            return data;
        }
        else
        {
            Debug.LogError(message: "Save file not found in " + path);
            return null;
        }
    }
}

```

Лістинг.3.15. SaveSystem.cs

					КН.КР.Б – 122 – 23 – ПЗ	Арк.
						50
Змн.	Арк.	№ докум.	Підпис	Дата		

Скрипт DialogueSRC.cs відповідає за реалізацію механіки діалогів у грі. У цьому скрипті особлива увага приділяється роботі з текстовим файлом та отриманню необхідної інформації з нього.

Спочатку ми відкриваємо необхідний файл та конвертуємо його вміст у змінну типу string. Це робиться для того, щоб мати можливість працювати з текстом та здійснювати пошук необхідних рядків.

Для пошуку конкретного рядка тексту за його ID, ми використовуємо метод GetBetween. Цей метод дозволяє отримати текст, який знаходиться між ID фрази та символом переносу на новий рядок (\r). Таким чином, ми можемо отримати необхідну змінну з даними без необхідності ручного редагування тексту (Лістинг 3.16).

Ця механіка дозволяє нам зручно та ефективно отримувати необхідну інформацію з текстового файлу та використовувати її для побудови діалогів у грі.:

```
public static string getBetween(string file, string strStart, string strEnd)
{
    if (file.Contains(strStart) && file.Contains(strEnd))
    {
        int Start, End;
        Start = file.IndexOf(strStart, startIndex: 0) + strStart.Length;
        End = file.IndexOf(strEnd, Start);
        return file.Substring(Start, length: End - Start);
    }

    return "";
}
```

Лістинг.3.16. DialogueSRC getBetween()

3.3 Розробка інших аспектів гри

Окрім реалізації механік та коду гри, було відповідально за звуковий дизайн, сюжет та вигляд екранів завантаження. Кожен з цих аспектів можна коротко описати:

- Звуковий дизайн: було створено та інтегровано звукові ефекти, музику та атмосферні звуки у гру. Звукові ефекти слугували для підкреслення дій

					КН.КР.Б – 122 – 23 – ПЗ	Арк.
						51
Змн.	Арк.	№ докум.	Підпис	Дата		

гравця, створення атмосфери навколишнього середовища та особливих подій. Музика додавала настрою та підтримувала емоційну атмосферу геймплею.

Реалізацію звуків та музики не потребує детального опису, оскільки працює завдяки одному коду рядку - `AudioSource.Play()` (Лістинг.3.17). Завдяки їй ми вмикаємо джерело звуку і воно програватиметься один раз - саме те, що нам і потрібно.

```
attacksfx.Play();
```

Лістинг.3.17. приклад програвання звуку та музики.

- Сюжет: було розроблено та впроваджено сюжетні елементи, діалоги персонажів та оповідання. Це включало створення захопливої історії, характерів персонажів, ключових моментів сюжету та взаємодію гравця з ними. Метою було створення захоплюючого та занурюючого ігрового досвіду.

- Екрани завантаження: було зайнято створенням та оформленням екранів завантаження, які з'являлися перед початком гри або при переході між рівнями. Основна мета полягала в створенні привабливого та професійного вигляду, який відповідав загальному стилю гри та привертав увагу гравця.

Ми використовуємо асинхронне завантаження для виконання процесу завантаження. Спочатку ми активуємо екран завантаження, а потім, використовуючи `AsyncOperation`, ми отримуємо стан завантаження сцени та візуально відображаємо його у вигляді колеса завантаження.

Коли сцена повністю завантажена і готова до відображення, ми повідомляємо гравця і просимо натиснути клавішу `Enter` для активації цієї сцени (Лістинг.3.18).

					КН.КР.Б – 122 – 23 – ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		52

```

private IEnumerator LoadSceneAsync(string sceneName)
{
    loadingScreen.SetActive(true);
    input.SetActive(false);

    yield return new WaitForSeconds(1f); // Затримка на 1 секунду

    asyncOperation = SceneManager.LoadSceneAsync(sceneName);
    asyncOperation.allowSceneActivation = false;

    while (!asyncOperation.isDone)
    {
        float progress = Mathf.Clamp01(asyncOperation.progress / 0.9f);
        progressBar.fillAmount = progress;
        if (progress >= 1.0f)
        {
            isLoadingComplete = true;
            input.SetActive(true);
        }

        yield return null;
    }

    ActivateScene();
}

```

Лістинг.3.18. AsyncLoad

Кожен з цих аспектів - звуковий дизайн, сюжет та екрани завантаження - є важливою складовою геймплею та допомагає створити цілісне та захоплююче враження від гри.

Висновки до третього розділу

В цьому підрозділі ми обговорили розробку програмного коду для реалізації різних аспектів гри. Ми розглянули роботу зі скриптами, які керують поведінкою ворогів та камери, а також з UI елементами. Ми також поговорили про механізм збереження даних та реалізацію рівнів.

У цьому розділі ми дізналися, як використовувати різні класи та методи для створення ворожих персонажів, як керувати камерою для плавного слідкування за гравцем, як працювати з UI елементами, включаючи їх вимкнення та активацію, і як зберігати та завантажувати дані гри.

Крім того, ми розглянули роль звукового дизайну, сюжету та екранів завантаження у грі. Звуковий дизайн створює атмосферу та підкреслює дії гравця, сюжет розкриває історію та персонажів, а екрани завантаження додають професійного вигляду грі перед початком геймплею.

					КН.КР.Б – 122 – 23 – ПЗ	Арк.
						53
Змн.	Арк.	№ докум.	Підпис	Дата		

Цей розділ дав нам уявлення про те, як розробляти програмний код для різних аспектів гри та як їх взаємодіяти між собою. Детальний розгляд коду та опис функціональності допоможуть нам зрозуміти, як гра працює та які можливості вона надає.

					КН.КР.Б – 122 – 23 – ПЗ	Арк.
						54
Змн.	Арк.	№ докум.	Підпис	Дата		

ВИСНОВКИ

Під час роботи над кваліфікаційною роботою було створено і реалізовано ігровий проект у жанрі сайдскролер екшн з елементами платформінгу. Для успішної реалізації цього проекту були виконані послідовні кроки, які були попередньо сплановані.

Початково був проведений аналіз ігрового ринку та конкурентів, що допомогло визначитися з аспектом гри, жанром, механіками та естетикою. Було продумано та створено маршрути для різних рівнів гри. Також були досліджені різні типи інтерфейсу користувача, що допомогло вирішити питання, що виникали при створенні UI для власного проекту.

Далі, була проведена аналітична перевірка функціональності самої гри, переконавшись, що запланований маршрут може бути успішно пройдений гравцем. Було перевірено коректність прокладення маршруту та можливість гравця досягти своєї мети.

Були реалізовані наступні системи:

1. Система переміщення: механіка переміщення відчувається реактивною та енергійною.
2. Система бойових механік: бойова система належним чином реалізована та ставить перед гравцем виклик, вимагаючи прийняття важливих рішень.
3. Система інвентаризації: гравець має можливість збирати предмети на локаціях та використовувати їх для полегшення проходження гри.
4. Система діалогів з персонажами: діалоги виконані у стилі візуальних новел, задовольняючи потреби гравця.
5. Механіки, які не входять в жодну з категорій: інші механіки, спрямовані на різні аспекти гри та зручність, працюють бездоганно.

Цей етап дозволив переконатися, що проект розвивається відповідно до задуму та всі його основні функціональні елементи працюють бездоганно.

					КН.КР.Б – 122 – 23 – ПЗ	Арк.
						55
Змн.	Арк.	№ докум.	Підпис	Дата		

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Steam [Електронний ресурс] – Режим доступу до ресурсу:
https://store.steampowered.com/app/283640/Salt_and_Sanctuary/
2. Steam [Електронний ресурс] – Режим доступу до ресурсу:
<https://store.steampowered.com/app/268910/Cuphead/>
3. Steam [Електронний ресурс] – Режим доступу до ресурсу:
https://store.steampowered.com/app/367520/Hollow_Knight/
4. Unity in Action: Multiplatform Game Development in C# / Joe Hocking, 2015.
- 53с.
5. Mastering Unity 2D Game Development / Simon Jackson, 2014. – 121с.
6. Unity Game Development Cookbook / Michael G. Flores, 2019. – 402с.
7. Learning C# by Developing Games with Unity / Harrison Ferrone, 2022. – 99с.
8. Unity Game Optimization / Chris Dickinson, 2019. – 188с.
9. Unity in Action: Multiplatform Game Development in C# / Joe Hocking, 2019.
– 98с.
10. Unity 2018 Cookbook / Matt Smith, 2018. – 10с.
11. Документація Unity [Електронний ресурс] – Режим доступу до ресурсу:
<https://docs.unity3d.com>
12. Документація Unity Libraries [Електронний ресурс] – Режим доступу до
ресурсу: <https://docs.unity3d.com/Manual/>
13. Документація Unity for Beginners [Електронний ресурс] – Режим доступу
до ресурсу: <https://unity.com/how-to/beginner/5-unity-tutorials-new-game-developers>
14. Unity Tutorials [Електронний ресурс] - Режим доступу до ресурсу:
<https://www.tutorialspoint.com/unity/index.htm>
15. ItProger [Електронний ресурс] - Режим доступу до ресурсу:
<https://itproger.com/ua/course/unity-csharp>
16. Stack Overflow [Електронний ресурс] - Режим доступу до ресурсу:
<https://stackoverflow.com/>

					КН.КР.Б – 122 – 23 – ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		56

17. DotNetPearls [Електронний ресурс] - Режим доступу до ресурсу:
<https://www.dotnetperls.com/between-before-after>
18. C# Documentation [Електронний ресурс] - Режим доступу до ресурсу:
<https://learn.microsoft.com/en-us/dotnet/csharp/>
19. Souls-Like Games [Електронний ресурс] - Режим доступу до ресурсу:
<https://en.wikipedia.org/wiki/Soulslike>
20. Unity discussions [Електронний ресурс] - Режим доступу до ресурсу:
<https://discussions.unity.com/>

					КН.КР.Б – 122 – 23 – ПЗ	Арк.
						57
Змн.	Арк.	№ докум.	Підпис	Дата		

ДОДАТОК

					КН.КР.Б – 122 – 23 – ПЗ	Арк.
						58
Змн.	Арк.	№ докум.	Підпис	Дата		

```
using System;
using System.Collections;
using System.Collections.Generic;
using Unity.Mathematics;
using UnityEditorInternal;
using UnityEngine;
using UnityEngine.SceneManagement;

public class PlayerClass : MonoBehaviour
{
    #region PlayerChara
    public float hpMax = 100f;
    public float spMax = 50f;
    public int piecesToGrade = 10;
    public int pieces = 0;
    public int level = 1;
    #endregion

    #region FightBehaviour
    public float damage = 10;
    #endregion

    #region Movement
    public float speed = 8f;
    public float jumpPower = 16f;
    public float spintMult = 1.5f;
    #endregion

    #region Inventory
    public int flaskMax = 2;
    public List<int> inv;
    public List<int> amount;
    #endregion

    #region Progress
    public bool lvl_0_open = true;
    public bool lvl_hub_open = true;
    public bool lvl_1_open = false;
    public bool lvl_2_open = false;

    public bool lvl_0_completed = false;
    public bool lvl_1_completed = false;
    public bool lvl_2_completed = false;
```

```

//smith
public bool que_01_00_smith_open= true;
public bool que_01_00_smith_completed = false;
public bool que_01_01_smith_open = false;
public bool que_01_01_smith_completed = false;

//reaper
public bool que_01_00_reaper_open = true;
public bool que_01_00_reaper_completed = false;
public bool que_01_01_reaper_open = false;
public bool que_01_01_reaper_completed = false;

//alchemist
public bool que_01_00_alchemist_open = true;
public bool que_01_00_alchemist_completed = false;
public bool que_01_01_alchemist_open = false;
public bool que_01_01_alchemist_completed = false;

//lvl_0_items
public bool lvl_0_shard_picked = false;
public bool lvl_0_flower_picked = false;
public bool lvl_0_scroll_picked = false;

//lvl_1_items
public bool lvl_1_shard_picked = false;
public bool lvl_1_flower_picked = false;
public bool lvl_1_scroll_picked = false;
#endregion
public string currScene;

public Vector3 respPos;

public PlayerClass(PlayerClass data)
{
hpMax = data.hpMax;
spMax = data.spMax;
level = data.level;
damage = data.damage;
piecesToGrade = data.piecesToGrade;
pieces = data.pieces;
currScene = data.updateScene();
flaskMax = data.flaskMax;
inv = data.inv;
amount = data.amount;
lvl_0_open = data.lvl_0_open;

```

```

lvl_hub_open = data.lvl_hub_open;
lvl_1_open = data.lvl_1_open;
lvl_2_open = data.lvl_2_open;
lvl_0_completed = data.lvl_0_completed;
lvl_1_completed = data.lvl_1_completed;
lvl_2_completed = data.lvl_2_completed;

que_01_00_smith_open = data.que_01_00_smith_open;
que_01_00_smith_completed = data.que_01_00_smith_completed;
que_01_01_smith_open = data.que_01_01_smith_open;
que_01_01_smith_completed = data.que_01_01_smith_completed;

que_01_00_reaper_open = data.que_01_00_reaper_open;
que_01_00_reaper_completed = data.que_01_00_reaper_completed;
que_01_01_reaper_open = data.que_01_01_reaper_open;
que_01_01_reaper_completed = data.que_01_01_reaper_completed;

que_01_00_alchemist_open = data.que_01_00_alchemist_open;
que_01_00_alchemist_completed = data.que_01_00_alchemist_completed;
que_01_01_alchemist_open = data.que_01_01_alchemist_open;
que_01_01_alchemist_completed = data.que_01_01_alchemist_completed;

lvl_0_shard_picked = data.lvl_0_shard_picked;
lvl_0_flower_picked = data.lvl_0_flower_picked;
lvl_0_scroll_picked = data.lvl_0_scroll_picked;

lvl_1_shard_picked = data.lvl_1_shard_picked;
lvl_1_flower_picked = data.lvl_1_flower_picked;
lvl_1_scroll_picked = data.lvl_1_scroll_picked;
}

public string updateScene()
{
    currScene = SceneManager.GetActiveScene().name;
    return currScene;
}

public void GetPos()
{
    var tmpPlayer = GameObject.FindWithTag("Player");
    respPos = new
Vector3(transform.position.x,transform.position.y,transform.position.z);
}
}

```

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

public class PlayerMovement : MonoBehaviour
{
    public Animator animator;
    private SpriteRenderer sprite;
    #region Variables
    private float horizontal;
    public float faceDir;

    [SerializeField] public Rigidbody2D rb;
    [SerializeField] private GameObject groundCheck;
    [SerializeField] private GameObject wallCheck;
    [SerializeField] private LayerMask groundLayer;

    [SerializeField] private float dashingVelocity;
    [SerializeField] private float dashingTime;
    public Vector2 dashingDir;
    public float dashForce = 5f;
    public float dashCost = 25f;
    public float dashCD = 1f;
    public float dashCDcurr = 0f;
    private float lastDodgeTime = -1f;
    public bool isDashing;
    private bool _canDash = true;
    #endregion

    public CapsuleCollider2D coll;

    public GameObject player;
    [SerializeField] private PlayerChara stats;
    [SerializeField] private FightBehaviour fight;
    public PlayerClass playerClass;
    public GameObject interestPoint;
    public AudioSource dashSFX;
    public AudioSource jumpSFX;

    private void Start()
    {
        coll = GetComponent<CapsuleCollider2D>();
    }
}
```

```

player = GameObject.FindWithTag("Player");
playerClass = GetComponent<PlayerClass>();
sprite = GetComponent<SpriteRenderer>();
animator = GetComponent<Animator>();
rb = GetComponent<Rigidbody2D>();
stats = GetComponent<PlayerChara>();
fight = GetComponent<FightBehaviour>();
groundCheck = GameObject.Find("groundCheck");
}

void Update()
{
    if (stats.isDead || stats.isStunned || stats.inDialogue || fight.isParry ||
fight.isAttacking )
    {
        rb.velocity = Vector2.zero;
        return;
    }

    if
(GameObject.FindWithTag("MainCamera").GetComponent<CameraFollow>().inPau
se)
    {
        return;
    }

    if (isDashing && AtWall())
    {
        rb.velocity = Vector2.zero;
        return;
    }

    if(isDashing)
        return;

    var inputX = Input.GetAxisRaw("Horizontal");
    var jumpInput = Input.GetKeyDown(KeyCode.W);
    var dodgeInput = Input.GetKeyDown(KeyCode.Space);

    if (fight.isBlocking)
    {
        rb.velocity = Vector2.zero;
        animator.SetFloat("Speed", 0);
        return;
    }
}

```

```

animator.SetFloat("Speed", Math.Abs(inputX));

if (inputX > 0)
{
    player.transform.localScale = new Vector3(6,6,1);
}
else if (inputX < 0)
{
    player.transform.localScale = new Vector3(-6,6,1);
}

if (!fight.isBlocking)
{
    rb.velocity = new Vector2(inputX * playerClass.speed, rb.velocity.y);
    if (jumpInput && isGrounded() && stats.canJump())
    {
        jumpSFX.Play();
        rb.velocity = new Vector2(rb.velocity.x, playerClass.jumpPower);
        stats.spCurrent -= stats.jumpCost;
    }

    if (dodgeInput && CanDodge())
    {
        Dodge();
    }

    if (isGrounded())
    {
        _canDash = true;
        animator.SetBool("isJump", false);
    }
    else
    {
        animator.SetBool("isJump", true);
    }

    if (!fight.isFighting)
    {
        if (inputX == 1f || inputX == -1f)
        {
            faceDir = inputX;
        }
    }
}

```

```

    }

    private bool CanDodge()
    {
        return Time.time - lastDodgeTime > dashCD && stats.spCurrent >= dashCost
        && rb.velocity != Vector2.zero && !isDashing;
    }

    private void Dodge()
    {
        dashSFX.Play();
        lastDodgeTime = Time.time;
        stats.spCurrent -= dashCost;
        isDashing = true;
        coll.isTrigger = true;
        rb.constraints = RigidbodyConstraints2D.FreezePositionY;
        rb.AddForce(FaceDirection() * dashForce, ForceMode2D.Impulse);
        animator.SetBool("isDash", true);
        StartCoroutine(StopDashing());
    }

    private IEnumerator StopDashing()
    {
        yield return new WaitForSeconds(dashingTime);
        isDashing = false;
        animator.SetBool("isDash", false);
        coll.isTrigger = false;
        rb.constraints &= ~RigidbodyConstraints2D.FreezePositionY;
        rb.constraints = RigidbodyConstraints2D.FreezeRotation;
    }

    public bool isGrounded()
    {
        return Physics2D.OverlapCircle(groundCheck.transform.position,
0.5f,groundLayer);
    }

    public bool AtWall()
    {
        return Physics2D.OverlapCircle(wallCheck.transform.position, 0.1f,
groundLayer);
    }

    public Vector2 FaceDirection()
    {

```

```
    if (faceDir == 1)
    {
        return Vector2.right;
    }
    else
    {
        return Vector2.left;
    }
}
```

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

public class FightBehaviour : MonoBehaviour
{
    [SerializeField] private PlayerMovement move;
    public PlayerClass playerClass;
    private PlayerChara chara;
    private LayerMask enemyMask;
    private float rayDistanse = 5f;
    public bool isFighting = false;
    public bool isAttacking = false;
    public bool isBlocking = false;
    public bool isParry = false;
    public bool isRecentlyHit = false;
    private float unHitTime = 3f;
    private float startUnHitTime = 0f;
    public GameObject cam;
    private float timeBtwAttack;
    private float startTimeBtwAttack = 0.5f;
    private int attackCount = 0;
    private int attackCountMax = 3;
    private float attackDelayTime = 1f;
    private float parryTime = 0.5f;
    private GameObject attackPos;
    private bool parryCan = true;
    public float attackRange = 2f;
    //public float damage = 10f;
    private bool isBlockClicked = false;
    private bool canAttack = true;
    private Animator animCont;

    public AudioSource attacksfx;
    void Start()
    {
        playerClass = GetComponent<PlayerClass>();
        animCont = GetComponent<Animator>();
        chara = GetComponent<PlayerChara>();
        move = GetComponent<PlayerMovement>();
        enemyMask = LayerMask.GetMask("Enemy");
        attackPos = GameObject.Find("attackPos");
        cam = GameObject.Find("Main Camera");
    }
}
```

```

    }

    public bool canBlockCheck()
    {
        if (!isAttacking)
        {
            return true;
        }
        else
        {
            return false;
        }
    }

    public IEnumerator attack()
    {
        attacksfx.Play();
        animCont.SetTrigger("Attack");
        isAttacking = true;
        canAttack = false;
        yield return new WaitForSeconds(0.25f);
        timeBtwAttack = startTimeBtwAttack;
        Collider2D[] enemyToDamage =
Physics2D.OverlapCircleAll(attackPos.transform.position, attackRange, enemyMask);
        for (int i = 0; i < enemyToDamage.Length; i++)
        {

            enemyToDamage[i].GetComponent<EnemyClass>().TakeDamage(playerClass.dama
ge);
        }
        isAttacking = false;
        StartCoroutine(AttackCD());
    }

    public IEnumerator CanParry()
    {
        if (!isAttacking && !isBlocking)
        {
            animCont.SetTrigger("Parry");
            isParry = true;
            parryCan = false;
            yield return new WaitForSeconds(0.2f);
            isParry = false;
            StartCoroutine(ParryCD());
        }
    }

```

```

    }

    private IEnumerator AttackCD()
    {
        yield return new WaitForSeconds(startTimeBtwAttack);
        canAttack = true;
    }

    public IEnumerator ParryCD()
    {
        yield return new WaitForSeconds(parryTime);
        parryCan = true;
    }

    void Update()
    {
        if (chara.isDead || chara.isStunned || chara.inDialogue ||
cam.GetComponent<CameraFollow>().inPause)
            return;

        if (Input.GetKeyDown(KeyCode.Mouse0) && canAttack && move.rb.velocity
== Vector2.zero)
        {
            StartCoroutine(attack());
        }
        else if (Input.GetKey(KeyCode.Mouse1))
        {
            if (canBlockCheck())
            {
                animCont.SetBool("Block", true);
                isBlocking = true;
                StartCoroutine(CanParry());
            }
        }
        else if (Input.GetMouseButtonDown(2) && parryCan && move.rb.velocity ==
Vector2.zero)
        {
            StartCoroutine(CanParry());
        }
        else
        {
            animCont.SetBool("Block", false);
            isBlocking = false;
        }

        if (isRecentlyHit)

```

```
{
  startUnHitTime += Time.deltaTime;
  if (startUnHitTime >= unHitTime)
  {
    startUnHitTime = 0;
    isRecentlyHit = false;
  }
}
}
```

```

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

public class PlayerChara : MonoBehaviour
{
    #region BasicStats
    public float hpCurrent = 100f;

    public float spCurrent = 50f;

    private bool spRegenerating = false;
    private bool enduranceRegenerating = false;

    public bool isStunned = false;
    private float stunTime = 1f;

    public GameObject RespawnPoint;
    private LevelCreate levelCreateSRC;
    private FightBehaviour fight;

    public bool isDead = false;
    public bool deathCouroutine = false;

    public bool inDialogue = false;
    #endregion

    #region MoveCost
    [SerializeField] public float jumpCost = 10f;
    [SerializeField] public float dashCost = 25f;
    #endregion
    public PlayerClass player;
    public bool deacrese = false;
    public Camera cam;
    public CameraFollow camSRC;
    private Animator animCont;

    public AudioSource blocksfx;
    public AudioSource parrysfx;
    public AudioSource hitSFX;

    public bool takeDmg(float dmg, int type)
    {

```

```
if(GameObject.FindWithTag("Player").GetComponent<PlayerMovement>().isDashing && (type == 0 || type == 1)) // if dodging && onlyDodge attack or default attack
    return true;
```

```
    if
    (GameObject.FindWithTag("Player").GetComponent<PlayerMovement>().isDashing && type == 2) // if dodging && onlyParry attack
    {
        hpCurrent -= dmg;
        return false;
    }
```

```
    if (fight.isBlocking && type == 0 && !fight.isParry) // if block default attack
    {
        spCurrent -= dmg;
        blocksfx.Play();
        return false;
    }
```

```
    if (fight.isParry && (type == 0 || type == 2)) // if parry default or parryOnly attack
    {
        parrysfx.Play();
        StopCoroutine(fight.CanParry());
        return true;
    }
```

```
    animCont.SetTrigger("Hurt"); // others
    hpCurrent -= dmg;
    hitSFX.Play();
    return false;
}
```

```
public void getPieces(float amount)
{
    var target = player.pieces + amount;
    player.pieces += (int)amount;
}
```

```
public bool canJump()
{
    if (spCurrent >= jumpCost)
        return true;
    else
```

```

        return false;
    }

    public bool canDash()
    {
        if (spCurrent >= dashCost)
            return true;
        else
            return false;
    }

    void Start()
    {
        player = GetComponent<PlayerClass>();
        animCont = GetComponent<Animator>();
        cam = Camera.main;
        camSRC = cam.GetComponent<CameraFollow>();
        levelCreateSRC =
GameObject.FindWithTag("lvlScr").GetComponent<LevelCreate>();
        RespawnPoint = GameObject.FindWithTag("playerPos");
        fight = GetComponent<FightBehaviour>();
        assignStats();
    }

    public void RespawnPointAssign(GameObject pos)
    {
        RespawnPoint = pos;
    }

    public void assignStats()
    {
        hpCurrent = player.hpMax;
        spCurrent = player.spMax;
    }

    private IEnumerator RegenSP()
    {
        spRegenerating = true;
        yield return new WaitForSeconds(3f);
        while (true)
        {
            if (spCurrent < player.spMax)
            {
                spCurrent += 4;
                yield return new WaitForSeconds(0.3f);
            }
        }
    }

```

```

    }
    else
    {
        spRegenerating = false;
        yield return null;
        break;
    }
}
}

```

```

public IEnumerator Death()

```

```

{
    deathCourotine = true;
    isDead = true;
    animCont.SetBool("isDead", true);
    cam.GetComponent<CameraFollow>().YouDied.SetActive(true);
    camSRC.BlackScreenTransparency(1);
    animCont.SetBool("isDead", false);
    yield return new WaitForSeconds(3f);
    assignStats();
    player.pieces /= 2;
    gameObject.transform.position = RespawnPoint.transform.position;
    cam.GetComponent<CameraFollow>().YouDied.SetActive(false);

```

```

    levelCreateSRC.GetComponent<LevelCreate>().foreachEnemyFindDestroy();

```

```

levelCreateSRC.GetComponent<LevelCreate>().foreachCycle(levelCreateSRC.GetComponent<LevelCreate>().Enemy,
levelCreateSRC.GetComponent<LevelCreate>().EnemyPos);
    isDead = false;
    deathCourotine = false;
    camSRC.BlackScreenTransparency(0);
}

```

```

private IEnumerator Stun()

```

```

{
    isStunned = true;
    yield return new WaitForSeconds(stunTime);
    isStunned = false;
}

```

```

void Update()

```

```

{
    if (hpCurrent <= 0 && !deathCourotine)
    {

```

```
        StartCoroutine(Death());
    }
    if (hpCurrent > player.hpMax)
    {
        hpCurrent = player.hpMax;
    }

    if (spCurrent < 0)
        spCurrent = 0;
    if (spCurrent < player.spMax && !spRegenerating)
    {
        StartCoroutine(RegenSP());
    }
    if (spCurrent > player.spMax)
        spCurrent = player.spMax;
}
}
```

```

using System.Collections;
using System.Collections.Generic;
using TMPro;
using UnityEngine;
using UnityEngine.UIElements;
using Image = UnityEngine.UI.Image;

public class PlayerInventory : MonoBehaviour
{
    private PlayerChara chara;
    private PlayerMovement move;
    private FightBehaviour fight;
    private PlayerClass playerClass;
    private GameObject itemBar;
    private Transform posItemUse;
    public int quickCurrent = 0;
    private GameObject knife;
    private GameObject bomb;
    public TextAsset itemsList;
    public TextAsset itemsSprite;
    private string itemListString;

    public AudioSource bombsfx;
    public AudioSource knifesfx;
    public AudioSource potionsfx;
    void Start()
    {
        playerClass = GetComponent<PlayerClass>();
        knife = Resources.Load("Knife") as GameObject;
        bomb = Resources.Load("Bomb") as GameObject;
        posItemUse = gameObject.transform.GetChild(1).transform;
        itemBar = GameObject.Find("itemBar");
        chara = GetComponent<PlayerChara>();
        move = GetComponent<PlayerMovement>();
        fight = GetComponent<FightBehaviour>();
        itemListString = itemsList.text;
        if (!playerClass.inv.Contains(2))
        {
            playerClass.inv.Add(2);
            playerClass.amount.Add(playerClass.flaskMax);
        }
        ChangeSprite();
    }
    public void TakeItem(int id)

```

```

{
    if (playerClass.inv.Contains(id))
    {
        int i = playerClass.inv.IndexOf(id);
        playerClass.amount[i]++;
    }else
    {
        playerClass.inv.Add(id);
        playerClass.amount.Add(1);
    }
}

public void UseItem(int index)
{
    var tmpID = playerClass.inv.IndexOf(index);
    if (playerClass.inv[index] == 2 && playerClass.amount[index] > 0)
    {
        chara.hpCurrent += 25;
        potionsfx.Play();
        playerClass.amount[index]--;
    }
    else if (playerClass.inv[index] == 3 && playerClass.amount[index] > 0)
    {
        var tmppos = new Vector2(posItemUse.position.x, posItemUse.position.y);
        GameObject knife_throw = Instantiate(knife, tmppos, Quaternion.identity);
        knivesfx.Play();
        playerClass.amount[index]--;
    }
    else if (playerClass.inv[index] == 4 && playerClass.amount[index] > 0)
    {
        var tmppos = new Vector2(posItemUse.position.x, posItemUse.position.y);
        GameObject bomb_throw = Instantiate(bomb, tmppos, Quaternion.identity);
        playerClass.amount[index]--;
        bombsfx.Play();
    }
    ChangeSprite();
}

public void ResetFlask()
{
    playerClass.amount[playerClass.inv.IndexOf(2)] = playerClass.flaskMax;
}

private void ChangeItem(int n)
{

```

```

    if (quickCurrent == playerClass.inv.Count-1 && n == 1)
    {
        quickCurrent = 0;
        return;
    }
    else if (quickCurrent == 0 && n == -1)
    {
        quickCurrent = playerClass.inv.Count-1;
        return;
    }

    quickCurrent += n;
}

void Update()
{
    if
(GameObject.FindWithTag("MainCamera").GetComponent<CameraFollow>().inPause)
    {
        return;
    }

    if (Input.GetKeyDown(KeyCode.I))
    {
        Debug.Log(playerClass.inv);
    }
    else if (Input.GetKeyDown(KeyCode.Q) && !fight.isAttacking
&& !fight.isBlocking)
    {
        UseItem(quickCurrent);
    }
    else if (Input.GetAxis("Mouse ScrollWheel") > 0f ) // forward
    {
        ChangeItem(1);
        ChangeSprite();
    }
    else if (Input.GetAxis("Mouse ScrollWheel") < 0f ) // backwards
    {
        ChangeItem(-1);
        ChangeSprite();
    }
}

private void ChangeSprite()
{

```

```

        var tmp = playerClass.inv[quickCurrent];
        var tmp2 = playerClass.inv.IndexOf(tmp);
        var text = itemsSprite.text;
        var spr_path = getBetween(text,playerClass.inv[quickCurrent].ToString() + ":" ,
"\r");
        Sprite itmSpr = LoadSpriteFromFile(spr_path);
        itemBar.transform.GetChild(2).GetComponent<TextMeshProUGUI>().text =
playerClass.amount[tmp2].ToString();
        itemBar.transform.GetChild(1).GetComponent<Image>().sprite = itmSpr;
    }
    private static string getBetween(string file, string strStart, string strEnd)
    {
        if (file.Contains(strStart) && file.Contains(strEnd))
        {
            int Start, End;
            Start = file.IndexOf(strStart, 0) + strStart.Length;
            End = file.IndexOf(strEnd, Start);
            return file.Substring(Start, End - Start);
        }

        return "";
    }
    private Sprite LoadSpriteFromFile(string path)
    {
        // Load the texture from the file
        Texture2D texture = LoadTextureFromFile(path);

        // Create a sprite from the texture
        Sprite sprite = Sprite.Create(texture, new Rect(0, 0, texture.width,
texture.height), Vector2.zero);

        return sprite;
    }
    private Texture2D LoadTextureFromFile(string path)
    {
        // Load the image data from the file
        byte[] imageData = System.IO.File.ReadAllBytes(path);

        // Create a new texture and load the image data into it
        Texture2D texture = new Texture2D(2, 2);
        texture.LoadImage(imageData);

        return texture;
    }
}

```

```

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

public class FightBehaviour : MonoBehaviour
{
    [SerializeField] private PlayerMovement move;
    public PlayerClass playerClass;
    private PlayerChara chara;
    private LayerMask enemyMask;
    private float rayDistanse = 5f;
    public bool isFighting = false;
    public bool isAttacking = false;
    public bool isBlocking = false;
    public bool isParry = false;
    public bool isRecentlyHit = false;
    private float unHitTime = 3f;
    private float startUnHitTime = 0f;
    public GameObject cam;
    private float timeBtwAttack;
    private float startTimeBtwAttack = 0.5f;
    private int attackCount = 0;
    private int attackCountMax = 3;
    private float attackDelayTime = 1f;
    private float parryTime = 0.5f;
    private GameObject attackPos;
    private bool parryCan = true;
    public float attackRange = 2f;
    //public float damage = 10f;
    private bool isBlockClicked = false;
    private bool canAttack = true;
    private Animator animCont;

    public AudioSource attacksfx;
    void Start()
    {
        playerClass = GetComponent<PlayerClass>();
        animCont = GetComponent<Animator>();
        chara = GetComponent<PlayerChara>();
        move = GetComponent<PlayerMovement>();
        enemyMask = LayerMask.GetMask("Enemy");
        attackPos = GameObject.Find("attackPos");
        cam = GameObject.Find("Main Camera");
    }
}

```

```

    }

    public bool canBlockCheck()
    {
        if (!isAttacking)
        {
            return true;
        }
        else
        {
            return false;
        }
    }

    public IEnumerator attack()
    {
        attacksfx.Play();
        animCont.SetTrigger("Attack");
        isAttacking = true;
        canAttack = false;
        yield return new WaitForSeconds(0.25f);
        timeBtwAttack = startTimeBtwAttack;
        Collider2D[] enemyToDamage =
Physics2D.OverlapCircleAll(attackPos.transform.position, attackRange, enemyMask);
        for (int i = 0; i < enemyToDamage.Length; i++)
        {

            enemyToDamage[i].GetComponent<EnemyClass>().TakeDamage(playerClass.dama
ge);
        }
        isAttacking = false;
        StartCoroutine(AttackCD());
    }

    public IEnumerator CanParry()
    {
        if (!isAttacking && !isBlocking)
        {
            animCont.SetTrigger("Parry");
            isParry = true;
            parryCan = false;
            yield return new WaitForSeconds(0.2f);
            isParry = false;
            StartCoroutine(ParryCD());
        }
    }

```

```

    }

    private IEnumerator AttackCD()
    {
        yield return new WaitForSeconds(startTimeBtwAttack);
        canAttack = true;
    }

    public IEnumerator ParryCD()
    {
        yield return new WaitForSeconds(parryTime);
        parryCan = true;
    }

    void Update()
    {
        if (chara.isDead || chara.isStunned || chara.inDialogue ||
cam.GetComponent<CameraFollow>().inPause)
            return;

        if (Input.GetKeyDown(KeyCode.Mouse0) && canAttack && move.rb.velocity
== Vector2.zero)
        {
            StartCoroutine(attack());
        }
        else if (Input.GetKey(KeyCode.Mouse1))
        {
            if (canBlockCheck())
            {
                animCont.SetBool("Block", true);
                isBlocking = true;
                StartCoroutine(CanParry());
            }
        }
        else if (Input.GetMouseButtonDown(2) && parryCan && move.rb.velocity ==
Vector2.zero)
        {
            StartCoroutine(CanParry());
        }
        else
        {
            animCont.SetBool("Block", false);
            isBlocking = false;
        }

        if (isRecentlyHit)

```

```

    {
        startUnHitTime += Time.deltaTime;
        if (startUnHitTime >= unHitTime)
        {
            startUnHitTime = 0;
            isRecentlyHit = false;
        }
    }
}
}

```

Додаток Ж EnemyClass.cs

```

using System.Collections;
using System.Collections.Generic;
using System.Runtime.CompilerServices;
using UnityEditorInternal.Profiling.Memory.Experimental;
using UnityEngine;
using Random = System.Random;

```

```

public class EnemyClass : MonoBehaviour
{

```

```

    #region stats
    public float hpMax;
    public float hpCurrent;
    public float speed;
    public bool isDead;
    public float pieces;
    #endregion

```

```

    #region Attacks
    public float dmg;
    public float range;
    public float attackCd;
    public float attackDelay;
    public bool isStunned;
    public bool canOnlyDodge;
    public bool canOnlyParry;
    #endregion

```

```

    #region Look
    public float distanceToSee;
    public float distanceToAttack;
    public float distanceToPlayer;
    #endregion

```

#region Others

```
Random rng = new Random();  
public bool isAttacking = false;  
public bool canAttack = true;  
#endregion
```

#region References

```
public GameObject playerObj;  
public GameObject itemDrop;  
public Rigidbody2D rb;  
public Animator animController;  
public EnemyBehavior enemyBehavior;  
public GameObject attackPoss;  
public GameObject indicatorDodge;  
public GameObject indicatorParry;  
public LayerMask playerLayer;  
#endregion
```

#region Methods

```
public IEnumerator AttackCd()
```

```
{  
    canAttack = false;  
    yield return new WaitForSeconds(attackCd);  
    canAttack = true;  
}
```

```
public bool SeePlayer()
```

```
{  
    if (distanceToPlayer <= distanceToSee)  
        return true;  
    return false;  
}
```

```
public bool CanAttack()
```

```
{  
    if (distanceToPlayer <= distanceToAttack && canAttack)  
        return true;  
    return false;  
}
```

```
public void Death()
```

```
{  
    StopAllCoroutines();  
    rb.velocity = Vector2.zero;  
    enemyBehavior.enabled = false;  
    GetComponent<CapsuleCollider2D>().isTrigger = true;
```

```

rb.isKinematic = true;
isDead = true;
animController.SetTrigger("isDead");

int dice = rng.Next(1,100);
if (dice % 2 == 0)
{
    var pos = new Vector3(transform.position.x, transform.position.y,
transform.position.z);
    GameObject pickUp = Instantiate(itemDrop, pos, Quaternion.identity);
    dice = rng.Next(1, 100);
    if (dice % 2 == 0) // knifeID == 3
    {
        pickUp.GetComponent<ItemPickUp>().id = 3;
    }
    else // bombID == 4
    {
        pickUp.GetComponent<ItemPickUp>().id = 4;
    }
}
playerObj.GetComponent<PlayerChara>().getPieces(pieces);
this.enabled = false;
}

private IEnumerator StunCd()
{
    isStunned = true;
    animController.SetTrigger("isStunned");
    yield return new WaitForSeconds(2f);
    isStunned = false;
}

public void TakeDamage(float damage)
{
    if (gameObject != null)
    {
        if (!isAttacking)
        {
            animController.SetTrigger("getHit");
        }
        if (isStunned) //hit after stun
        {
            hpCurrent -= damage * 2f;
            StopCoroutine(StunCd());
            isStunned = false;
        }
    }
}

```

```

        return;
    }

    hpCurrent -= damage; //normal hit
}
}

public bool InRangeAttack()
{
    if (distanceToPlayer <= range)
    {
        return true;
    }
    return false;
}

public IEnumerator Attack()
{
    // 0 - default attack
    // 1 - only dodge
    // 2 - only parry
    isAttacking = true;
    var dice = rng.Next(0, 100);
    if (canOnlyDodge && 0 <= dice && dice <= 10) // onlyDodge
    {
        animController.SetTrigger("attackSpecial");
        // Indicator HERE
        indicatorDodge.SetActive(true);
        var tmpColor = indicatorDodge.GetComponent<SpriteRenderer>().color;
        tmpColor.a = 100f;
        indicatorDodge.GetComponent<SpriteRenderer>().color = tmpColor;
        yield return new WaitForSeconds(attackDelay - 0.2f);
        // Indicator FLASH
        tmpColor.a = 255f;
        indicatorDodge.GetComponent<SpriteRenderer>().color = tmpColor;
        yield return new WaitForSeconds(0.2f);
        Collider2D[] colls =
Physics2D.OverlapCircleAll(attackPoss.transform.position, range, playerLayer);
        foreach (var i in colls)
        {
            if (i.GetComponent<PlayerChara>().takeDmg(dmg, 1))
            {
                StartCoroutine(StunCd());
            }
        }
    }
}

```

```

        indicatorDodge.SetActive(false);
    }
    else if (canOnlyParry && 90 <= dice && dice < 100) // onlyParry
    {
        animController.SetTrigger("attackSpecial");
        // Indicator HERE
        indicatorParry.SetActive(true);
        var tmpColor = indicatorDodge.GetComponent<SpriteRenderer>().color;
        tmpColor.a = 100f;
        indicatorParry.GetComponent<SpriteRenderer>().color = tmpColor;
        yield return new WaitForSeconds(attackDelay - 0.2f);
        // Indicator FLASH
        tmpColor.a = 255f;
        indicatorParry.GetComponent<SpriteRenderer>().color = tmpColor;
        yield return new WaitForSeconds(0.2f);
        Collider2D[] colls =
Physics2D.OverlapCircleAll(attackPoss.transform.position, range, playerLayer);
        foreach (var i in colls)
        {
            if (i.GetComponent<PlayerChara>().takeDmg(dmg, 2))
            {
                StartCoroutine(StunCd());
            }
        }
        indicatorParry.SetActive(false);
    }
    else // default
    {
        animController.SetTrigger("attack");
        yield return new WaitForSeconds(attackDelay);
        Collider2D[] colls =
Physics2D.OverlapCircleAll(attackPoss.transform.position, range, playerLayer);
        foreach (var i in colls)
        {
            if (i.GetComponent<PlayerChara>().takeDmg(dmg, 0))
            {
                StartCoroutine(StunCd());
            }
        }
    }
    isAttacking = false;
    StartCoroutine(AttackCd());
}

public void Move()

```

```

{
    Vector2 direction = (playerObj.transform.position -
transform.position).normalized;
    rb.velocity = new Vector2(direction.x * speed, rb.velocity.y);
    animController.SetTrigger("run");
    if (rb.velocity.x > 0)           // mirror gameobject
    {
        transform.localScale = new Vector3(1, 1, 1);
    }
    else if (rb.velocity.x < 0)
    {
        transform.localScale = new Vector3(-1, 1, 1);
    }
}

public void DestroyObj()
{
    Destroy(gameObject);
}
#endregion
}

```

Додаток 3 EnemyBehaviour.cs

```

using System.Collections;
using System.Collections.Generic;
using System.Net.Http.Headers;
using UnityEngine;
using UnityEngine.EventSystems;

public class EnemyBehavior : MonoBehaviour
{
    public EnemyClass enemyClass;
    public bool inAction = false;
    void Start()
    {
        enemyClass = GetComponent<EnemyClass>();
        enemyClass.playerObj = GameObject.FindWithTag("Player");
        enemyClass.animController.SetTrigger("start");
        transform.localScale = new Vector3(1, 1, 0);
    }

    void Update()
    {

```

```

    if (enemyClass.isDead || enemyClass.isStunned || !enemyClass.canAttack)
    {
        enemyClass.rb.velocity = Vector2.zero;
        return;
    }

    if (enemyClass.hpCurrent <= 0 && !enemyClass.isDead)
    {
        enemyClass.rb.velocity = Vector2.zero;
        enemyClass.Death();
    }

    enemyClass.distanceToPlayer = Vector2.Distance(transform.position,
enemyClass.playerObj.transform.position);
    if (enemyClass.SeePlayer() && !enemyClass.CanAttack())
    {
        enemyClass.Move();
    }
    else if (enemyClass.CanAttack() && !enemyClass.isAttacking && !inAction)
    {
        enemyClass.rb.velocity = Vector2.zero;
        StartCoroutine(enemyClass.Attack());
    }
}

private IEnumerator Pause(float time)
{
    inAction = true;
    yield return new WaitForSeconds(time);
    inAction = false;
}
}

```