

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:

«Розробка екшн-гри «Flesh And Bones»»

Виконав студент групи КН-19-1
ХАРЧЕНКО Андрій Володимирович

Керівник роботи:
ЛЕВКІВСЬКИЙ Віталій Леонідович

Рецензент:
МАРЧУК Галина Вікторівна

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: **ХАРЧЕНКО Андрій Володимирович**

Керівник роботи: **ЛЕВКІВСЬКИЙ Віталій Леонідович**

Тема роботи: **«Розробка екшн-гри «Flesh And Bones»»,**

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

Термін здачі закінченої роботи: **«10» червня 2023 р.**

Вихідні дані роботи (зазначається предмет і об'єкт дослідження): об'єктом дослідження є процес проектування та розробки екшн-гри. Предметом дослідження є використання рушія Unity 3D та редактора 3Ds Max Autodesc для створення екшн-гри.

Консультанти кваліфікаційної роботи із зазначенням розділів, що їх стосуються:

Розділ	Консультант	Дата	
		Завдання видав	Завдання прийняв

Календарний план

№ з/п	Назва етапів 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 – 5 березня 2023	Виконано
2	Проектування структури системи	6 березня 2023 – 25 березня 2023	Виконано
3	Написання програмного коду	26 березня 2023 – 16 травня 2023	Виконано
4	Тестування системи	16 травня 2023 – 1 червня 2023	Виконано
5	Оформлення пояснювальної записки	10 травня 2023 – 5 червня 2023	Виконано
6	Попередній захист	10 червня 2023	Виконано

Здобувач вищої освіти **Андрій ХАРЧЕНКО**
Керівник **Віталій ЛЕВКІВСЬКИЙ**

РЕФЕРАТ

Завданням на кваліфікаційну роботу є створення екшн-гри «Flesh And Bones».

Пояснювальна записка до кваліфікаційної роботи на тему «Розробка гри «Flesh And Bones»» складається з вступу, трьох розділів, висновків, списку використаної літератури та додатку. Текстова частина викладена на 61 сторінці друкованого тексту. Пояснювальна записка має 20 сторінок додатків. Список використаних джерел містить 34 найменувань і займає 3 сторінки. В роботі наведено 32 рисунки та 10 таблиць. Загальний обсяг роботи – 81 сторінка.

У роботі було складено концепт та дизайн документи, що містять в собі всю необхідну для створення гри інформацію та зображення змодельованих тривимірних моделей. Було проаналізовано існуючі аналоги програмного забезпечення та визначено основні переваги та недоліки.

Також було зображено розроблену архітектуру важливих структурних елементів за допомогою діаграм та описано роботу основних методів важливих класів гри. Крім того, було проведено тестування ігрового додатку. Наведено його результати та описано процеси усунення виявлених дефектів.

Ключові слова: UNITY, ЕКШН, ГРА, АНІМАЦІЯ, ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ.

					КН.КР.Б – 122 – 23 – ПЗ			
Змн.	Арк.	№ докум.	Підпис	Дата	Розробка екшн-гри «Flesh And Bones» Пояснювальна записка	Літ.	Арк.	Аркушів
Розроб.		А.В. ХАРЧЕНКО						
Керівник		В.Л. ЛЕВКІВСЬКИЙ					3	81
Реценз.		Г.В. МАРЧУК				Житомирська політехніка, група КН-19-1		
Н. Контр.								
Зав. Каф.		М.С. ГРАФ						

ABSTRACT

The task for the qualifying work is to create an action game "Flesh And Bones".

An explanatory note to the qualifying work on "Developing game « Flesh And Bones»» consists of an introduction, three chapters, conclusions, list of references and an appendix.

The text is set out on 61 pages of printed text.

The explanatory note has 20 pages of appendices. The list of used sources contains 34 titles and takes 3 pages. The work presents 32 pictures and 10 table. The total volume of the work is 82 pages.

Concept and design documents containing all the necessary information for the creation of the game and images of the created three-dimensional models were compiled in the work. The existing analogues of the software were analyzed and the main advantages and disadvantages were determined.

Also, the designed architecture of the important structural elements was depicted with the help of a diagram, and the operation of the main methods of the important classes of the game was described. In addition, the game application was tested. Its results are shown and the processes of solving the found defects are described.

Keywords: UNITY, ACTION, GAME, ANIMATION, SOFTWARE.

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

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ	6
ВСТУП	7
РОЗДІЛ 1. АНАЛІЗ ПРОБЛЕМАТИКИ, МЕТОДІВ ТА ЗАСОБІВ ВИРІШЕННЯ ЗАДАЧІ	9
1.1 Постановка задачі	9
1.2 Аналіз існуючого програмного забезпечення	10
1.3 Концепт-документ (Concept Document)	16
1.4 Обґрунтування вибору інструментальних засобів та вимоги до апаратного забезпечення	18
Висновки до першого розділу	19
РОЗДІЛ 2. ПРОЕКТУВАННЯ ТА РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ	20
2.1 Дизайн-документ (Game Design)	20
2.2 Розробка архітектури ігрового додатку	31
2.3 Опис роботи з анімаціями	38
2.4 Основні етапи реалізації гри	41
Висновки до другого розділу	46
РОЗДІЛ 3. ОПИС РОБОТИ З ПРОГРАМНИМ ДОДАТКОМ ТА ЙОГО ТЕСТУВАННЯ	47
3.1 Опис роботи з програмним додатком	47
3.2 Тестування роботи програмного забезпечення	52
Висновки до третього розділу	56
ВИСНОВКИ	57
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	58
ДОДАТКИ	62

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

LShift – Left Shift

RCtrl – Right Control

UI – User Interface

КР – кваліфікаційна робота

ЛКМ – ліва клавіша миші

ПЗ – програмне забезпечення

ПКМ – права клавіша миші

					КН.КР.Б – 122 – 23 – ПЗ	Арк.
						6
Змн.	Арк.	№ докум.	Підпис	Дата		

ВСТУП

Актуальність теми.

У сучасному світі ігри відіграють далеко не останню роль у житті середньостатичної людини. Сьогодні вони є важливим індустріальним сектором, що становить значну частину розважальної галузі. Ігри розвиваються з кожним роком, та пропонують неймовірно широкий спектр жанрів та технологій. Особистості різної статі, від підлітка до особи похилого віку, звертаються до цієї потіхи заради різноманітних потреб. Загалом, основною ціллю використання продуктів ігрової індустрії є звичайна можливість приємно провести свій час на дозвілля, проте для безлічі людей – це цілий світ, який надає їм змогу розвивати свої навички та здібності, зустрічатися зі своїми друзями та навіть заробляти на цьому кошти проживання. Зараз ігри стали більш доступними та різноманітними, тому кожен може знайти щось для себе, незалежно від свого віку, зацікавлень та здібностей.

Через те, що популярність комп'ютерних ігор з кожним днем все зростає, їм знаходять все більше нових галузей використання. Як повідомляє видання Knowable Magazine, деякі типи відеоігор можуть покращити роботу мозку під час виконання вузького набору дій. Ряд досліджень також показав їхню ефективність при освоєнні нової мови, вивчення математики та природних наук. Проведення приблизно години в день в шутерах покращує візуальну увагу, а гра в тетріс – просторове мислення. Проте потрібно пам'ятати, що якщо час на відеоігри витісняє час на інші завдання, інші навички можуть постраждати^[1].

На популярність ігор також вплинула пандемія Covid19, що замкнула людей в своїх будинках на час самоізоляції та дистанційного навчання, змушуючи їх знаходити способи проведення часу на дозвілля разом зі своїми друзями, не піддаючись небезпеці розповсюджувати вірус.

Жанр екшн завжди був дуже популярним серед гравців, оскільки він надає швидку та захоплюючу геймплейну динаміку. За останні кілька років популярність ігор жанру екшн зростає ще більше завдяки технологічному

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

прогресу, покращенню графіки та збільшенню можливостей геймдизайну. Переважна кількість піджанрів екшн-ігор мають свою частину уваги на ринку ігрової продукції[2].

Стиль «low-poly» також набирає свою популярність через свою простоту та мінімалізм. Це стиль моделювання, коли тривимірні моделі мають невелику кількість полігонів. Він також використовується, якщо приємна якість візуалізації може бути отримана з меншими витратами, тобто «low-poly» добре позначається на продуктивності.

Метою створення даного проекту є розробка гри «Flesh And Bones» в жанрі екшн на рушії Unity.

Об'єктом дослідження є процес проектування та розробки екшн-гри.

Предметом дослідження є використання рушія Unity 3D та редактора 3Ds Max Autodesc для створення екшн-гри.

Методи дослідження: в роботі було використано методи об'єктно-орієнтованого проектування, програмування та дизайну.

В рамках розробки кваліфікаційної роботи були опубліковані наукові праці:

Харченко А. В., Левківський В.Л. Особливості використання системи анімацій Mecanim при створенні гри на Unity. Тези XIII Міжнародної науково-технічної конференції «Інформаційно-комп'ютерні технології – 2023 (ІКТ-2023)», м. Житомир, 30 - 31 березня 2023 р. – Житомир: Житомирська політехніка, 2023. С. 78-79.

					КН.КР.Б – 122 – 23 – ПЗ	Арк.
						8
Змн.	Арк.	№ докум.	Підпис	Дата		

РОЗДІЛ 1. АНАЛІЗ ПРОБЛЕМАТИКИ, МЕТОДІВ ТА ЗАСОБІВ ВИРІШЕННЯ ЗАДАЧІ

1.1 Постановка задачі

Основною метою розробки проекту є створення ігрового додатку, що забезпечить декілька годин веселого та динамічного геймплею. Створений продукт надасть користувачам можливість провести час, насолоджуючись грою та успіхами головного героя, що будуть залежати від його власних дій.

Розробка проекту включатиме в себе декілька етапів, ціллю яких буде створення визначених частин гри. Основними серед них є:

1. Користувацький інтерфейс
 - 1.1.Розробити концепцію дизайну
 - 1.2.Створити основні елементи інтерфейсу у наявних графічних редакторах
 - 1.3.Використати створені елементи в Unity
 - 1.3.1. Зверстати інтерфейс головного меню
 - 1.3.2. Зверстати інтерфейс меню налаштувань
 - 1.3.3. Зверстати інтерфейс інвентаря
 - 1.3.4. Зверстати інтерфейс меню паузи
 - 1.3.5. Зверстати інтерфейс HUD
2. Моделювання та анімування
 - 2.1.Створення моделей зброї
 - 2.2.Створення моделі людини
 - 2.3.Анімування головного героя
 - 2.4.Створення моделі ворогів
 - 2.5.Анімування ворогів
 - 2.6.Анімування боса
3. Розробка основних модулів гри
 - 3.1.Модуль управління головним героєм
 - 3.2.Модуль здібностей головного героя
 - 3.3.Модуль управління анімаціями персонажа

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

3.4.Модуль атак та шкоди

3.5.Модуль статус ефектів

3.6.Модуль штучного інтелекту ворогів

4. Створення візуальних ефектів

4.1.Ефекти здібностей

4.2.Ефекти снарядів

Результатом роботи над проектом є готовий ігровий додаток жанру екшн.

1.2 Аналіз існуючого програмного забезпечення

Жанр екшн є одним з найпопулярніших та найрозповсюдженіших жанрів у світі ігор. Його можна охарактеризувати геймплеєм з високою інтенсивністю, де від гравця вимагаються швидкі дії на поточну ситуацію у грі. Велику кількість екземплярів цього жанру можна пояснити тим, що він містить в собі безліч піджанрів, таких як:

- шутер, головна механіка в якій полягає у використанні зброї або інших засобів ліквідації противника на великих відстанях;
- платформер, акцент в якій робиться на керуванні головним персонажем та подоланні перешкод;
- файтинг, який фокусується на механіці ближнього бою;
- стелс-екшн, в якому гравцю необхідно уникати прямого контакту з ворогами;
- hack and slash, який акцентується на швидкому бою з широким арсеналом зброї на комбінацій атак;
- beat'em up, в якому гравцю доводиться вести бій одночасно з великою кількістю ворогів[3].

В даному випадку аналіз буде відбуватися такий ігрових додатків, геймплей яких прийматиме схожі риси з геймплеєм кваліфікаційного проекту. Тому слід навести перелік рис, які було вирішено надати геймплею під час його розробки:

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

1. Управління переміщення головним героєм здійснюється клавішами, атаки та огляд місцевості – мишкою.
2. Розміщення камери вище головного героя та яка дивиться на поле бою під кутом, що надає гравцю більш тактичне зображення області гри.
3. Наявність здібностей у головного героя та можливість вибору або їх модифікацій.
4. Швидкий та інтенсивний геймплей, акцент відбувається на вміннях гравця.
5. Наявність складних ворогів, які мають в розпорядженні певний набір можливостей та здібностей.
6. Прогресія здібностей та предметів.

Отже, для аналізу було обрано наступні продукти ігрової індустрії: Battlerite, Akane, Hyper Light Drifter. Всі вони мають характерний подібний геймплей та надають гравцю схожу палітру емоцій під час процесу гри. Для деяких з них найкращим варіантом задля отримання повного досвіду гри є використання геймпаду, проте це не є обов'язковим.

Battlerite – безкоштовна командна мультиплеєрна онлайн-гра жанру «battle arena», опублікована та розроблена Stunlock Studios на рушій Unity. Її геймплей пропонує гравцю бої, що виділяються своєю швидкістю та інтенсивністю. В грі існує вибір з-поміж безлічі персонажів, кожен з яких має свої власні унікальні здібності та перелік обов'язків в команді. Самі бої проходять 2х2 або 3х3, та відбуваються в доволі невеликому замкнутому просторі, що змушує використовувати особливості навколишнього середовища.

Кожен ігровий персонаж також відрізняється своїм унікальним дизайном: стилем одягу, рухами та атаками, візуальними ефектами тощо, що також вносить свій внесок до якості процесу гри. Графічний стиль можна охарактеризувати як «стилізоване фентезі» – світ відтворено яскраво, кольорово та весело, анімації персонажів детальні, виразні та характерні для кожного героя.

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

Сюжетної лінії в самій грі, як такої, немає, проте кожен персонаж має власну біографію, що описує певним чином ігровий всесвіт та надає гравцю розуміння подій та причин, через які вони відбуваються[4].

Переваги: веселий та динамічний геймплей; донат не впливає на гру; гра за різних героїв відрізняється; приємний візуальний стиль.

Недоліки: низький онлайн; може бути складною у освоєнні.

На рисунку 1.1 зображено постер гри Battlerite.



Рис. 1.1 – Battlerite

Akane – аркадний файтинг-слешер на арені, розроблена і видана компанією Ludic Studios. Її геймплей також пропонує швидкі та інтенсивні бої з великою кількістю ворогів одночасно. Від гравця вимагається відбивати їх атаки та вчасно здійснювати свої. У головної героїні також є можливість використовувати вогнепальну зброю дальнього враження, що надає користувачу ширший вибір дій в процесі бою. Хоча геймплей і може виглядати доволі простим, він випробовує рефлекси та навички гравця.

Після кожної поразки у користувача є можливість покращити або змінити конфігурацію зброї для наступної битви, що надає грі відчутні риси жанру «rougelike».

Графічний стиль можна описати як «retro pixel art» в кіберпанк-середовщі – він переймає виключно позитивні риси старих класичних ігор у 8 або 16 біт, що надає грі естетику ностальгії. Стиль Akane характеризується

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

простими та деталізованими піксельними зображеннями. Незважаючи на це, він вміло передає швидкість рухів та екшн-наповненість гри[5].

Гра містить в собі сюжет, хоча його наявність і є чистою формальністю, адже весь його вміст подається гравцю протягом перших п'яти хвилин, що включає в себе першу відео-вставку та тьюторіал.

Переваги: приємний візуальний стиль; наявність різної зброї, що змінюють геймплей; гра може продовжуватись нескінченно.

Недоліки: швидко набридає; мало контенту.

На рисунку 1.2 зображений постер зі грою Akane.



Рис. 1.2 – Akane

Hyper Light Drifter – рольова екшн-гра, розроблена та опублікована студією розробки Heart Machine. Основна частина часу гри включає в себе дослідження внутрішнього світу, пошук предметів для покращення головного героя та головоломки, за вирішення яких нагородою є ті ж самі предмети. Проте, щоб ретельно дослідити локацію, необхідно перемогти певну кількість ворогів, що там знаходяться. Самі битви проходить інтенсивно, і їх складність може відштовхнути від себе частину гравців. Проте основною складністю є те, що світ приймає доволі чималі розміри, гравець має велику свободу дій, але поточного завдання, мітки на карті, або навіть простого тексту з поясненнями, в грі немає – всі діалоги з різними внутрішньоігровими персонажами виконані шляхом обміну картинок. Все вищезгадане надає користувачу особливий, персональний досвід від процесу

гри та дає можливість по-справжньому насолодитися грізною атмосферою світу, що вмирає.

Бойова механіка приймає витончені риси – гравцеві необхідно продумувати кожен свій рух, спостерігати за діями ворогів та тримати ситуацію у своїх руках, яка швидко може змінитися не у користь головного героя. Для ведення бою у гравця в асортименті є: різноманітні комбо, далекобійна зброя, механіка ривків та навіть особливі здібності, які користувач може придбати або навчитись у неігрових персонажів.

Візуальний стиль можна описати як «retro pixel art», проте, на відміну від Akane, він виконаний під середовище постапокаліптичної Землі та загадкових небезпечних руїн. Палітра кольорів у Hyper Light Drifter ретельно підібрана, щоб створити смутний атмосферний тон. Він часто включає в себе насичені відтінки синього, фіолетового та помаранчевого, що сприяє створенню ефектного та захоплюючого світу гри[6].

Переваги: велика кількість покращень, що змінюють геймплей; приємний візуальний стиль та звуковий супровід.

Недоліки: може здатися занадто повільною.

На рисунку 1.3 зображений момент з гри Hyper Light Drifter.

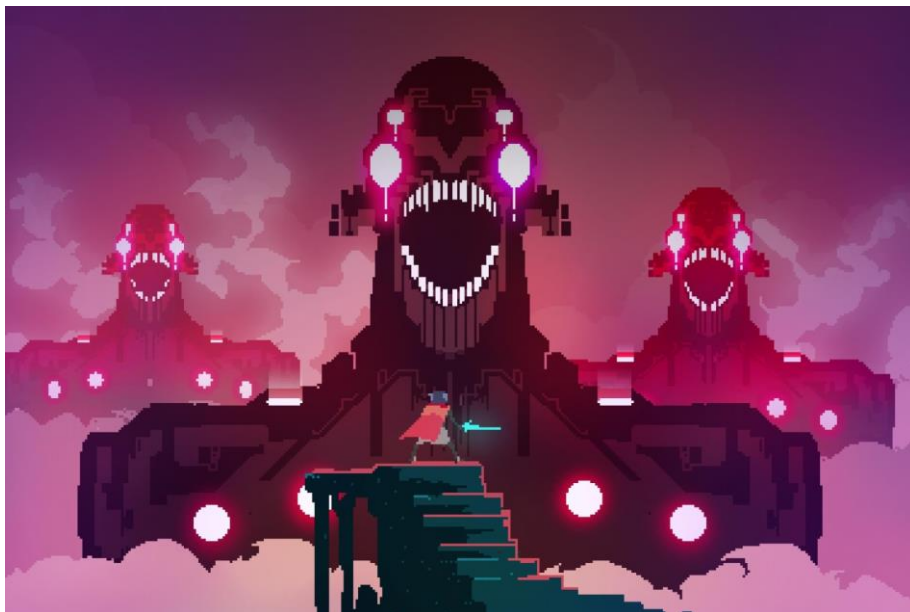


Рис. 1.3 – Hyper Light Drifter

Розглянемо дані аналоги за мінімальними системними вимогами.

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

Таблиця 1.1

Порівняння мінімальних системних вимог аналогів

Назва	Процесор	Оперативна пам'ять	Відеокарта	Місце на диску
Battlerite	2,8 GHz	4 GB	Від 256 MB відеопам'яті	3 GB
Akane	2,4 GHz	2 GB	Від 256 MB відеопам'яті	200 MB
Hyper Light Drifter	1.2 GHz	4 GB	Від 512 MB відеопам'яті	2 GB

Розглянемо геймплейні особливості та деталі вищезгаданих аналогів.

Таблиця 1.2

Порівняння геймплейних особливостей аналогів

Назва	Мультплеєр	Вибір зброї	Покращення	Відкритий світ	Бойова механіка
Battlerite	Так	Ні	Так	Ні	4 / 5
Akane	Ні	Так	Ні	Ні	3.5 / 5
Hyper Light Drifter	Ні	Ні	Так	Так	5 / 5

Розглянемо графіку та аудіо даних ігрових додатків.

Таблиця 1.3

Порівняння графіки та аудіо аналогів

Назва	Візуальний стиль	Звуковий супровід	Візуальні ефекти	Анімації	Середня
Battlerite	3	2	5	5	3,75
Akane	4	4	4	2	3,5
Hyper Light Drifter	5	5	4	4	4,5

Для оцінки аналогів було взято 5-бальну шкалу оцінювання, де 5 – це найкращий в даному розділі серед інших, а 1 – найгірший.

Проаналізувавши особливості взятих на аналіз продуктів-аналогів кваліфікаційного проекту можна визначити напрямки, за якими слід рухатись

при розробці проекту, недоліки, котрі краще уникати або виправити, та аспекти, на які акцентувати геймплей.

Отже, розроблена гра має перейняти переваги та виправити недоліки розглянутих вище прикладів:

1. Приємний візуальний стиль та ефекти
2. Динамічний геймплей
3. Наявність вибору зброї та можливість її покращувати
4. Прогресія

1.3 Концепт-документ (Concept Document)

Жанр і цільова аудиторія

Жанр – «Action». Піджанри: RPG», «Slasher». Вони відзначаються інтенсивним та швидким геймплеем, де акцент робиться на бойових взаємодіях гравця зі ворогами. Також ігри даного жанру можуть мати елементи прогресії та розвитку персонажа, які дозволяють гравцю збирати досвід, отримувати нові здібності, поліпшувати характеристики та вибирати спеціалізації тощо. Крім того, вони пропонують великі світи для дослідження, з різноманітними локаціями, загадками, завданнями та ворогами. Гравець може проходити через різні рівні або зони, зустрічати різноманітних персонажів та виконувати квести[7].

Гра націлена на чоловіків віком від 16 до 28 років, психотип по Бартлу яких – killer-explorer. Від гри вони очікують тяжкі та видовищні битви, між переривами яких є можливість дослідити внутрішньоігровий світ, інтерактувати з всесвітом та знайти для ведення бою нові предмети та аксесуари. Потенційні гравці, в більшості, є хардкорними або мідкорними. Гра може потребувати від гравців швидкої реакції та рішучих дій.

Основні особливості гри

Особливості гри полягають в динамічному веденні бою та приємному візуальному стилі «low-poly».

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

Геймплей

Геймплей полягатиме у використанні всіх наявних у головного героя здібностей задля отримання перемоги над ворогами. Їх кількість може різнитися, від одного ворога, який входить до «вищого рангу», якого перемогти є доволі складно, так і безлічі низькорівневих, основною проблемою яких подолати є виключно їх кількість. За подолання деяких ворогів або дослідження карти головний гравець в процесі гри буде отримувати або знаходити різноманітні предмети, які тим чи іншим чином внесуть свій внесок у перемозі над супротивником.

Сюжет

Основні події відбуваються у V-XI столітті, де відкрили існування життєвої енергії у тілі живих істот. Ця знахідка створила людей, які почали використовувати цей вид енергії для своїх цілей. В результаті багатьох подій використання вищезгаданої сили заборонили через високу небезпеку життю та здоров'ю. Також поважною причиною стало просте нерозуміння, що доволі часто має тенденцію перетворюватися в страх.

Головний герой – командир штурмового загін. Його відправили зі своєю командою перевірити статус особливого селища, розташованого далеко за стінами королівства. По прибутті на визначене місце вони виявили відсутність людей та дивну будівлю, що стояло в кінці поселення.

При перевірці споруди команду було атаковано невідомими ворожими силами. Головного героя та деілька інших членів команди було ув'язнено. Від цього моменту керування головним героєм надається гравцю, якому потрібно буде врятуватися, пробиваючись через групи різноманітних ворогів, дізнаючись все більше про місце, в якому він знаходиться.

Платформа

Гра розробляється виключно під РС-платформу.

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

1.4 Обґрунтування вибору інструментальних засобів та вимоги до апаратного забезпечення

На сьогоднішній день існує доволі чимала кількість різноманітних рушіїв для розробки ігор, відштовхуючись від потреб розробників. Найпопулярніших серед них є Unity. Одним з його основних переваг є його мультиплатформенність – він підтримує розробку ігор під такі платформи, як: персональний комп'ютер, консолі, мобільні пристрої та навіть віртуальну реальність. Unity має всі необхідні інструменти для розробки як 2D ігор, так і 3D та відносно доволі легкий у засвоєнні. Причиною цього є його зрозумілий зручний візуальний інтерфейс та принцип компонентного підходу до розробки, який полягає у використанні та маніпулюванні різноманітними компонентами, кожен з яких виконує свою певну функцію та розташовуються безпосередньо на ігрових об'єктах. Скриптова система ігрового рушія зроблена на Mono — вільному відкритому проєкті з реалізації .NET Framework[8]. Для програмування, в основному, використовують мову програмування C#, хоча також є UnityScript та Boo, які не часто можна зустріти серед розробників[9].

Іншим широковідомим рушієм є конкурент Unity – Unreal Engine. На відміну від Unity, в ньому використовується два основним способи програмування: мова C++ та так звані «блупрінти» – візуальна система програмування, що дозволяє проектувати складну логіку та функції інтеракції, без написання коду вручну. Окрім вищезгаданих «блупрінтів», серед його основних особливостей та переваг є якість графічної складової, що включає в себе широкий спектр графічних можливостей розробки, фотореалістичну графіку, якісне освітлення, тіні, воду, візуальні ефекти тощо. Також Unreal Engine має доволі потужну систему фізики, що дозволяє розробникам створювати реалістичну взаємодію об'єктів між собою, збільшуючи тим самим реалізм внутрішньоігрового всесвіту ігрового продукту[10].

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

Незважаючи на всі особливості Unreal Engine, було обрано Unity як рушій для розробки проекту через наступні його переваги:

1. Мова програмування C#. Вона має активну спільноту розробників та широкий асортимент ресурсів, бібліотек та інструментів.
2. Активна спільнота. Unity має свою власну документацію, в якій описані всі необхідні відомості про цей чи інший функціонал рушія.
3. Візуальний редактор. Unity має потужну систему інструментів, яка дає розробникам можливість легко налаштовувати сцени, ігрові об'єкти, компоненти, розташовані на них, та логіку гри загалом без потреби навіть написання коду.

Для моделювання, анімування та текстурювання моделей для гри використовувався 3D Max, для створення спрайтів та необхідних для користувацького інтерфейсу графічних зображень було використано Adobe Photoshop.

Висновки до першого розділу

В результаті написання першого розділу було поставлено задачу на кваліфікаційну роботу, а саме – розробити екшн гру, геймплей якої буде напрямлений на інтенсивність та дослідження ігрового світу. Було проведено аналіз ігрових додатків та визначено їх позитивні та негативні риси, сформульовано головні аспекти, на які слід орієнтуватися при розробці проекту, та деталі, які слід до уваги.

В концепт-документі було зазначено жанр та піджанри ігрового додатку, цільову аудиторію, для якої буде розроблятися гра, її особливості, геймплей, сюжет та платформу.

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

РОЗДІЛ 2. ПРОЕКТУВАННЯ ТА РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Дизайн-документ (Game Design)

Гра розроблена в інтерактивному форматі. Сюжет, як такий, подається у вигляді тексту: залишені листи, послання, написи, фрази головного героя тощо. Мета даної гри – дістатись виходу з рівня, пробиваючись через групи ворогів, які очікують головного героя на кожному його кроці.

Управління головним героєм.

Управління переміщенням персонажа здійснюється за допомогою клавіш WASD, що відповідають за рух по осях X та Z в ігровому світі, тобто в сторони: вліво, вправо, вперед та назад, а повороти по осі Y – мишкою. Під час руху головний герой може здійснити швидкий ривок, в бік, куди він рухається в момент його здійснення. Клавіша, яка по замовчуванню за це відповідає, – LShift. Якщо натиснути на вищезгадану клавішу в той проміжок часу, коли головний герой не виконує переміщення по ігровому середовищі, він здійснить «ухилення». При натисканні клавіші LCtrl, не важливо, чи рухається головний герой, він спробує відбити ворожу атаку.

За здійснення атак, загалом, відповідають клавіші ЛКМ та ПКМ, які змушують головного героя виконувати удари в тому напрямку, куди він дивиться. В залежності від натиснутої клавіші, буде програватись різна анімація атаки. Також в асортименті головного героя є удар ногою, який відштовхуватиме ворогів та предмети в напрямку погляду головного героя.

Різноманітні здібності, якими головний герой може користуватись, по замовчуванню, призначенні на такий набір клавіш, як QERZXC.

Керування інвентарем здійснюється за допомогою мишки, де натискання ЛКМ по слоту з предметом виведе його інформацію у відповідальну за це панель, а ПКМ – надягне його на персонажа. Відкриття інтерфейсу інвентаря можна здійснити клавішою Tab, а визвати меню паузи під час гри – Esc.

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

Враховуючи головне меню, в результаті роботи над проектом буде розроблено п'ять сцен. Одна відповідає за головне меню, яке відкриватиметься при початковому запуску гри, три – сюжетна лінія, і ще одна, якщо сюжетної кампанії виявиться недостатньо, де гравець зможе проводити необмежений проміжок часу, розважаючись на арені з ворогами.

Ігрові механіки.

Основними ігровими механіками є переміщення головного героя та ведення бою.

Рух персонаж може здійснювати лише в площині XZ. Під час руху головний герой може виконувати ривок в сторону, в яку він прямував на початку його виконання. Під час цього стану він все одно може отримувати шкоду, тому єдиним призначенням використання ривка є швидке переміщення по ігровому середовищу.

Задля можливості уникнути отримання шкоди від ворожої атаки використовується механіка ухилення. Ця можливість є не проти всіх рухів противників, адже деякі з них неможливо оминати, перебуваючи на небезпечній від них відстані. Схожа ситуація з механікою парирування. При здійсненні спроби зробити відбиття перевіряється, чи можливо це виконати проти даної атаки. При успішному результаті визиваються відповідні події, в яких реалізовано наступне управління станами персонажів.

Головний герой має можливість використовувати різноманітні здібності, які він буде отримувати на своєму шляху. При натисканні відповідної клавіші управління персонажем, він переходить у стан, який відповідає за логіку використання вмінь. Керування анімаціями та іншими необхідними ігровими об'єктами надається безпосередньо функції використаної здібності.

Під час здійснення атак головний герой переходить у відповідний стан, при якому його швидкість переміщення та обертання по осі Y модифіковано властивостями використаної атаки. Наприклад, при проведенні атак руками або одноручною зброєю, головний персонаж, зазвичай, лише трохи

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

сповільнює швидкість руху, а при виконанні атак важкою зброєю, він зупиняється зовсім і втрачає можливість змінювати напрямок погляду під час самої атаки.

Необхідність використання механіки удару ногою буде супроводжувати гравця протягом лівової частки геймплею гри. Вона змушує героя виконати удар, який відштовхує предмети та інших персонажів своєму шляху залежно від їх маси. Під час її здійснення головний герой не може рухатись, виконувати атаки та використовувати інші здібності до кінця анімації удару. Слід зазначити, щоб відштовхувати також можуть деякі атаки ворогів та зброї. При відштовхуванні предмету він нанесе фізичну шкоду першому персонажу у розмірі швидкості руху, поділеної на масу цього предмету. При зменшенні швидкості до певного значення він втрачає можливість наносити шкоду та стає кінематичним.

Наявність інвентарю надає гравцю можливість змінювати поточне спорядження головного героя. При спробі надягти предмет, слот якого вже зайнятий іншим предметом, він заміниться на новий, а старий повернеться назад до інвентарю.

В грі розроблена система атрибутів, кожен з яких відповідатиме за конкретні властивості персонажа. Всього їх три, це базовий набір, який простежується в більшості ігор жанру «RPG»: сила, спритність та кмітливість[11]. Кожен персонаж, який може отримувати шкоду, має власний початковий комплект значень атрибутів, і при першій своїй ініціалізації вони вносять зміни в різноманітні характеристики, наприклад: сила впливає на кількість здоров'я персонажа, спритність відповідає за швидкість пересування, а кмітливість – за кількість життєвої енергії, яким персонаж може керувати та використовувати в здібностях. Також вони відповідають за значення опору до різноманітних типів шкоди. Наприклад, чим більше у персонажа здоров'я – тим сильніше в нього опір до фізичної шкоди, отрути, кровотечі та холоду.

На початку головний герой має наступні значення атрибутів:

- Сила: 5
- Спритність: 5

					КН.КР.Б – 122 – 23 – ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		22

- Кмітливість: 5

Пошкодження з часом (DoT) – це термін, який використовується для опису шкоди від різних типів джерел, що наносяться персонажеві протягом певного періоду через визначений інтервал часу[12]. Джерелами накладання DoT можуть бути результати використання здібностей, предмету, ефект атаки зброї тощо. Прикладом такої шкоди може бути кровотеча, полум'я, отрута. Її типом може бути всі можливі типи шкоди в грі.

Персонажі та створені моделі.

Основним гейм-об'єктом є персонаж головного героя (рис. 2.1), що являє собою звичайну людину без особливих видатних візуальних властивостей. Зовнішньо головний герой не відрізняється від середньостатистичної людини – він має світлий колір шкіри та середні розміри тіла. Більшість екіпованих гравцем предметів будуть візуалізовані на моделі головного героя, а зміна типу зброї буде вносити зміни в його рухи, наприклад: атаки, використання здібностей, бігу тощо.



Рис. 2.1 – Модель головного героя

Переважаючою частиною персонажів гри є саме вороги, з якими головному герою доведеться зіткнутись. Найпершою перешкодою для гравця буде звичайний «зомбі» модель якого запозичена у головного героя, проте його зовнішній вигляд все ж таки відрізняється від звичайного – колір ближчий за відтінком до зеленого, а пропорції тіла трішки худорлявіші. Його

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

рухи можна охарактеризувати як повільні та недбалі, а в розпорядженні він має лише одну атаку. Окрім звичайного «зомбі», можна виокремити такого, що використовує зброю (рис. 2.2). На відміну від попереднього, він може виконувати дві атаки. «Зомбі» будуть наявні на всіх локаціях гри, починаючи з першої.



Рис. 2.2 – Ворог «зомбі»

Наступного ворога за складністю гравець зустріне «гуля» (рис. 2.3). Він має в розпорядженні п'ять атак і набагато швидший та витриваліший за «зомбі». Також він може накладати кровотечу та відштовхувати головного героя. Розміщуватись вони будуть на другому та третьому рівні гри.

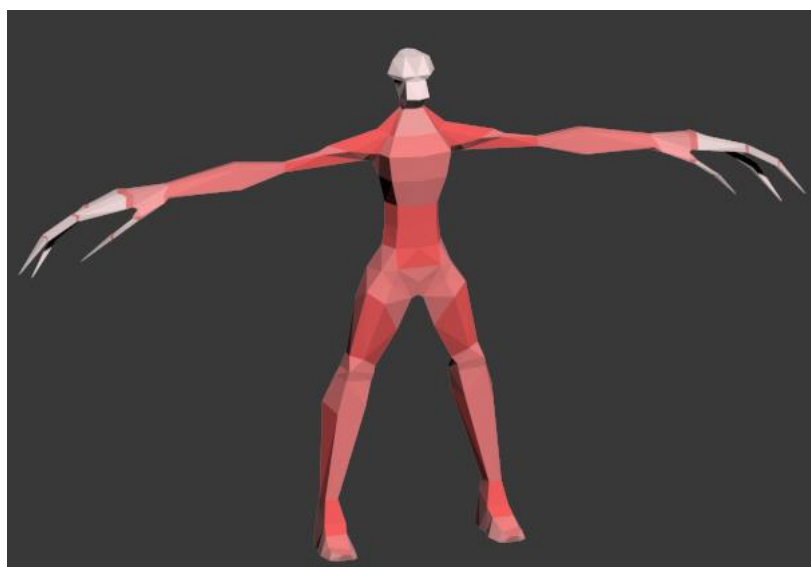


Рис. 2.3 – Ворог «гуль»

Маг – єдиний ворог, ціль якого, на відміну від інших, не підійти до гравця, а тримати зручну для нього дистанцію. Задля атак він використовує вогневі та енергетичні снаряди, що можуть як летіти за напрямком, так і за зазначеною ціллю, наприклад, гравцем. Має особливий стан: коли після довготривалого бою у нього закінчується енергія, він зупиняється та собирається із силами. Під час цього процесу у нього змінюються опори до атак, що в результаті захищає від фізичної шкоди, та робить вразливим до енергії.



Рис. 2.4 – Ворог маг

Ворог з мечем (рис. 2.5) є одним з найнебезпечніших ворогів для гравця – він швидко зменшує відстань до головного героя та здійснює смертельні атаки, яких в його асортименті нараховується п'ять штук. При великій дистанції він здатен створювати та кидати снаряд із льоду у головного героя, що змушує гравця завжди бути в ближньому бою. Це єдиний ворог, що вміє виконувати ривок перед здійсненням атаки з ціллю збити гравця з пантелику. Направлення, довжина, та швидкість ривку розраховується від випадкового значення, тому передбачити його майже неможливо. Також він має велику кількість здоров'я та кмітливості, що дарує йому високий опір до фізичної шкоди, проте також вразливість до вогню.

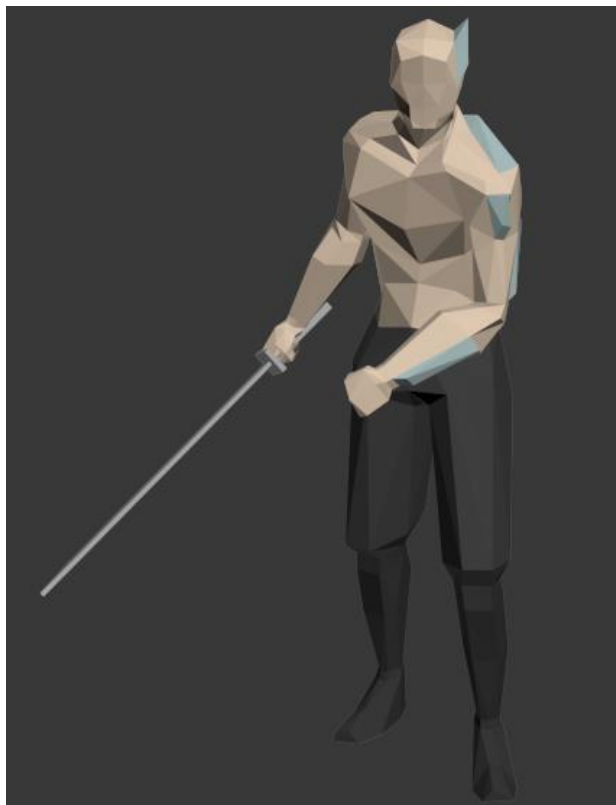


Рис. 2.5 – Ворог з мечем

Окрім ворогів, гравцю прийдеться зустрітись персонажа, з яким можна інтеректувати (рис. 2.6). На першому рівні він допомагає герою опинитись на іншій стороні решітки.



Рис. 2.6 – Персонаж з бородою

					КН.КР.Б – 122 – 23 – ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		26

Для реалізації певних здібностей, механік, геймплейних предметів були розроблені додаткові тривимірні моделі. Щупальце (рис. 2.7) – використовується в одній здібності головного героя, що полягає в розміщенні ігрового об’єкту, яке атакує ворогів.



Рис. 2.7 – Щупальце

Для реалізації об’єктів, що створюють та викидують снаряди з ціллю нашкодити головною герою, також було створено відповідні моделі (рис. 2.8).

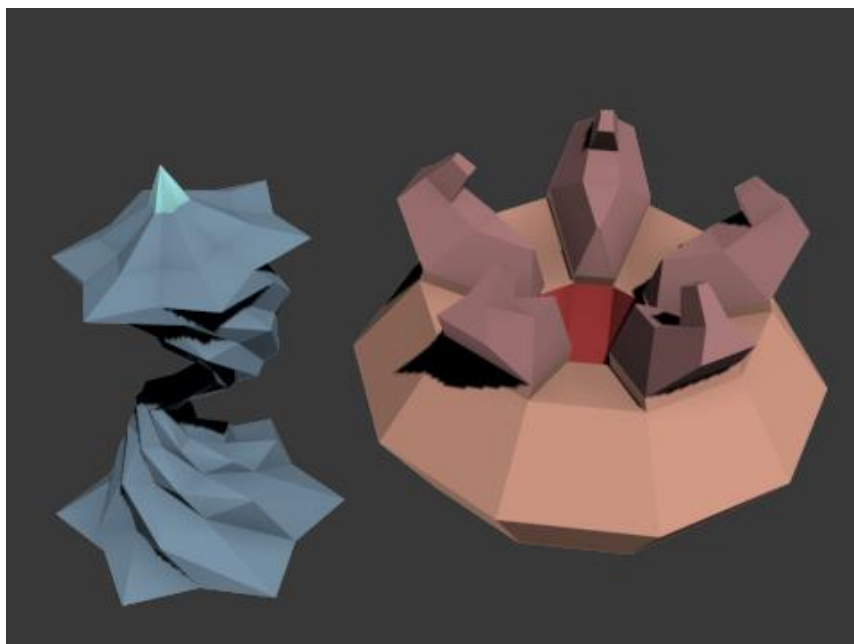


Рис. 2.8 – Пастки, що стріляють снарядами

Важіль (рис. 2.9) використовується як тригер, що виконує певні дії при його використанні. Інтеракція з ним виконуватиметься за допомогою ударів.



Рис. 2.9 – Важіль



Рис. 2.10 – Створені моделі зброї

Вся зброя (рис. 2.10), якою головний герой може користуватись, класифікується за своїм типом, різниться за характеристиками, властивостями та вносить зміни у атрибути персонажа. Деякі з них мають

					КН.КР.Б – 122 – 23 – ПЗ	Арк.
						28
Змн.	Арк.	№ докум.	Підпис	Дата		

додаткові варіації за зовнішнім виглядом та геймплейними ефектами при користуванні.

Також було створено моделі екіпування, які, при використанні, будуть одягнуті на модель головного героя. На рис. 2.11 можна бачити приклад зовнішніх виглядів шоломів.

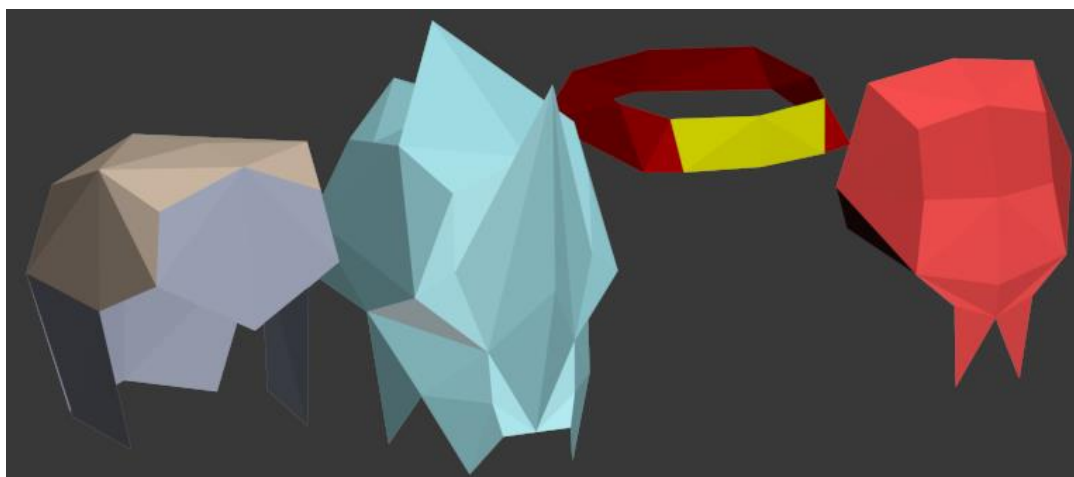


Рис. 2.11 – Створенні моделі шоломів

Ігрові локації знаходяться в замкнутому просторі, тому їх створення полягає в моделюванні шляху, по якому гравець буде прямувати, та кімнат, в яких буде розміщений ігровий контент, за допомогою простих геометричних форм. Гра починається на першому рівні, що являє собою в'язницю, з якої головному герою прийдеться вибиратись.

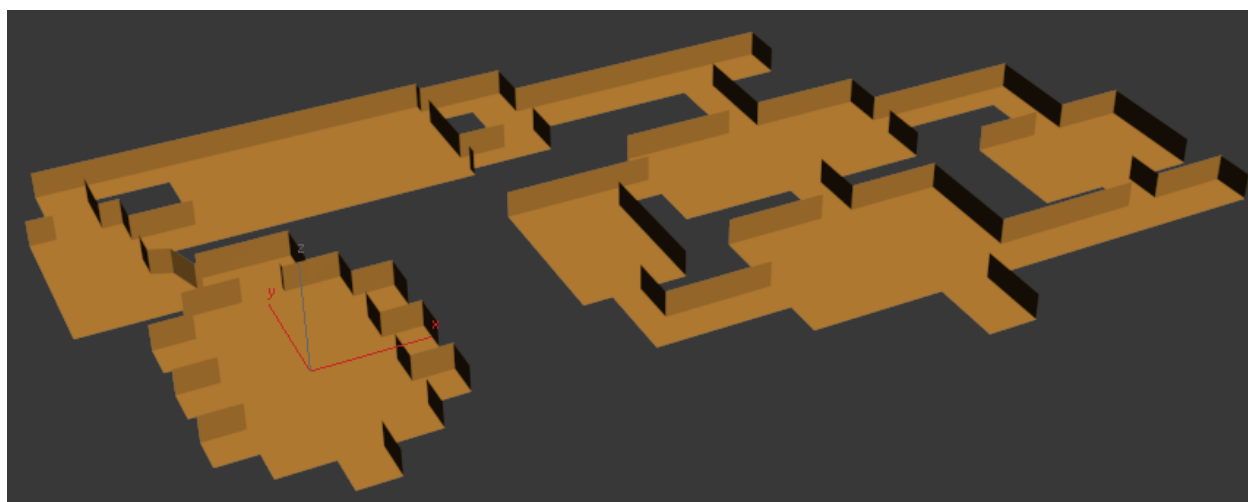


Рис. 2.11 – Форма першого рівня

Другий рівень (рис. 2.12) буде приймати риси замку, який починається з арени. Локація також включає в себе вузькі проходи та ще одну в'язницю в східній стороні рівня.

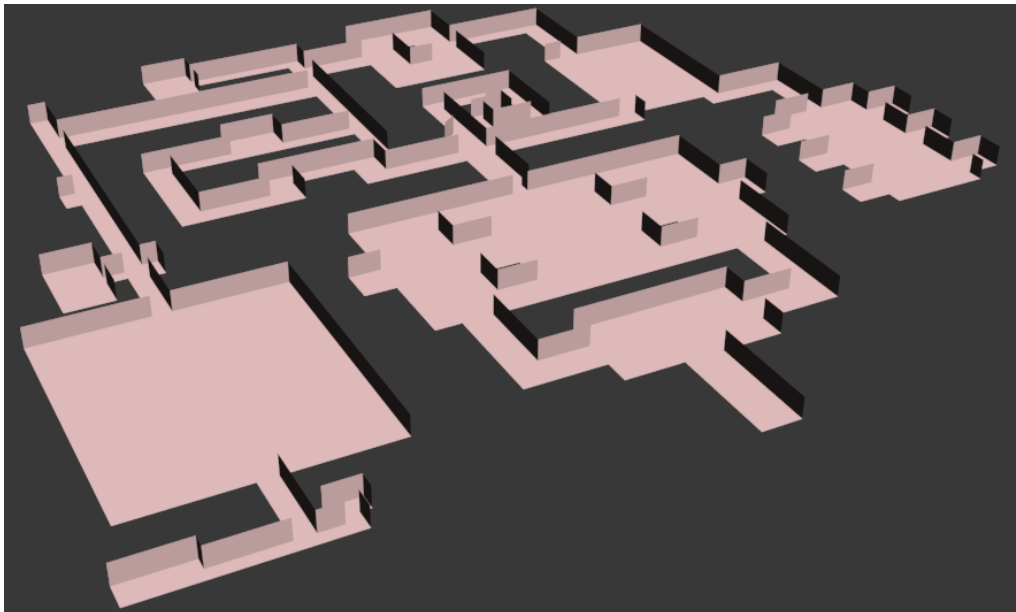


Рис. 2.12 – Форма другого рівня

Третій рівень (рис. 2.13) є останнім та найбільшим серед інших. На ньому присутні довгі вузькі проходи, широкі арени та декілька сюжетних кімнат з предметами екіпування. В кінці поверху запланована важка битва з безліччю ворогами.

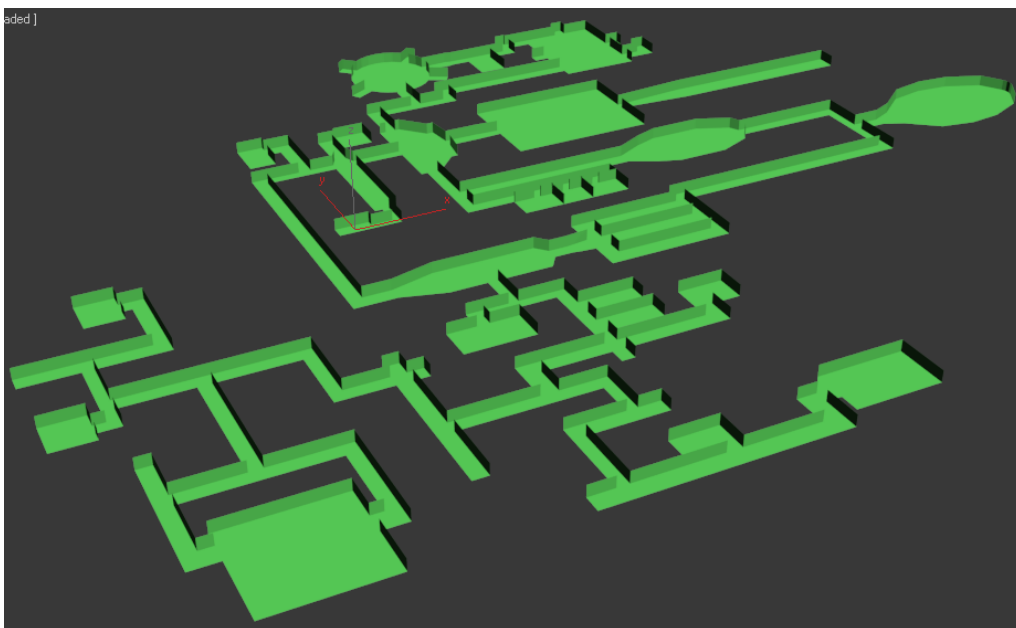


Рис. 2.13 – Форма третього рівня

Окрім сюжетних рівнів є арена (рис. 2.14), на якій гравець зможе проводити необмежену кількість часу.

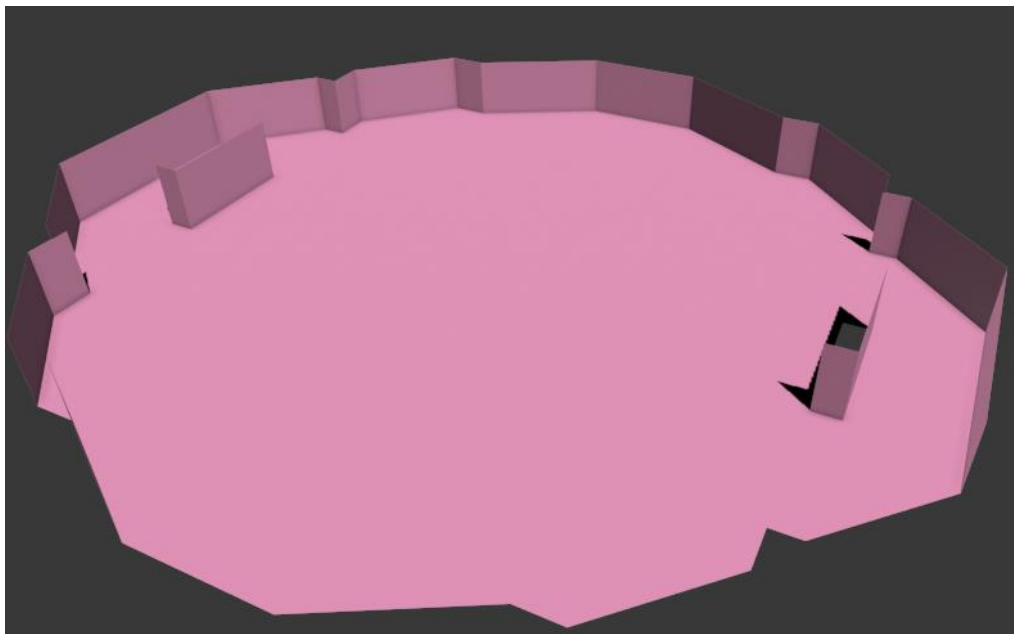


Рис. 2.14 – Форма сцени «Арена»

При досягненні одиниць здоров'я головного героя значення нуль або менше сцена перезавантажується, завантажуючі останній збережений прогрес. Щоб перейти на наступний рівень потрібно дійти до певного тригера в кінці сцени.

Як можа бачити, при розробці моделей для гри дотримувався стиль «low-poly», як і було зазначено раніше.

Для реквізитів та інших візуальних ігрових об'єктів використовувалися моделі із безкоштовних пакунків Asset Store стилю «low-poly». Всі звукові ефекти було взято з безкоштовних інтернет ресурсів та Asset Store.

2.2 Розробка архітектури ігрового додатку

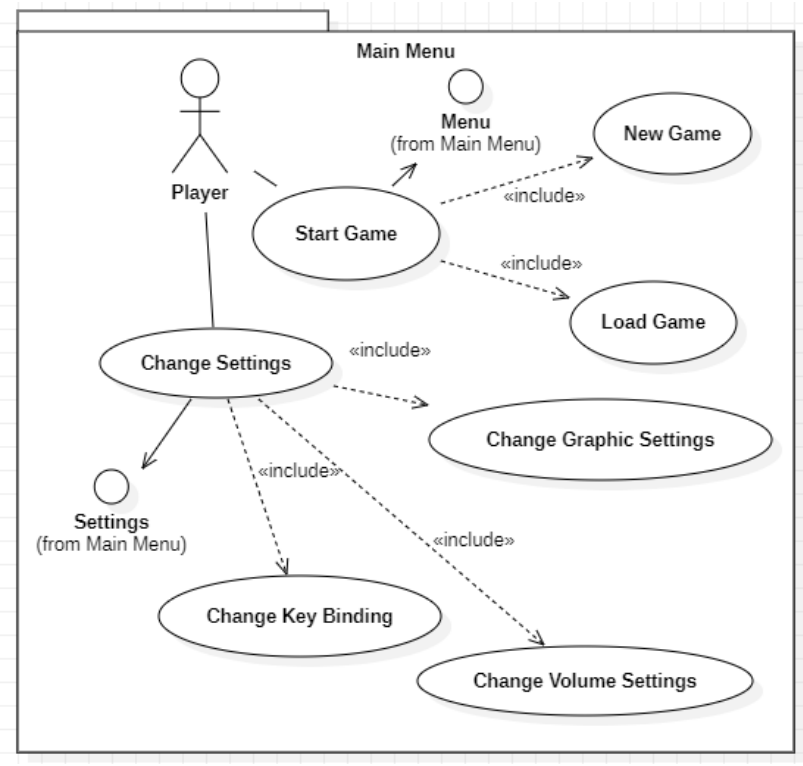
Діаграма варіантів використання – це графічний інструмент моделювання, який використовується для відображення взаємодії між системою та її зовнішніми користувачами через послідовність випадків використання. Він дозволяє уявити, як система взаємодіє з користувачами або зовнішніми системами для досягнення певних цілей.

Діаграми варіантів використання широко використовується на ранніх етапах проектування програмного забезпечення для визначення вимог до системи і встановлення основних функціональних можливостей. Вони

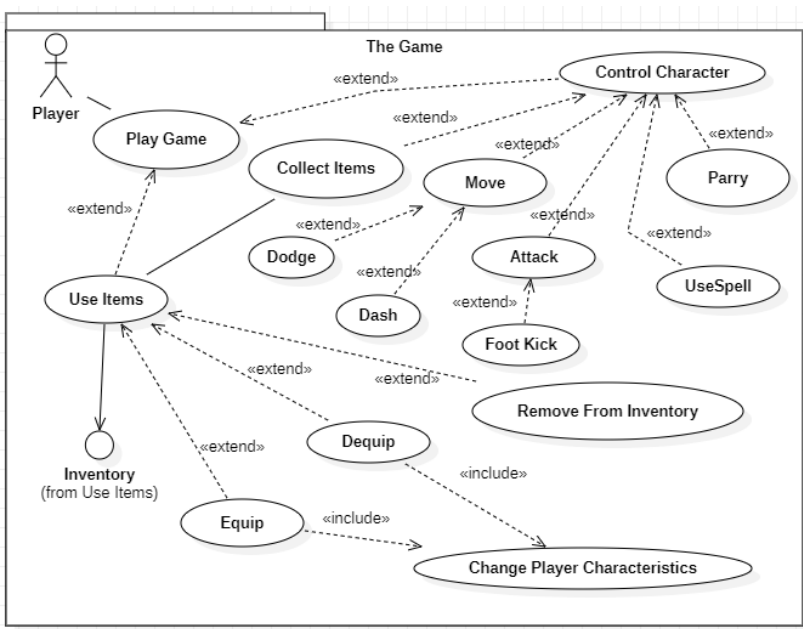
					КН.КР.Б – 122 – 23 – ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		31

допомагають ідентифікувати, описати та організувати випадки, які можуть виникати під час взаємодії з системою.[\[13\]](#)

Побудовану діаграму варіантів використання можна бачити на рисунку 2.15.



а)



б)

Рис. 2.15 – Діаграма варіантів використання:
а – перша частина, б – друга частина

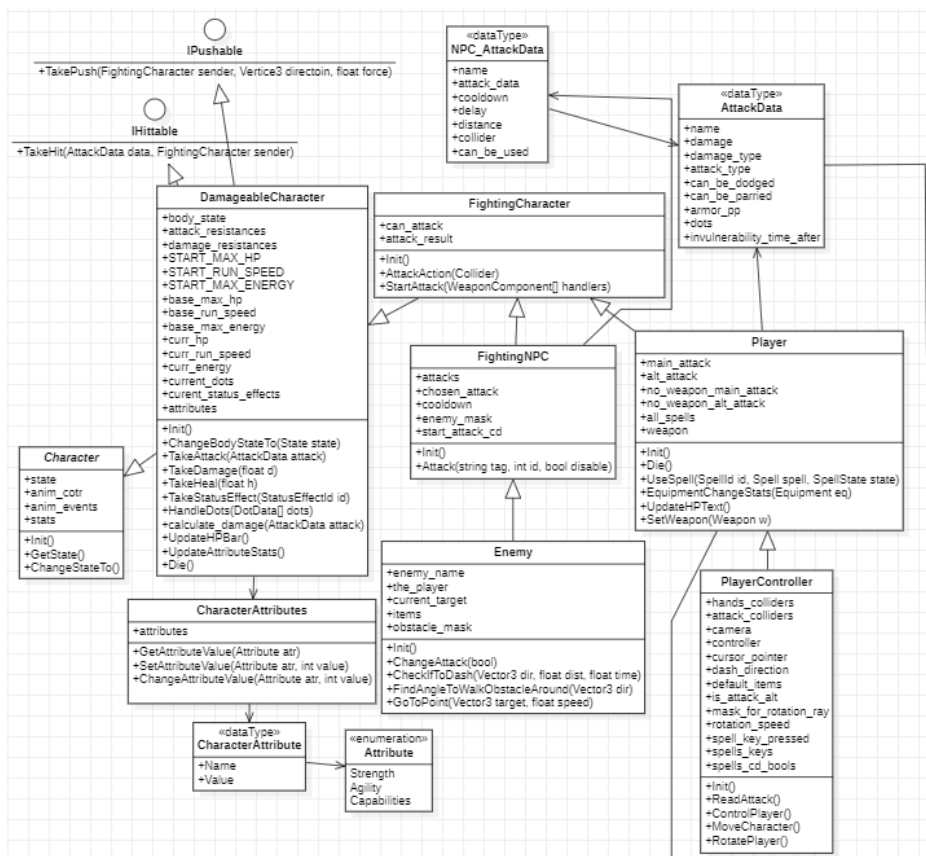
В ній зображено перелік потенційний дій, що може здійснювати актор, в даному випадку – Player, тобто Гравець.

При запуску ігрового додатку в першу чергу користувач попадає в головне меню. На цій стадії він може змінити налаштування або почати гру – нову або продовжити стару.

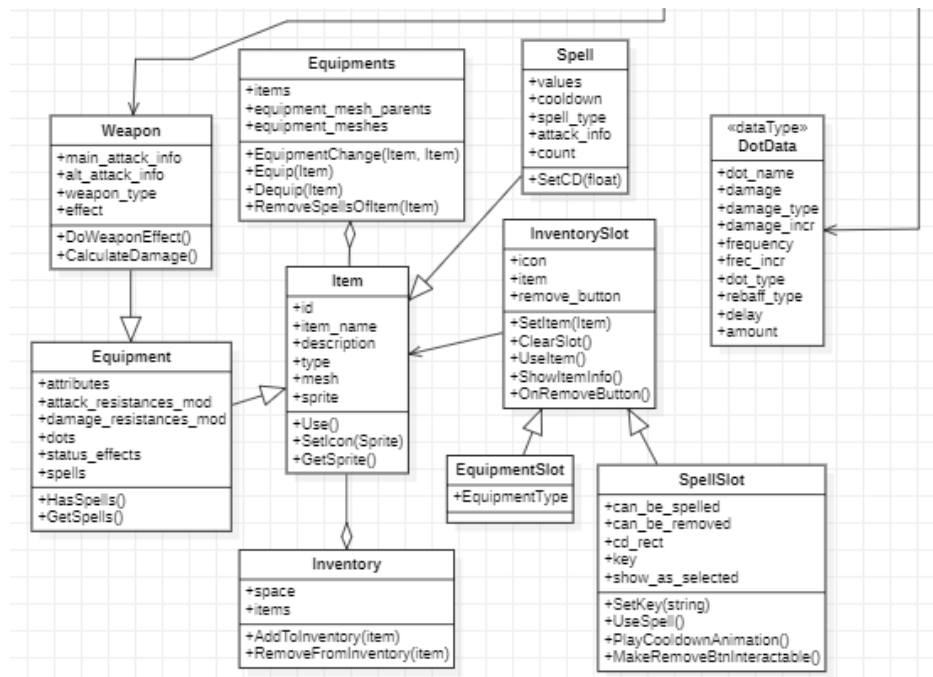
Після завантаження сцени, гравцю надається управління головним героєм, що включає в себе: переміщення персонажа, здійснення атак, удару ногою, використання механік парирування та ухилення, підбирання предметів, використання здібностей, наявних у героя на даний момент, менеджмент інвентарем тощо.

Щоб використати предмет, його спершу потрібно підібрати зі сцени, тому дана залежність наявна на діаграмі. При їх використанні, він або надягається на персонажа, або знімається, якщо вже був надягнутий. Це включає в себе зміну характеристик персонажа.

Найбільшою структурою із великою кількістю зв'язків та наслідування є реалізація функціоналу персонажу (рис. 2.16).



a)



б)

Рис. 2.16 – Діаграма класів персонажів:

а – перша частина, б – друга частина

Опис класів, а також їх полів та методів (у класах використовується англійська назва полів та методів з використанням так званого зміїного регістру – snake_case):

- Клас **Character** – абстрактний клас, в якому реалізовані стани, функції переходу між ними та події, які визиваються при зміні поточного стану.
- Клас **DamageableCharacter** – клас персонажа, в якому реалізований функціонал отримання шкоди та можливість бути відштовхнутим. Містить атрибути, від яких залежать його характеристики, та інформацію про них, такі як: поточна та максимальна кількість життєвих сил, швидкість руху, кількість енергії в даний момент часу та його максимальний обсяг тощо. Також містить в собі інформацію про опори до типів атак та шкоди.
- Клас **CharacterAttributes** – клас, що містить поля та методи для роботи з атрибутами. Так як їх буває декілька типів, то було вирішено відокремити роботу з ними в окремий клас.

- Клас **FightingCharacter** – клас, в якому реалізовані всі необхідні методи, необхідні для здійснення атак, наприклад, функція, яка визивається, коли тригер зброї наштовхується на колайдер ворога, та події, як для опрацювання результату атаки, так і ті, що вмикають та вимикають тригер зброї в певні моменти анімації. Він наслідує від DamageableCharacter тому він також може атаки і приймати.
- Клас **FightingNPC** – клас, що містить в собі функціонал для персонажів які можуть битися, та яких користувач не може контролювати, тобто Non-Playable Character.
- Структура **AttackData** – містить в собі інформацію про атаку: кількість шкоди, її тип, можливість її відбити або ухилитися від неї, її рівень пробиття броні та можливі ефекти, які вона може накладати.
- Структура **NPC_AttackData** – спеціально для класу FightingNPC створена структура даних, що буде зберігати інформацію про кожну атаку персонажа. Цією інформацією може бути: дистанція, з якої можна здійснювати атаку, затримка перед нею, дистанція, при якій ворог фіксує її ціль, логічні поля, що відповідають за такі властивості, як можливість виконання атаки, проміжок часу, через який вона стане доступною, скільки часу ворог відпочиває після та перед наступною атакою тощо.
- Структура **DotData** – структура, що зберігає інформацію про шкоду, яка наноситься поступово, як то: вогонь, отрута, кровотеча тощо. Можна налаштувати затримку перед першим нанесенням шкоди, її кількість, частоту, тип зміни, що настає при повторному нанесенні ефекту, інкремент тощо.
- Клас **Player** наслідує від FightingCharacter та зберігає інформацію про п'ять змінних типу AttackData: звичайна та альтернативна атаки руками, поточні та дані про удар ногою. Також він зберігає

інформацію про всі здібності, які гравець здатен використовувати, та поточну зброю.

- Клас **PlayerController** – клас, що відповідає за управління діями головного героя. Містить в собі всі потрібні для цього поля та методи. Керує станами та переходами між ними. Призначає функції до подій.
- Клас **Enemy** – клас, що відповідає за дані, необхідні ворогам, такі як назва, їх тип, посилення на екземпляр головного героя, поточну ціль, предмети, які випадуть при смерті тощо. Містить методи, необхідні для переміщення та виконання інших дій.
- Клас **Item** – клас з інформацією про предмет. Зберігає його ідентифікатор, назву, опис, тип, 3D-модель, при наявності, та спрайт, який буде зображуватись в графічному інтерфейсі слоту інвентаря.
- Класи **Equipments** та **Inventory** слугують певними «сховищами» для об'єктів типу Item. Вони мають методи додавання та вилучення елементів зі списку, та опрацювання наступних змін. Різниця в тому, що перший – зберігає лише предмети, що надягнуті на головного героя в даний час, а останній – решту.
- Клас **Equipment** – клас, що наслідує від Item. Може бути використаний, в результаті чого надягнеться на персонажа.
- Клас **Weapon** – клас зброї. Зберігає інформацію про шкоду та ефекти, при її використанні. Наслідує від Equipment.
- Клас **InventorySlot** – клас слоту в інвентарі. Має методи додавання та вилучення предмету.
- Клас **SpellSlot** – клас слоту, що зберігає предмет типу **Spell**, який містить в собі дані про заклинання.
- Клас **EquipmentSlot** зберігає предмет типу Equipment.
- Інтерфейси **IPushabe** та **IHittable** відповідальні за однойменні ігрові механіки. Механіка відштовхування реалізована лише у

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

ігровому об'єкті типу MonoBehavior, який наслідує від інтерфейсу IPushable. А задля надання об'єкту можливості реагувати на удари, йому необхідно наслідувати інтерфейс IHittable, при тому, немає потреби бути MonoBehavior – прикладом можуть служити стіни, які завжди повертають результат удару HitWall.

Слід зазначити, що в даному проектуванні було використано декілька патернів.

В розробці ігор дуже часто зустрічається випадок, коли різні об'єкти повинні мати однакові поля або методи, іншими словами, коли вони мають схожий функціонал. Прикладом може служити створення персонажів, або ворогів, з різною поведінкою, характеристиками, метою тощо. Або предметів з різним типом і призначенням. Зазвичай у таких випадках використовують Шаблонний метод. Це поведінковий патерн проектування, який визначає каркас алгоритму, перекладаючи відповідальність на підкласи. Патерн дозволяє підкласам перевизначати кроки алгоритму, не змінюючи його загальної структури[14]. В даному випадку він використовується в класі Character.

Також було використано інший патерн – Міст. Це структурний патерн проектування, який розділяє один або кілька класів на дві окремі ієрархії – абстракцію та реалізацію, дозволяючи змінювати код в одній гілці класів, незалежно від іншої[15]. Він використовується в реалізації предметів та слотів, так як предмети можуть бути різного типу, а слоти – для різних предметів.

Для зручної роботи з ворогами та викликом тригерів було реалізовано поведінковий патерн проектування Спостерігач (рис. 2.17). Він є корисним у випадках, коли необхідно виконати дію у великої або невизначеної кількості екземплярів в певну подію або проміжок часу[16]. В даному випадку клас Enemy одночасно відіграє роль підписника та виконавця за допомогою класу EnemyDieEventHandler, а об'єкт, якого повідомляють про подію – Trigger.

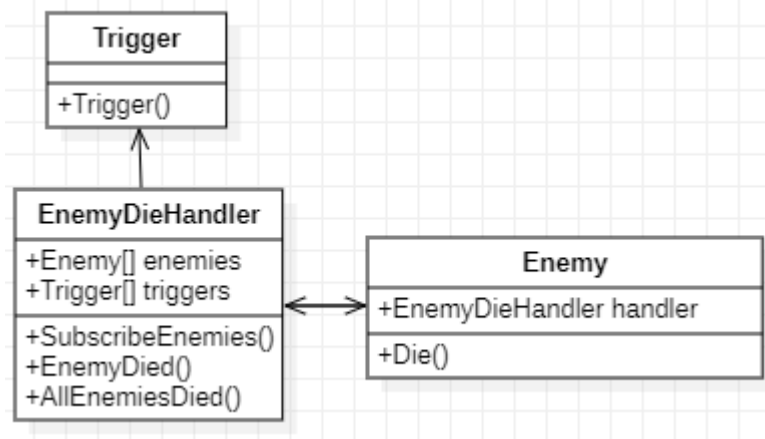


Рис. 2.17 – Реалізація патерну Спостерігач

2.3 Опис роботи з анімаціями

Анімації відіграють одну із найважливіших ролей не тільки в іграх жанру екшн. Вони дозволяють гравцю відчувати такі речі, як: інтенсивність бою, силу удару та його результат, стан персонажа тощо. Для геймплею вони мають дуже велике значення, особливо в даному випадку, коли вони використовуються для нанесення шкоди, виконання здібностей, програвання звукових ефектів, здійснення атак і так далі[17].

Unity надає доволі розширені можливості створення та керування анімаціями. Однією із її дуже корисних функцій є події анімацій – «Animation Event». Це такий механізм, який дозволяє додавати спеціальні мітки або події до анімаційних кліпів. Ці події відпрацьовують в певні моменти відтворення анімації, що відкриває можливості для взаємодії з анімацією і виконання певних дій під час її програвання[18].

Animation Event можна використовувати для таких цілей, як:

- Активація звукових ефектів або музики у відповідності до певних дій або подій в анімації.
- Активація візуальних ефектів або управління параметрами системи частинок.
- Визов методу, який змінює стан персонажа, фізику реквізиту або інші параметри ігрового об'єкта.
- Зміна параметрів анімації, таких як швидкість, масштаб, кольорові ефекти тощо.

Використання даного функціоналу може приймати різноманітні форми залежно від бажань та потреб.

Намагаючись забезпечити адаптивність та гнучкість функціоналу роботи з подіями анімацій, було створено метод AnimEvents, який зберігає функції типу System.Action. Він має декілька відкритих методів, які можуть бути викликаними анімацією на зазначеному кадрі, а вони, в свою чергу, викликають збережені функції. Таким чином, можна керувати подіями анімацій безпосередньо із скрипту анімованого ігрового об'єкту, позбавившись потреби створення індивідуальних методів в класі, що визивається.

В процесі реалізації механіки бою в класі AnimEvents було створено методи StartAttack та AttackOver. Вони викликаються в анімаціях атак на початку та кінці удару. Зазвичай, функції, що призначені на вищезгадані події, відповідальні за ввімкнення та вимкнення тригерів зброї.

Іншою корисною технікою при керуванні анімаціями в Unity є хешування змінних аніматора. Це мікрооптимізація, яка полягає в завчасному зберіганні унікального ідентифікатора, хеш-коду, кожної змінної аніматора, який потім використовується для посилання на цю змінну під час роботи. Це може бути корисно, коли потрібно часто звертатись до змінних аніматора або виконувати операції зміни значень швидко і безпосередньо[\[19\]](#). Приклад використання даного підходу може виглядати:

1. Оголошення змінної аніматора

```
public Animator anim;
```

2. Хешування змінної

```
private int hash_var_name = Animator.StringToHash ("var_name");
```

3. Використання хеш-коду для доступу до змінної аніматора

```
float var_value = anim.GetFloat(hash_var_name);
```

```
anim.SetBool(hash_var_name, true);
```

Серед переваг можна виділити:

- Менша кількість помилок. Хешування змінних дозволяє запобігти орфографічним помилкам або помилковим використанням назв змінних аніматора.
- Менше використання пам'яті. Маніпулювання рядками, замість хеш-кодів, у функції Update залишає після себе мусор, який потрібно прибрати. Це змушує працювати GarbageCollector, в результаті зменшуючи продуктивність гри.

Для зручного використання даного підходу було створено клас AnimController, що відповідає за зберігання хеш-кодів, програвання анімацій та додаткової логіки, яка можлива у всіх дочірніх об'єктах. Наприклад, у класі PlayerAnimController, який керує анімаціями головного героя та переходами між ними, реалізована механіка, що запам'ятовує, з якої сторони, або якою рукою, був виконаний останній удар. Це дає можливість програвати коректну анімацію наступної атаки, навіть якщо головний герой вийшов зі стану, в якому він їх виконує.

Так як головний герой під час руху дивитися в бік розміщення курсору, виникає проблема з підбором анімації. Чудовим рішенням даної проблеми слугує використання BlendTree (рис. 2.18).

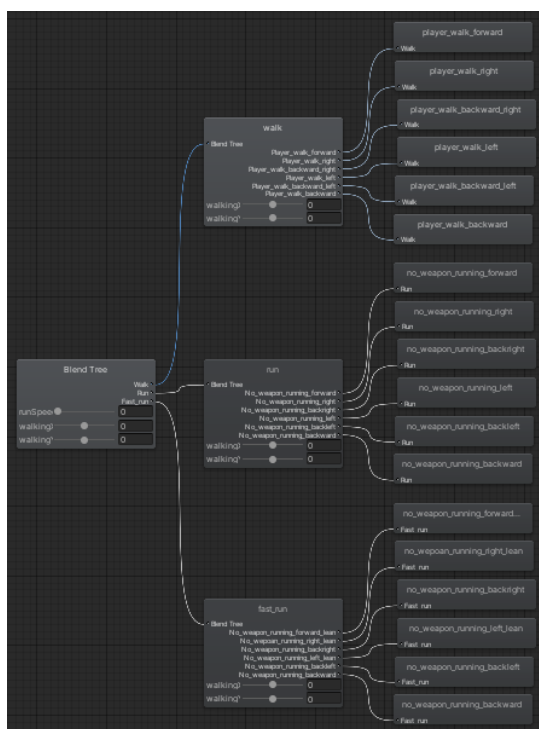


Рис. 2.18 – Використання BlendTree для розрахування анімації бігу

Це механізм, який використовується для керування анімацією на основі комбінації вхідних параметрів. Зазвичай він є корисним для створення складних та плавних анімацій персонажів. BlendTree приймає кілька анімаційних кліпів або станів і змішує їх разом на основі певних змінних.[\[20\]](#)

В даному випадку було створено анімацію бігу вперед, назад, в сторони обличчям та в сторони спиною. Також дані анімації було модифіковано для створення швидкого бігу та спокійної ходьби. В сумі виходить 18 анімацій та 4 BlendTree для керування ними. Змінні, що враховуються при розрахуванні, включають в себе три значення: швидкість бігу, X повороту та Y повороту. Частина коду, в якій розраховуються змінні потовороту виглядають наступним чином:

```
// turn look coordinates to radians and then to degrees
float look_angle = Mathf.Atan2(transform.forward.z, transform.forward.x) *180 / Mathf.PI;
look_angle = (float)System.Math.Round(look_angle, 2);

// same with walk coordinates
float walking_angle = Mathf.Atan2(direction.z, direction.x) * 180 / Mathf.PI;

// calculate difference
float anim_angle = walking_angle - look_angle;
// turn difference angle to coordinates
float x = Mathf.Sin(Mathf.PI * 2 * anim_angle / 360);
float y = Mathf.Cos(Mathf.PI * 2 * anim_angle / 360);
```

Змінна під назвою «direction» зберігає напрямлення бігу персонажа. Як можна бачити, спочатку за допомогою тригонометричних формул розраховується кут погляду та бігу головного героя, отримуємо їх різницю та переводимо в координати. Значення «y» в BlendTree відповідає за вперед/назад, а «x» – за ліво/право.

2.4 Основні етапи реалізації гри

Основною механікою гри є, без сумнівів, управління діями головного героя, що включає в себе: переміщення персонажем по ігровому середовищу, здійснення атак, виконання здібностей та інтеракцію з ігровими об'єктами. Воно здійснюється за допомогою скрипту PlayerController. Залежно від поточного стану героя, він може виконувати різні дії.

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

Основні методи PlayerController

Назва	Призначення
Start	Первинна ініціалізація необхідних змінних персонажа. Налаштування подій, колайдерів зброї та отримання посилань на компоненти.
Update	Викликається кожного кадру рушієм гри. Містить в собі виклик функції ControlCharacter.
LateUpdate	Викликається рушієм наприкінці кожного кадру. Містить виклик методу RotatePlayer() та оновлення позиції ігрового об'єкту, що зображує курсор в ігровому середовищі.
RotatePlayer	Отримує промінь, що йде від камери через точку на екрані до колайдера землі та обертає ігровий об'єкт головного героя по напрямленню до їх зіткнення.
ControlCharacter	Залежно від поточного стану головного персонажа викликає методи MoveCharacter, ReadInteraction, ReadSpells, Spelling та ReadAttack.
MoveCharacter	Відповідає за функціонал переміщення персонажа по середовищу. Зчитує значення клавіш та, залежно від них, рухає героя із зазначеною швидкістю. Може тимчасово переводити персонажа в стан ривка.
ReadInteraction	Дозволяє взаємодіяти з об'єктами на відповідну клавішу.
ReadSpells	Надає можливість використовувати здібності героя.
Spelling	Визивається в стані виконання здібності та дозволяє взаємодіяти з нею за допомогою вводу користувача.
ReadAttack	Перевіряє, чи натиснув гравець клавіши атаки та визиває програвання відповідних анімацій.

Іншими важливими класами, що містять в собі дуже велику кількість реалізованих механік є FightingCharacter та DamageableCharacter. Перший містить в собі необхідні методи та змінні для здійснення атак, а другий – для розрахунку результату удару та отримання шкоди. Від цих класів наслідують такі важливі елементи гри як Player та PlayerController, де перший має всю необхідну інформацію для гравця, а останній – для керування його персонажем.

Основні методи FightingCharacter

Назва	Призначення
AttackAction	Функція, що викликається при зіткненні триггеру зброї з колайдером тіла.
SetAttackEvents	Встановлення подій анімацій при початку та кінці удару.

Основні методи DamageableCharacter

Назва	Призначення
Init	Ініціалізація необхідних змінних, налаштування характеристик залежно від значень атрибутів, функцій подій тощо.
TakeHit	Відповідає за розрахування результату удару, залежно від поточного стану персонажу та властивості атаки.
TakeAttack	Розраховує кінцеве значення шкоди, враховуючи пробиваність атаки, міцність броні та опір до типу шкоди.
TakeDamage	Змінює значення здоров'я, викликає відповідні події та ефекти.
TakePush	Змушує персонажа відштовхнутись від джерела з певною силою.
TakeStunn	Змушує персонажа перейти в стан, в якому він не здатен виконувати ніякі дії.
HandleDots	Відповідальний за логіку шкоди, що наноситься поступово.

Кожен персонаж має власні опори до різних тип атак та шкоди. При попаданні зброї по тілу розраховується результат удару, і, якщо він успішний, наступні розрахунки будуть пов'язані із визначенням отриманих пошкоджень. Формула розрахунку із врахуванням броні виглядає наступним чином:

$$d * (1 - \frac{A_p - A_{pp}}{A_{max}})$$

Де

- d – початкове значення шкоди
- A_p – опір до атаки
- A_{pp} – ігнорування опору цією атакою
- A_{max} – максимальне значення різниці

Можна помітити, що якщо різниця в дробі буде приймати значення більше за A_{max} – шкода буде від’ємною, тому перед розрахунком наявна перевірка цього значення. Проте вона відсутня при розрахунку опору до шкоди, взявши за максимальне значення число 100. Це означає, що якщо у персонажа опір до, нехай, отрути приймає значення 200 – він буде від неї лікуватися.

Збереження ігрового процесу відбувається за допомогою бінарної серіалізації. Структуру об’єкта, що зберігає інформацію про ігрову сесію, зображено в таблиці 2.4.

Таблиця 2.4

Поля структури SaveData

Назва	Тип	Призначення
CurrentHP	float	Зберігає значення поточного здоров’я головного героя.
CurrentEN	float	Зберігає значення поточної енергії головного героя.
Upgrades	int[]	Зберігає значення покращень атрибутів головного героя.
SceneId	int	Зберігає ідентифікатор сцени.
ArenaLevel	int	Зберігає норм ітерації появи ворогів на арені.
Inventory	string[]	Зберігає назви предметів, що були в інвентарі під час збереження.

Продовження таблиці 2.4

EquipedEquipments	string[]	Зберігає назви екіпувань, що були налягнуті на головного героя під час збереження. Назви можуть бути пустими рядками.
EquipedSpell	string[]	Зберігає назви здібностей, що були готові до використання під час зберігання. Назви можуть бути пустими рядками.

Структуру об'єкта, що зберігає інформацію про налаштування системи, можна бачити в таблиці 2.5.

Таблиця 2.5

Поля структури SettingsData

Назва	Тип	Призначення
ResIndex	int	Зберігає ідентифікатор розширення екрану.
QuilityIndex	int	Зберігає ідентифікатор якості графіки.
Volume	float	Зберігає значення гучності звуку.
Fullscreen	bool	Повноекранний режим.
SpellKeys	int[]	Зберігає клавіши для використання здібностей.

Для оптимізації використання ігрових об'єктів, життєвий цикл яких займає доволі мало часу, проте їх створення є частим явищем, було реалізовано так званий пул об'єктів, який дозволяє використовувати їх повторно. Основна логіка полягає в тому, що замість видалення об'єкта зі сцени, що активізує роботу Garbage Collector'a і зменшить продуктивність, він просто переходить в стан неактивного до свого наступного використання. Об'єктами що так зберігаються є снаряди, різноманітні візуальні ефекти та вороги на арені. Основні методи класу Obesjtpool можна бачити в таблиці 2.6.

Основні методи класу ObjectPool

Назва	Призначення
Awake	Використовується для початкової ініціалізації списку ігрових об'єктів.
Start	Створює зазначену кількість ігрових об'єктів та додає їх до списку.
GetPooledObject	Повертає неактивний ігровий об'єкт.
AddInstanceOfObjectToList	Створює ігровий об'єкт та додає його до списку у випадку, якщо всі наявні є активними.

Висновки до другого розділу

В результаті другого розділу було складено дизайн-документ, в якому описано управління діями головного героя, наявні ігрові механіки та всі створенні тривимірні моделі.

Було описано створену архітектуру програмного продукту за допомогою використання діаграми варіантів використання та діаграм класів із зазначенням всіх реалізованих патернів.

Також було описано класи, що використовувалися для: роботи з анімацією та їх подіями; керування діями головного героя; нанесення та отримання шкоди та інших важливих механік; пулу об'єктів.

РОЗДІЛ 3. ОПИС РОБОТИ З ПРОГРАМНИМ ДОДАТКОМ ТА ЙОГО ТЕСТУВАННЯ

3.1 Опис роботи з програмним додатком

При запуску гри гравець бачить перед собою головне меню (рис. 3.1), яке надає користувачу наступний вибір дій: почати гру, перейти в налаштування або вийти з програмного додатку.



Рис. 3.1 – Головне меню

Налаштування гри (рис. 3.2) включає в себе визначення таких властивостей, як: розширення екрану, рівень якості графіки, повний екран, гучність звуку та клавіші, які відповідають за слоти здібностей.



Рис. 3.2 – Налаштування гри

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

Перед початком гри гравцю надається вибір слоту для збереження прогресу (рис. 3.3).

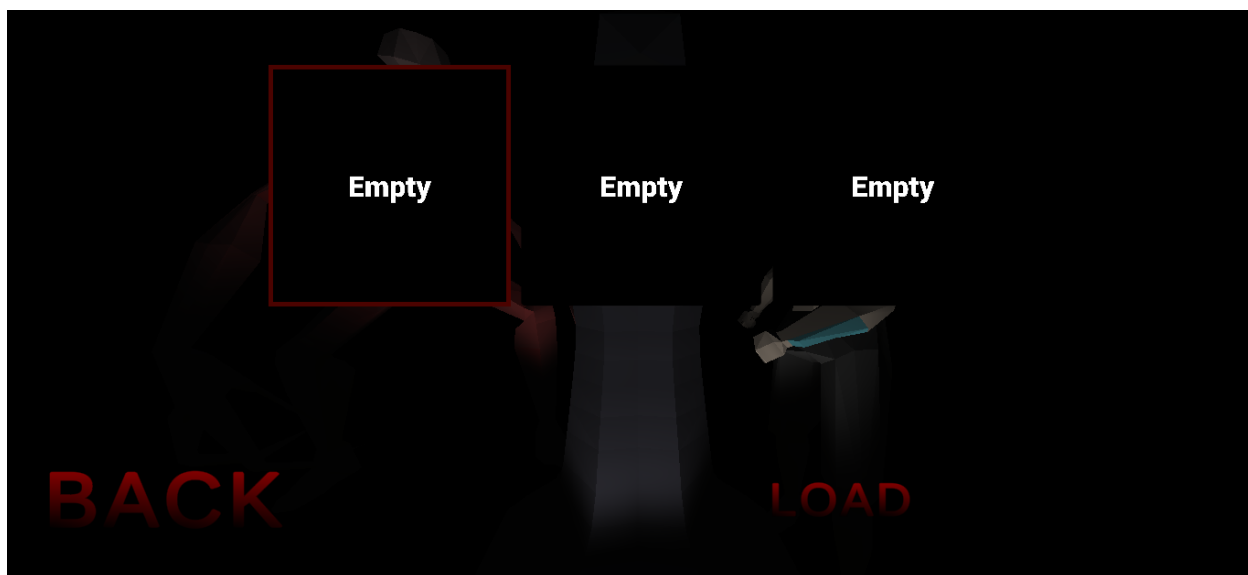


Рис. 3.3 – Вибір слоту збереження прогресу гри

При наявності вже існуючих ігрових сесій вони будуть зображені в своєму слоті (рис. 3.4).

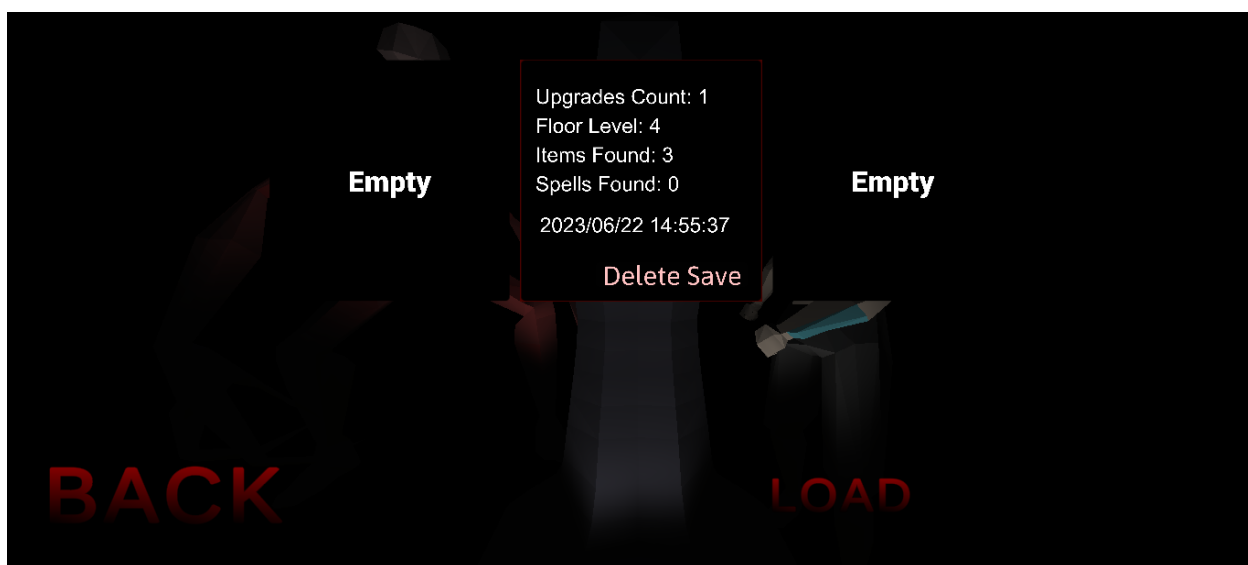


Рис. 3.4 – Слот із збереженою ігровою сесією

При виборі пустого слоту наступною дією буде вибір режиму гри: арена чи сюжетний (рис. 3.5). Від цього вибору буде залежати ідентифікатор сцени, що буде завантажена на початку гри. Різниця між ареною та сюжетною кампанією полягає в тому, що в першому випадку вся гра відбувається на одній локації, коли як в другому гравець здатен перейти на наступну сцену.



Рис. 3.5 – Вибір режиму гри

Для того, щоб повернутись в головне меню слід скористатися паузою, та обрати необхідний варіант (рис. 3.6).

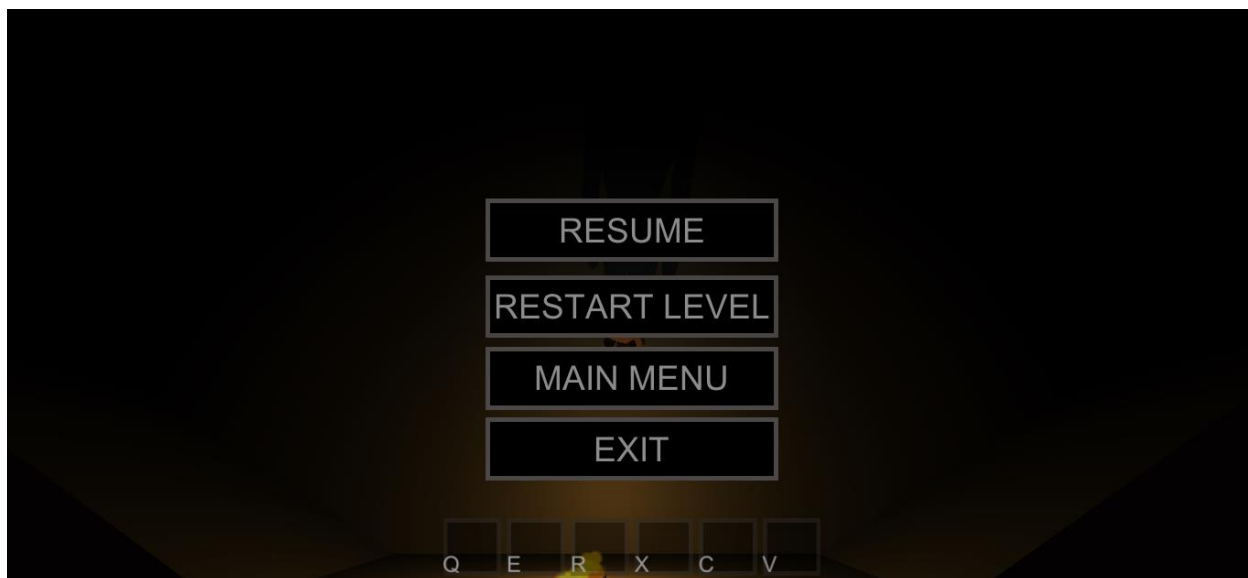


Рис. 3.6 – Пауза під час гри

Кожного третього рівня на арені та в особливих моментах в сюжетному режимі гравець зможе покращити свого персонажа за допомогою збільшення одного з його атрибутів. Користувацький інтрефейс даного вікна можна бачити на рисунку 3.7. Користувач не має можливості обрати атрибут першу секунду після бою, адже часто буває випадок, коли гравець продовжує натискати на клавіши мишки деякий час після смерті ворога.



Рис. 3.7 – Вікно збільшення атрибуту персонажа

Також на рисунку.3.7 можна помітити візуальне зображення кількості здоров'я та енергії у гравця.

Арену, на якій гравець буде проводити свій час, можна побачити на рисунку 3.8.

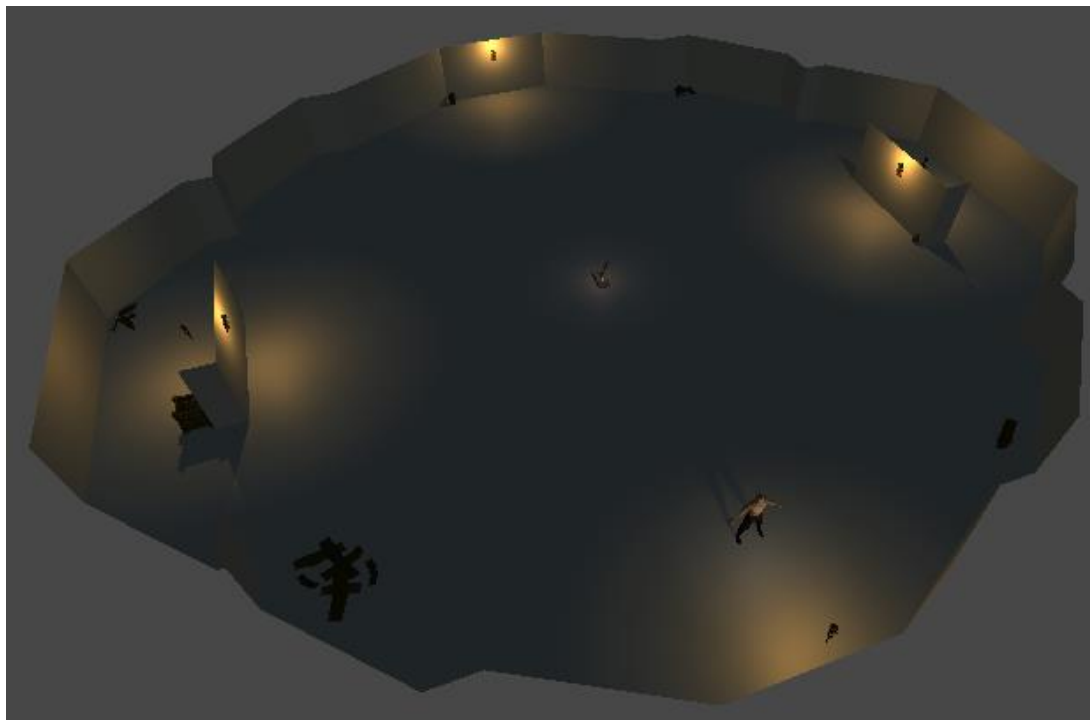


Рис. 3.8 – Зовнішній вигляд сцени «Арена»

В центрі кімнати розташований ричаг, при використанні який створює ворогів, кількість яких відповідає ітерації арени. Для того, щоб гравець зрозумів, що йому робити, головний герой каже фразу «I guess I should kick the lever» (рис. 3.9).

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

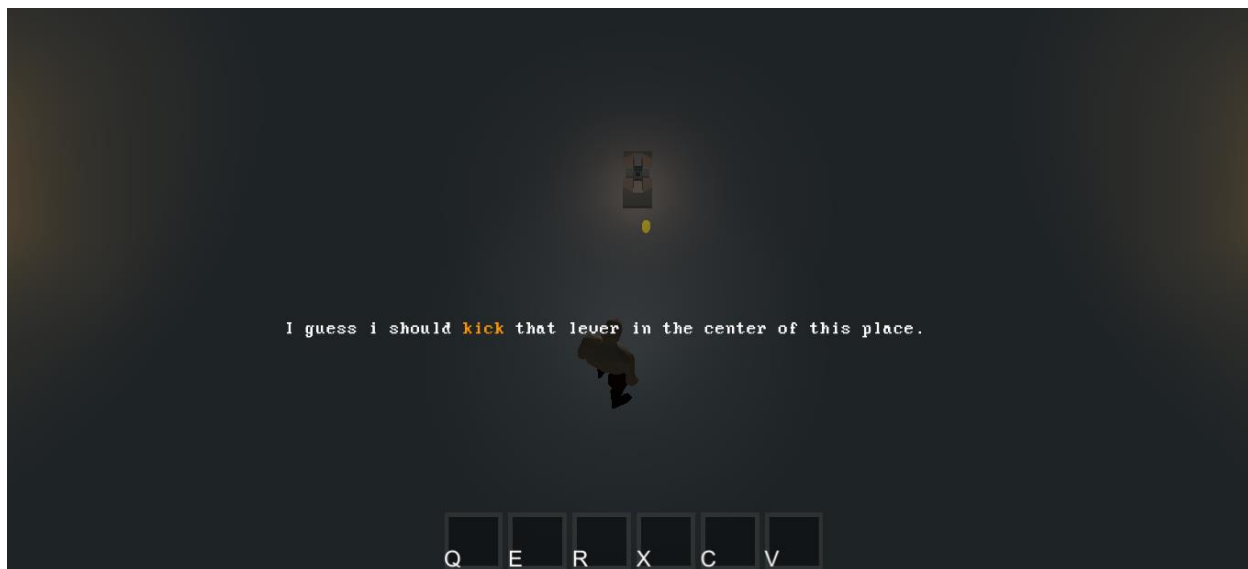


Рис. 3.9 – Фраза головного героя, що підказує гравцю наступні дії

При завантаженні першої сцени сюжетного режиму (рис. 3.10) головний герой прокидається у тюремній кімнаті та говорить «Time to get out of here» та натякає, що він здатен на те, щоб вибити клітку ногою.



Рис. 3.10 – Початкова сцена сюжетного режиму

Такі фрази головний герой каже на протязі всієї сюжетної лінії (рис. 3.11). Вони можуть бути жартівними, серйозними або просто відігравати роль підказки. Зазвичай, персонаж, яким керує гравець, любить коментувати навколишнє середовище, створюючи цим атмосферу гри з кимось. Механіка фраз також може бути використана і у ворогів. Так, наприклад, при виході зі тюремного блоку, один з ворогів коментує дану ситуацію в негативному тоні.



Рис. 3.11 – Приклад фрази сюжетної лінії

Інтерфейс було розроблено, намагаючись надати йому риси інтуїтивно зрозумілого.

3.2 Тестування роботи програмного забезпечення

Проведено тестування гри для виявлення та усунення дефектів.

Таблиця 3.1

Результати тестування

Сцена	Вхідні дані	Очікувані дані	Спостережені дані
Головне меню	Натиск на кнопку початку гри	Відкривається меню вибору слота збереження	Відкривається меню вибору слота збереження
	Натиск на кнопку завантаження збереженої гри	Завантажується гра, додаються предмети в інвентар, видаляються підняті предмети зі сцени тощо.	Завантажується гра, додаються предмети в інвентар, видаляються підняті предмети зі сцени тощо.
	Натиск на кнопку видалення збереженої гри	Уточнення у користувача щодо впевненості у своїх діях. Очищення слоту збереження	Уточнення у користувача щодо впевненості у своїх діях. Очищення слоту збереження
	Натиск на кнопку налаштувань	Відкривається меню налаштувань	Відкривається меню налаштувань

	Натиск на кнопку виходу	Ігровий додаток завершує роботу	Ігровий додаток завершує роботу
Меню налаштувань	Зміна дозволу екрана	Розширення екрана змінюється	Розширення екрана змінюється
	Зміна режиму екрана	Змінюється режим на повноекранний або ні	Змінюється режим на повноекранний або ні
	Зміна графіки	Змінюється графіка на вибрану зі списку	Змінюється графіка на вибрану зі списку
	Рух скролера, що відповідає за гучність	Змінюється рівень гучності	Змінюється рівень гучності
	Вибір клавіши для здібності із списку	Клавіша, що відповідає за виконання здібності в обраному слоті змінюється	Клавіша, що відповідає за виконання здібності в обраному слоті змінюється
	Натиск на кнопку повернення до головного меню	Зміни налаштувань зберігаються	Зміни налаштувань зберігаються
Меню паузи	Натиск кнопки продовження гри	Меню паузи закривається, гра продовжується	Меню паузи закривається, гра продовжується
	Натиск на кнопку перезавантаження рівня	Рівень перезавантажується і завантажує останні збережені дії	Рівень перезавантажується і завантажує останні збережені дії
	Натиск кнопки повернення до головного меню	Завантажується перша сцена з головним меню	Завантажується перша сцена з головним меню
	Натиск кнопки вихода з гри	Програмний додаток припиняє роботу	Програмний додаток припиняє роботу
Основна сцена	Натиск кнопок WASD	Персонаж рухається	Персонаж рухається

	Зіткнення з перешкодою	Персонаж зупиняється	Персонаж зупиняється
	Натиск ЛКМ або ПКМ	Здійснення атак	Здійснення атак
	Натиск Space	Здійснення удару ногою	Здійснення удару ногою
	Натиск ЛКМ або ПКМ біля ворога	Якщо зброя наткнулась на ворога – нанесення шкоди	Якщо зброя наткнулась на ворога – нанесення шкоди
	Натиск Tab	Відкривається інвентар	Відкривається інвентар
	Натиск L Shift під час руху	Персонаж робить ривок	Персонаж робить ривок
	Натиск L Shift не під час руху	Персонаж робить ухилення	Персонаж робить ухилення
	Натиск F біля предмету	Предмет додається в інвентар. Якщо є текст – предмет додається в інвентар після виводу всього тексту або після зазначеної фрази.	Предмет додається в інвентар. Якщо є текст – предмет додається в інвентар після виводу всього тексту або після зазначеної фрази.
	Натискання ПКМ по предмету в інвентарі	Предмет використовується	Предмет використовується
	Екіпування предмету	При наявності, створюється тривимірна модель на головному герою. Предмет вносить свої зміни у значення атрибутів та опорів. В інвентар додаються здібності, що надає цей предмет.	При наявності, створюється тривимірна модель на головному герою. Предмет вносить свої зміни у значення атрибутів та опорів. В інвентар додаються здібності, що надає цей предмет.

	Здійснення удару ногою по ричагу на карті арени	Якщо всі вороги були переможені - починається нова хвиля	Якщо всі вороги були переможені - починається нова хвиля
	Перебування ворога з мечем за стіною від героя	Ворог обходить стіну	Ворог проходить стіну наскрізь
	Перебування ворога «зомбі» на іншій стороні об'єкту інтер'єру «стіна клітки»	Ворог намагається знайти шлях до головного героя проходячи через стіну	Ворог ігнорував стіну та проходить її наскрізь
	Збереження гри в кінці переможеної хвилі на арені або наприкінці сюжетного рівня	Зберігаються наступні параметри: к-сть здоров'я та енергії в головного героя, інформація про інвентар, номер сцени та рівня, якщо наявний	Зберігаються наступні параметри: к-сть здоров'я та енергії в головного героя, інформація про інвентар, номер сцени та рівня, якщо наявний
	Використання здібності «Stun»	Всі вороги в радіусі застигають на певний проміжок часу	Вороги застигали, та якщо попасти в особливий момент – навіть застигші йшли в сторону гравця

При тестуванні було виявлено ряд помилок. Наприклад, вороги проходили скрізь стіни та об'єкти інтер'єру. Ця проблема могла бути вирішеною за допомогою розташування інтер'єру або моделювання такої геометрії мапи, що не створювали би необхідність персонажу обійти перешкоду. Проте, було вирішено розробити алгоритм, що використовує промені, задля виявлення наявності перешкод до цілі. Якщо коротко, коли ворог підходить до стіни, він перевіряє за допомогою Raycast променя існування перешкод на шляху. Залежно від цього, шукається кут, на який ворогу потрібно повернути. Якщо перешкод на шляху немає – ворог йде в сторону головного героя або, залежно від логіки окремого ворога, виконує

зазначені дії. Таким чином, вірніше сказати, що алгоритм не шукає шлях до цілі, а просто намагається обійти перешкоду.

Майже рівною по складності проблемою були тонкощі переходів між станами ворога. Дуже часто бувають випадки, коли необхідно відкласти певну дію на певний проміжок часу. Корисним інструментом для цього слугуватимуть корутини. В них можна зазначити не лише час, який потрібно почекати перед виконанням наступної дії, а й відкласти виконання, наприклад, на кінець кадру. Для запобігання їх «накладання» один на одного, явище, коли одна дія суперечить іншій, що іноді важко передбачити, було створено змінну, яка зберігає посилання на корутин, що використовується задля виконання дій через певний проміжок часу. Таким чином завжди є посилання на відкладену дію, що надає додатковий рівень контролю станами та переходами.

Проблема з проходженням ворогів через перешкоди зумовлювалася невірним розташуванням колайдерів.

Висновки до третього розділу

В результаті написання третього розділу було описано користувацький інтерфейс гри в: головному меню, меню налаштування параметрів, меню паузи під час гри, робота інтерфейсу тощо. Було наведено приклади фраз головного героя, що використовувалися для надання підказки гравцю та із розважальних цілей.

Також було описано процес тестування програмного додатку та його результати – було виявлено певний ряд дефектів та описано шляхи їх вирішення.

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

ВИСНОВКИ

В результаті роботи над кваліфікаційним проектом було розроблено ігровий додаток, що переймає всі риси жанру екшн та надає користувачу можливість здійснювати всі вищезгадані ігрові механіки. У гравця в асортименті є чотири сцени, придатних до гри, та два режими. Розроблено п'ятьох ворогів, кожен з яких відрізняється за своїми властивостями, поведінкою на полі бою та візуальними характеристиками. Головний герой може приміряти на собі двадцять чотири предмети екіпування та виконати одинадцять унікальних здібностей. Створені тривимірні моделі дотримуються візуального стилю «low-poly» та не навантажують систему своїм перебуванням на екрані. Користувацький інтерфейс розроблено так, щоб він виглядав інтуїтивно зрозумілим та привабливим.

Задля досягнення вищезгаданих результатів було:

1. Проведено аналіз існуючих ігрових додатків аналогів за схожими рисами геймплею та визначено основні позитивні риси та недоліки.
2. Складено концепт документ, в якому зазначено жанр, основну аудиторію, описано геймплей, сюжет тощо.
3. Складено дизайн документ, в якому описано управління головним героєм, ігрові механіки, створено тривимірні моделі та анімовано персонажів. Створено їх концепцію та візуальний вигляд.
4. Розроблено архітектуру програмного додатку та зображено її за допомогою діаграм варіантів використання та діаграми класів.
5. Створено класи для роботи з анімаціями, управління діями головного героя, збереження прогресу ігрової сесії та описано їх методи у відповідному розділі.
6. Протестовано роботу ігрового додатку

Реалізований програмний комплекс готовий до використання та може бути розширений у майбутньому новими механіками, ворогами, предметами та здібностями.

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

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. What electronic games can teach us. *Knowable Magazine* | *Annual Reviews*. 2023 URL: <https://knowablemagazine.org/article/mind/2019/video-games-educational-benefits>
2. West J., Gould-Wilson J. 25 best action games you can play right now. *gamesradar*. 2023 URL: <https://www.gamesradar.com/best-action-games/>
3. Ігрові жанри | Безкоштовний онлайн-курс від компанії QATestLab. *Безкоштовний онлайн-курс від компанії QATestLab* | Головна сторінка. 2023 URL: <https://training.qatestlab.com/blog/technical-articles/games-genres/>
4. Battlerite - Games. *Battlerite*. 2023 URL: <https://www.battlerite.com/>
5. Akane. *Latest* | *Official PlayStation™Store US*. 2023 URL: https://store.playstation.com/en-us/product/UP7249-CUSA34397_00-7550698100197999
6. Hyper Light Drifter Wiki. *Hyper Light Drifter Wiki* | *Fandom*. 2023 URL: https://hyperlightdrifter.fandom.com/wiki/Hyper_Light_Drifter_Wiki
7. Жанри комп'ютерних ігор: список. Класифікація комп'ютерних ігор за жанрами. *UAEU Українська ТОП Газета*. 2023 URL: <https://uaeu.top/digital-online/zhanri-komp-yuternikh-igor-spisok-klasifikatsiya-komp-yuternikh-igor-za-zhanrami.html>
8. WebCite query result. *WebCite*. 2023 URL: <https://webcitation.org/6JAsnQ3d9?url=http://docs.unity3d.com/Documentation/Manual/HOWTO-MonoDevelop.html>
9. UnityScript's long ride off into the sunset | *Unity Blog*. *Unity Blog*. 2023 URL: <https://blog.unity.com/community/unityscripts-long-ride-off-into-the-sunset>
10. Unity vs Unreal Engine: Differences and Performance Comparison. *Kevuru Games*. 2023 URL: <https://kevrugames.com/blog/unity-vs-unreal-engine-pros-and-cons/>
11. Getting Started. *Pathfinder Roleplaying Game Reference Document*. 2023 URL: <http://legacy.aonprd.com/coreRuleBook/gettingStarted.html>

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

12. Dictionary.com. What Does 'DOT' Mean? | Acronyms by Dictionary.com. Dictionary.com. 2023 URL: <https://www.dictionary.com/e/acronyms/dot/>
13. What is Use Case Diagram?. *Ideal Modeling & Diagramming Tool for Agile Team Collaboration.* 2023 URL: <https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-use-case-diagram/>
14. Шаблонний метод. *Refactoring and Design Patterns.* 2023 URL: <https://refactoring.guru/uk/design-patterns/template-method>
15. Міст. *Refactoring and Design Patterns.* 2023 URL: <https://refactoring.guru/uk/design-patterns/bridge>
16. Спостерігач. *Refactoring and Design Patterns.* 2023 URL: <https://refactoring.guru/uk/design-patterns/observer>
17. Gaming Animation: Uses of Animation in the Gaming Industry. *Juego Studio.* URL: <https://www.juegostudio.com/blog/gaming-animation>
18. Unity - Manual: Using Animation Events. *Unity - Manual: Unity User Manual* 2021.3 (LTS). URL: <https://docs.unity3d.com/Manual/script-AnimationWindowEvent.html>
19. Unity - Scripting API: Animator.StringToHash. *Unity - Manual: Unity User Manual* 2021.3 (LTS). URL: <https://docs.unity3d.com/ScriptReference/Animator.StringToHash.html>
20. Unity - Scripting API: BlendTree. *Unity - Manual: Unity User Manual* 2021.3 (LTS). URL: <https://docs.unity3d.com/ScriptReference/Animations.BlendTree.html>
21. Unity - Scripting API: Rigidbody. *Unity - Manual: Unity User Manual* 2021.3 (LTS). URL: <https://docs.unity3d.com/ScriptReference/Rigidbody.html>
22. Unity - Scripting API: Resources. *Unity - Manual: Unity User Manual* 2021.3 (LTS). URL: <https://docs.unity3d.com/ScriptReference/Resources.html>

					KH.KP.Б – 122 – 23 – ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		
						59

23. Unity - Scripting API: Random. *Unity - Manual: Unity User Manual 2021.3 (LTS)*. URL: <https://docs.unity3d.com/ScriptReference/Random.html>
24. Unity - Scripting API: RaycastHit. *Unity - Manual: Unity User Manual 2021.3 (LTS)*. URL: <https://docs.unity3d.com/ScriptReference/RaycastHit.html>
25. Unity - Scripting API: Ray. *Unity - Manual: Unity User Manual 2021.3 (LTS)*. URL: <https://docs.unity3d.com/ScriptReference/Ray.html>
26. Unity - Scripting API: Sprite. *Unity - Manual: Unity User Manual 2021.3 (LTS)*. URL: <https://docs.unity3d.com/ScriptReference/Sprite.html>
27. UI.Button - Unity Scripting API. *Unity - Manual: Unity User Manual 2021.3 (LTS)*. URL: <https://docs.unity3d.com/ru/2018.4/ScriptReference/UI.Button.html>
28. Unity - Scripting API: GameObject.SetActive. *Unity - Manual: Unity User Manual 2021.3 (LTS)*. URL: <https://docs.unity3d.com/ScriptReference/GameObject.SetActive.html>
29. Unity - Scripting API: Transform.localEulerAngles. *Unity - Manual: Unity User Manual 2021.3 (LTS)*. URL: <https://docs.unity3d.com/ScriptReference/Transform-localEulerAngles.html>
30. Unity - Scripting API: Coroutine. *Unity - Manual: Unity User Manual 2021.3 (LTS)*. URL: <https://docs.unity3d.com/ScriptReference/Coroutine.html>
31. URL: <https://docs.unity3d.com/ScriptReference/WaitForEndOfFrame.html>
32. Unity - Manual: Animator Controller. *Unity - Manual: Unity User Manual 2021.3 (LTS)*. URL: <https://docs.unity3d.com/Manual/class-AnimatorController.html>
33. Unity - Manual: Animation transitions. *Unity - Manual: Unity User Manual 2021.3 (LTS)*. URL: <https://docs.unity3d.com/Manual/class-Transition.html>

34. Unity - Manual: Animation States. *Unity - Manual: Unity User Manual 2021.3 (LTS)*. URL: <https://docs.unity3d.com/Manual/class-State.html>

					КН.КР.Б – 122 – 23 – ПЗ	Арк.
						61
Змн.	Арк.	№ докум.	Підпис	Дата		

ДОДАТКИ

Найбільш важливі функції

Важливі методи класу **PlayerController**:

```
private void Start()
{
    base.Init();
    Cursor.visible = false;
    camera = Camera.main;
    anim_contr = GetComponentInChildren<PlayerAnimController>();
    anim_events = GetComponentInChildren<AnimEvents>();

    if(LoadInfo.spell_keys == null) {
        spell_keys = new KeyCode[6] { KeyCode.Q, KeyCode.E, KeyCode.R,
        KeyCode.X, KeyCode.C, KeyCode.V};
    }
    else {
        spell_keys = LoadInfo.spell_keys;
    }

    for(int i = 0; i < _spell_slots.Length; i++)
    {
        _spell_slots[i].SetKey(spell_keys[i].ToString());
    }

    SetHandsCollidersAsCurrent();
    SetFootCollider();

    for(int i =0;i<default_items.Length;i++){
        //Equipments.GetInstance().Equip(default_items[i]);
        Inventory.GetInstance().AddToInventory(default_items[i]);
    }

    OnStateChangeEvent += StateChange;

    OnStartAttackEvent += WeaponRotationModifier;
    AfterAttackEvent += WeaponRotationModifierBack;

    OnPushEvent += OnPushStunn;
    OnStunnEvent += OnPushStunn;

    AfterPushEvent += AfterPushStunn;
    AfterStunnEvent += AfterPushStunn;

    OnParryEvent += Parry;
    OnBeingParriedEvent += BeingParried;

    StartCoroutine(GetUpNextFrame());
}

private System.Collections.IEnumerator GetUpNextFrame() {
    yield return null;
    anim_contr.PlayAnimation(ActionForAnim.GetUp);
}

public void SetHandsCollidersAsCurrent()
{

```

```

        current_attack_colliders = new WeaponComponent[_hands_colliders.Length];
        for (int i = 0; i < current_attack_colliders.Length; i++)
        {
            current_attack_colliders[i] = _hands_colliders[i];
        }
        current_attack_data_getter = BaseAttackDataGetter;
        Array.ForEach(current_attack_colliders, e =>
e.SetOnTriggerEnterAction(AttackAction, current_attack_data_getter));
        SetAttackEvents(current_attack_colliders);

        (anim_contr as PlayerAnimController).SetAnimator(WeaponType.None);
    }

    private void SetFootCollider(){
        foot_collider.SetOnTriggerEnterAction(AttackAction, () => foot_kick);
    }

    public void SetWeaponCollidersHandler(WeaponComponent[] c)
    {
        current_attack_colliders = new WeaponComponent[c.Length];
        for (int i = 0; i < current_attack_colliders.Length; i++)
        {
            current_attack_colliders[i] = c[i];
        }

        current_attack_data_getter = BaseAttackDataGetter;
        Array.ForEach(c, e => e.SetOnTriggerEnterAction(AttackAction,
current_attack_data_getter));
        SetAttackEvents(c);
        Debug.Log("end of set weapon func");
    }

    private void Update()
    {
        ControlPlayer();
    }
    private void LateUpdate()
    {
        RotatePlayer();
        cursor_pointer.transform.position = _mouse_on_ground;
    }

    [SerializeField] private Rigidbody rig;
    public void ControlPlayer()
    {
        if(!_can_be_controlled) return;

        switch (State)
        {
            case State.Normal:
                MoveCharacter();
                ReadInteraction();
                ReadSpells();
                ReadAttack();
                break;
            case State.Spelling:
                MoveCharacter();
                Spelling();
                break;
            case State.DashBack:
                rig.velocity = dash_speed_mod * Run_Speed * dash_d;
                break;
        }
    }

```

```

        default:break;
    }
}

private void ReadInteraction() {
    if (!Input.GetKeyDown(KeyCode.F)) return;

    Collider[] cs = Physics.OverlapSphere(transform.position,
interact_overlap_distance, interactable_object_mask);
    if(cs.Length <= 0) return;

    float min_dist = float.MaxValue, tmp_dist;
    int min_index = 0;
    for(int i = 0; i < cs.Length; i++) {
        tmp_dist = Vector3.Distance(transform.position, cs[i].transform.position);
        if(tmp_dist < min_dist) {
            min_dist = tmp_dist;
            min_index = i;
        }
    }
    cs[min_index].GetComponentInParent<IInteractable>().Interact();
}

private void ReadSpells() {
    if(!_can_spell) return;

    for(int i = 0; i < spell_keys.Length; i++) {
        if(Input.GetKeyDown(spell_keys[i]) && _spell_slots[i].HaveSpell) {
            _chosen_spell_slot = _spell_slots[i];

            _chosen_spell_slot.GetSpellCosts(out float energy_cost, out float
health_cost);
            if(curr_energy < energy_cost || curr_hp <= health_cost)
            {
                // show player, that character does not have enough spell resources
                break;
            }

            spell_key_pressed = spell_keys[i];
            if(_chosen_spell_slot.UseSpell(SpellState.Preparing).IsMistake) {
                ChangeCanSpell(true);
            }
            break;
        }
    }
}

private void Spelling() {
    if (Input.GetKeyDown(spell_key_pressed)) { // cancel spelling

        _chosen_spell_slot.UseSpell(SpellState.Cancel);
        _can_attack = _can_dash = true;
        ChangeCanSpell(true);
        ChangeStateTo(State.Normal);
        return;
    }
}

```

```

        if (Input.GetMouseButtonDown(0)) {
            Spell.SpellUseResult res = _chosen_spell_slot.UseSpell(SpellState.InProcess);
            if(res.IsDone) { // cast, spelling over, back to normal state

                //anim_contr.PlayAnimation(ActionForAnim.CastAndOver);
                if(!res.Delayed) {
                    _can_attack = _can_dash = true;
                    ChangeCanSpell(true);
                    _chosen_spell_slot = null;
                    ChangeStateTo(State.Normal);
                }
            }
        }
    }

    public void SetIfCanMove(bool can)
    {
        _can_move = can;
        if(!can) {
            anim_contr.SetWalking(false);
        }
    }

    private void MoveCharacter()
    {
        if (!_can_move) {
            PlayerAnimController local_anim_contr = this.anim_contr as
            PlayerAnimController;
            IsPlayerMoving = false;
            local_anim_contr.SetWalking(false);
            return;
        }
        float horizontal = Input.GetAxisRaw("Horizontal");
        float vertical = Input.GetAxisRaw("Vertical");
        Vector3 direction = new Vector3(horizontal, 0f, vertical).normalized;

        PlayerAnimController anim_contr = this.anim_contr as PlayerAnimController;
        if (direction.magnitude >= 0.1f)
        {
            IsPlayerMoving = true;
            anim_contr.SetWalking(true, Run_Speed);
            anim_contr.CalcWalkAnim(direction);

            if (Input.GetKeyDown(KeyCode.LeftShift) && _can_dash)
            {
                ExitAttackState();
                SetRotatePlayer(false);
                ChangeStateTo(State.DashBack);
                _can_dash = false;
                dash_d = direction;
                dash_trail.enabled = true;

                (anim_contr as PlayerAnimController).SetSlowWalk(false,
                cur_run_anim_speed, true);
                change_state_cor = StartCoroutine(Coroutines.DelayCoroutine(dash_duration,
                () => {
                    SetRotatePlayer(true);
                    ChangeStateTo(State.Normal);
                    dash_trail.enabled = false;
                    (anim_contr as PlayerAnimController).SetSlowWalk(false,
                    cur_run_anim_speed);

```

```

        change_state_cor = StartCoroutine(Coroutines.DelayCoroutine(dash_cd,
() => _can_dash = true)); // dash cd
    }));
    }

    rig.velocity = Run_Speed * direction;
}
else
{
    rig.velocity = Vector3.zero;
    IsPlayerMoving = false;
    anim_contr.SetWalking(false);
    anim_contr.CalcWalkAnim(Vector3.zero);

    if (Input.GetKeyDown(KeyCode.LeftShift) && _can_dodge)
    {
        _can_dodge = false;
        _can_parry = false;
        ChangeStateTo(State.Dodge);
        ChangeBodyStateTo(BodyState.Dodge);
        anim_contr.PlayAnimation(ActionForAnim.Dodge);

        anim_events.SetAct(() => {
            ChangeBodyStateTo(BodyState.Normal);
        });

        anim_events.SetOnAnimationEndAct(() => {
            ChangeStateTo(State.Normal);
            _ = StartCoroutine(Coroutines.RealTimeCooldownCoroutine(.2f, () =>
            {
                _can_dodge = true;
                _can_parry = true;
            })); // cd
        });
        return;
    }
}
}
}

```

```

private void RotatePlayer()
{
    Ray ray = camera.ScreenPointToRay(Input.mousePosition);

    if (Physics.Raycast(ray, out RaycastHit raycastHit, 100f, ground_layer))
    {
        _mouse_on_ground = raycastHit.point;
        _mouse_on_ground.y = transform.position.y;
        if (!_can_be_rotated || !_can_be_controlled) return;
        Quaternion target = Quaternion.LookRotation((_mouse_on_ground -
transform.position).normalized);
        transform.rotation = Quaternion.Slerp(transform.rotation, target,
rotation_speed);
    }
}

```

```

public void ReadAttack()
{

```

```

// when cursor is on inventory UI, player could not be able to attack
if (!_can_attack || EventSystem.current.IsPointerOverGameObject()) {
    return;
}

    if(Input.GetKeyDown(KeyCode.Space)) {
        SetAttackEvents(foot_collider);

        ChangeStateTo(State.Waiting);
        rig.isKinematic = true;

        anim_contr.PlayAnimation(ActionForAnim.FootKick);
        AfterAttackEvent += AfterKick;
        anim_events.SetOnAnimationEndAct(() => {
            ChangeStateTo(State.Normal);
            rig.isKinematic = false;
        });
        return;
    }

    if(Input.GetKeyDown(KeyCode.LeftControl) && _weapon != null && _can_parry)
    {
        SetCans(false);
        ChangeBodyStateTo(BodyState.Parry);
        anim_contr.PlayAnimation(ActionForAnim.TryParry);

        anim_events.SetAct(() => {
            ChangeBodyStateTo(BodyState.Normal);
        });

        anim_events.SetOnAnimationEndAct(() => {
            _ = StartCoroutine(Coroutines.RealTimeCooldownCoroutine(.2f, () =>
            {
                SetCans(true);
            })); // cd
        });
        return;
    }

    if(Input.GetMouseButtonDown(0)) {
        is_alt_attack = false;
        _=Attack(false);
    }else if (Input.GetMouseButtonDown(1)) {
        is_alt_attack = true;
        _=Attack(true);
    }
}

private bool IsInTransition() => anim_contr.IsInTransition(1);
private bool is_alt_attack=false;
private bool Attack(bool is_alt_attack)
{
    if (is_alt_attack) {
        anim_contr.PlayAnimation(ActionForAnim.Alt_Attack);
    } else {
        anim_contr.PlayAnimation(ActionForAnim.Attack);
    }
}

_can_attack = false;
attack_result = AttackResult.NoHit;

//StartAttack(attack_colliders);

```

```

        return true;
    }

    public void SetWeapon(Weapon w)
    {
        _weapon = w;
        if(w == null)
        {
            main_attack = hands_main;
            alt_attack = hands_alt;
            SetHandsCollidersAsCurrent();
            return;
        }
        main_attack = w.MainAttackInfo;
        alt_attack = w.AltAttackInfo;
        (anim_contr as PlayerAnimController).SetAnimator(w.WeaponType);
        (anim_contr as PlayerAnimController).SetAttackSpeedMod(w.AttackSpeedMod);
    }

    protected override void PrepareToSpell()
    {
        ChangeStateTo(State.Spelling);
    }

    public bool CanChangeEquipment(){
        if(is_in_attack_state) return false;

        return true;
    }

    public void SetControl(bool can_control) {
        _can_be_controlled = can_control;
    }

```

Важливі функції класу SaveLoadManager:

```

    public static void SaveData<T>(T data, string path)
    {
        if (!Directory.Exists(Application.persistentDataPath + directory)) {
            _ = Directory.CreateDirectory(Application.persistentDataPath + directory);
        }

        BinaryFormatter formatter = new BinaryFormatter();
        path = Application.persistentDataPath + path;
        using (FileStream stream = new FileStream(path, FileMode.Create)){
            formatter.Serialize(stream, data);
            Debug.LogWarning(path);
        }
    }

    public static T LoadData<T>(string path)
    {
        path = Application.persistentDataPath + path;
        Debug.Log(path);
        if (File.Exists(path)){
            T res;
            BinaryFormatter formatter = new BinaryFormatter();
            using(FileStream stream = new FileStream(path, FileMode.Open))
            {

```

```

        res = (T)formatter.Deserialize(stream);
    }
    return res;
}else{
    Debug.LogError("Save file not found in" + path);
    return default(T);
}
}

public static void SaveGame(int scene_id, int arena_level = 0)
{
    Debug.Log("Saving game");

    SaveData data = new SaveData();
    Player pl = GameManager.GetPlayer();
    data.CurrentHP = pl.CurHP;
    data.CurrentEN = pl.CurEN;
    data.Upgrades = pl.GetUpgrades();
    data.SceneId = scene_id;
    data.ArenaLevel = arena_level;

    Item[] inventory = Inventory.GetInstance().Items.ToArray();
    data.Inventory = new string[inventory.Length];

    for(int i =0;i< inventory.Length; i++){
        Debug.Log(i);
        Debug.Log(inventory[i]);
        Debug.Log(inventory[i].Name);
        data.Inventory[i] = inventory[i].Name;
    }

    Item[] equipped = Equipments.GetInstance().Equiped_items;
    data.EquipedEquipments = new string[equiped.Length];
    for(int i =0;i< equipped.Length; i++){
        if (equiped[i]){
            data.EquipedEquipments[i] = equipped[i].Name;
        }else{
            data.EquipedEquipments[i] = string.Empty;
        }
    }

    SpellSlot[] _spell_slots = ameManager.GetPlayerController().GetSpellSlots;
    data.EquipedSpell = new string[_spell_slots.Length];
    for(int i =0;i< _spell_slots.Length; i++)
    {
        if (_spell_slots[i].GetItem){
            data.EquipedSpell[i] = _spell_slots[i].GetItem.Name;
        }else{
            data.EquipedSpell[i] = string.Empty;
        }
    }
    string save = "/Saves/" + LoadInfo.save_file;
    SaveData<SaveData>(data, save);
    LoadInfo.is_new = false; // if player die, it loads
}

public static void LoadGame(string file_name="/save.fab") {
    Debug.Log("loading game");

    string save_path = "/Saves/" + LoadInfo.save_file;
    SaveData data = LoadData<SaveData>(save_path);

```

```

        if(data == null) return;

        string[] items_to_delete_from_scene = new string[data.EquippedEquipments.Length +
data.Inventory.Length];
        Array.Copy(data.EquippedEquipments, items_to_delete_from_scene,
data.EquippedEquipments.Length);
        Array.Copy(data.Inventory, 0, items_to_delete_from_scene,
data.EquippedEquipments.Length, data.Inventory.Length);
        if(SceneManager.GetActiveScene().buildIndex == 4) { // arena scene build index
            // set arena level
            ArenaSpawner arsp = ArenaSpawner.GetInstance();
            arsp.SetIteration(data.ArenaLevel);
            arsp.RemoveItemsThatWasGiven(items_to_delete_from_scene);
        }else {
            GameManager.DestroyPickedUpItemsFromScene(items_to_delete_from_scene);
        }

        Item[] equipped equipments = new Item[data.EquippedEquipments.Length];
        for(int i =0;i< data.EquippedEquipments.Length; i++)
        {
            if (data.EquippedEquipments[i] == string.Empty) continue;
            equipped equipments[i] =
Resources.Load<Item>($"Items/{data.EquippedEquipments[i]}");
        }

        Item[] equipped_spells= new Item[data.EquippedSpell.Length];
        for(int i =0;i< data.EquippedSpell.Length; i++)
        {
            if (data.EquippedSpell[i] == string.Empty) continue;
            equipped_spells[i] = Resources.Load<Item>($"Items/{data.EquippedSpell[i]}");
        }
        Item[] inventory = new Item[data.Inventory.Length];
        for(int i =0;i< data.Inventory.Length; i++){
            inventory[i] = Resources.Load<Item>($"Items/{data.Inventory[i]}");
        }

        GameManager.GetNewItemPopUpsController().UpdateAddPopUpUsedItems(equipped_equipments);
        GameManager.GetNewItemPopUpsController().UpdateAddPopUpUsedItems(equipped_spells);
        GameManager.GetNewItemPopUpsController().UpdateAddPopUpUsedItems(inventory);
        for (int i = 0; i < data.EquippedEquipments.Length; i++){
            if (equipped equipments[i] == null) continue;
            Equipments.GetInstance().Equip(equipped equipments[i], false);
        }
        SpellSlot[] _spell_slots = GameManager.GetPlayerController().GetSpellSlots;
        for (int i = 0; i < data.EquippedSpell.Length; i++){
            if (equipped_spells[i] == null) continue;
            _spell_slots[i].SetItem(equipped_spells[i], false);
        }
        for (int i = 0; i < data.Inventory.Length; i++){
            _=Inventory.GetInstance().AddToInventory(inventory[i], false);
        }
        Player pl = GameManager.GetPlayer();
        pl.SetPlayerCurHP(data.CurrentHP);
        pl.SetPlayerCurEN(data.CurrentEN);
        pl.SetUpgrades(data.Upgrades);
    }

    public static bool CheckIfSaveExists(){
        return File.Exists(Application.persistentDataPath + save_path);
    }
    public static void DeleteSave(string file_name) {

```

```

        File.Delete(Application.persistentDataPath + "/Saves/"+file_name);
    }

```

Важливі методи класу DamageableCharacter:

```

protected override void Init()
{
    base.Init();
    HPChange += UpdateHPBar;
    ENChange += UpdateENBar;

    // these stats are going to be updated in the end of the Init function, but
    without setting them here everything crushes
    curr_hp = max_hp = START_MAX_HP;
    curr_energy = max_energy = START_MAX_ENERGY;
    curr_run_speed = base_run_speed = START_RUN_SPEED;
    cur_run_anim_speed = base_run_anim_speed = START_RUN_ANIMATION_SPEED;
    hp_regen = en_regen = 0f;

    DamageableCharacterStats stats = this.stats as DamageableCharacterStats;

    int damage_type_count = System.Enum.GetNames(typeof(DamageType)).Length;
    int attack_type_count = System.Enum.GetNames(typeof(AttackType)).Length;

    damage_resistances = new float[damage_type_count];
    attack_resistances = new float[attack_type_count];
    attributes = new CharacterAttributes(stats.CharacterAttributes);

    current_status_effects = new
System.Collections.Generic.List<CurrentStatusEffect>();
    current_dots = new System.Collections.Generic.List<CurrentDot>[damage_type_count];
    on_dot_tick_events = new System.Action<DamageableCharacter,
DotData>[damage_type_count];

    for (int i = 0; i < damage_resistances.Length; i++) {
        damage_resistances[i] = stats.DamageResistance((DamageType)i);
        current_dots[i] = new System.Collections.Generic.List<CurrentDot>();
    }

    for(int i = 0; i < attack_resistances.Length; i++) {
        attack_resistances[i] = stats.AttackResistance((AttackType)i);
    }
    ShowBars(true);
    SetEventOnAttributeChange(UpdateAtributeStats);
    InitAttributes();
    ShowBars(false);
    HPChange += ShowBarsAfterFirstAttack;
    OnUseEnergy += ShowBarsAfterFirstEnergyUse;
    regen_coroutine = StartCoroutine(Regeneration());
}

protected void ChangeCanTakeHit(bool can) {
    _can_take_hit = can;
}

protected void ShowBarsAfterFirstAttack() {
    ShowBars(true);
    HPChange -= ShowBarsAfterFirstAttack;
    OnUseEnergy -= ShowBarsAfterFirstEnergyUse;
}
private void ShowBarsAfterFirstEnergyUse(DamageableCharacter target, float
current_energy, float energy_value) {

```

```

        ShowBars(true);
        HPChange -= ShowBarsAfterFirstAttack;
        OnUseEnergy -= ShowBarsAfterFirstEnergyUse;
    }

    private System.Collections.IEnumerator Regeneration(){

        while(true) {

            if(hp_regen != 0){
                if(curr_hp + hp_regen >= max_hp) {
                    curr_hp = max_hp;
                }else {
                    curr_hp += hp_regen;
                }
                InvokeHPChangeEvent();
                if (curr_hp <= 0) {
                    Die(new DamageData("Regen", hp_regen, 0, DamageType.Physical), null);
                }
            }
            if(en_regen != 0) {
                if(curr_energy + en_regen >= max_energy) {
                    curr_energy = max_energy;
                }else {
                    curr_energy += en_regen;
                }
                InvokeENChangeEvent();
            }

            yield return new WaitForSeconds(1f);
        }
    }

    protected void InvokeHPChangeEvent()
    {
        HPChange?.Invoke();
    }

    protected void InvokeENChangeEvent()
    {
        ENChange?.Invoke();
    }

    public virtual void TakePush(FightingCharacter sender, Vector3 direction, float force,
float stunn_dur = 0f) {
        Rigidbody rig = GetComponent<Rigidbody>();
        if(rig == null) {
            Debug.Log("AddForce. Rigidbody is null");
            return;
        }
        if(!_can_be_pushed) return;
        _can_be_pushed = false;

        OnPushEvent?.Invoke(rig);
        // play pushed animation
        ChangeStateTo(State.Pushed);

        // start moving character
        push_coroutine = StartCoroutine(Pushing(rig, direction, force));

        anim_contr.PlayAnimation(ActionForAnim.Pushed);
        if(stunn_dur > 0f) {
            anim_events.SetOnAnimationEndAct(() => {

```

```

        _can_be_pushed = true;
        if(push_coroutine!=null) StopCoroutine(push_coroutine);
        TakeStunn(stunn_dur, rig);
    });
} else {
    anim_events.SetOnAnimationEndAct(() => {
        _can_be_pushed = true;
        if(push_coroutine!=null) StopCoroutine(push_coroutine);
        AfterPushEvent?.Invoke(rig);
    });
}
}
}

```

```
protected Coroutine push_coroutine = null, from_stunn_coroutine = null;
```

```
private System.Collections.IEnumerator Pushing(Rigidbody rig, Vector3 direction, float force){
```

```

    bool was_kinematic = false;
    if(rig.isKinematic) {
        rig.isKinematic = false;
        was_kinematic = true;
    }
    force /= rig.mass;
    Debug.Log(LayerMask.LayerToName(gameObject.layer));
    while(force > 0.1){
        rig.velocity = force * direction;
        force *= 0.9f;
        yield return new WaitForFixedUpdate();
    }
    rig.velocity = Vector3.zero;
    if(was_kinematic) {
        rig.isKinematic = true;
    }
}

```

```
public virtual void TakeStunn(float duration, Rigidbody rig=null) {
    if(duration <= 0 || !_can_be_stunned) return;
```

```

    if(rig == null) {
        rig = GetComponent<Rigidbody>();
        if(rig == null) {
            Debug.Log("TakeStunn. Rigidbody is null");
            return;
        }
    }
}

```

```
OnStunnEvent?.Invoke(rig);
```

```

    ChangeStateTo(State.Stunned);
    // play stunned animation
    anim_contr.PlayAnimation(ActionForAnim.Stunned);
    if(from_stunn_coroutine != null) {
        StopCoroutine(from_stunn_coroutine);
    }
    from_stunn_coroutine = StartCoroutine(Coroutines.DelayCoroutine(duration, ()=> {
        anim_contr.PlayAnimation(ActionForAnim.LeaveStunn);
        AfterStunnEvent?.Invoke(rig);
    }));
}

```

```
public AttackResult TakeHit(AttackData data, FightingCharacter attacker, bool show=true) {
```

```

        if (!_can_take_hit) return AttackResult.CannotBeAttacked;
        if (data.invulnerability_time_after > 0) {
            ChangeCanTakeHit(false);
            invulnerability_time_after_coroutine =
StartCoroutine(Coroutines.DelayCoroutine(data.invulnerability_time_after, () => {
ChangeCanTakeHit(true); }));
        }

        if (_body_state== BodyState.Parry && data.can_be_parried) {
            OnParryEvent(data, attacker, this);
            return AttackResult.Parried;
        }

        if (_body_state == BodyState.Dodge && data.can_be_dodged) {
            OnDodgeEvent?.Invoke(data, attacker, this);
            return AttackResult.Dodged;
        }

        AttackResult res = AttackResult.Hit;
        TakeAttack(data, attacker, show);
        OnTakeHit?.Invoke(data, attacker, this, out res);

        return res;
    }

    private static float CalculateArmorDifference(float[] attack_resistances, AttackData
ad) {
        return attack_resistances[(int)ad.attack_type] - ad.armor_pp;
    }

    static protected float CalculateDamage(AttackData data, float[] attack_resistances,
float[] damage_resistances)
    {
        float armor_diff = CalculateArmorDifference(attack_resistances, data);

        if (armor_diff >= MAX_ARMOR_VALUE) return 0;
        float damage = data.damage_data.damage;
        damage *= (1 - armor_diff / MAX_ARMOR_VALUE);
        damage *= (1 - damage_resistances[(int)data.damage_data.damage_type] / 100);
        return damage;
    }

    static protected float CalculateEnergyLoss(AttackData data, float[]
attack_resistances, float[] damage_resistances) {
        if(data.damage_data.en_damage <= 0) return 0;

        float armor_diff = data.attack_type == AttackType.Magic ?
attack_resistances[(int)data.attack_type] - data.armor_pp : 0;

        if (armor_diff >= MAX_ARMOR_VALUE) return 0;
        float en_damage = data.damage_data.en_damage;
        en_damage *= (1 - armor_diff / MAX_ARMOR_VALUE);
        en_damage *= (1 - damage_resistances[(int)DamageType.Energy] / 100);
        return en_damage;
    }

    public virtual void TakeAttack(AttackData attack, FightingCharacter attacker, bool
show_damage = true){

        OnTakeAttackEvent?.Invoke(attack, attacker, this); // event
        if (attack.attack_type != AttackType.Pure) {
            attack.damage_data.damage = CalculateDamage(attack, attack_resistances,
damage_resistances);

```

```

        attack.damage_data.en_damage = CalculateEnergyLoss(attack, attack_resistances,
damage_resistances);
    }

    if(attack.attack_type != AttackType.Magic) {
        TakeDamageEffect(attacker, attack);
    }

    TakeEnergyLoss(attack, attacker, false, show_damage);
    TakeDamage(attack.damage_data, attacker, false, show_damage);

    HandleDots(attack.dots);
}

public virtual void TakeDamage(DamageData data, FightingCharacter attacker, bool
is_dot = false, bool show=true)
{
    float the_damage = data.damage;
    if (the_damage > 0) {

        if(!is_dot) {
            AudioManager.PlayDamageSound(data, this);
        }

        curr_hp -= the_damage;
        curr_hp = (float)System.Math.Round((double)curr_hp, 2);
        float percent_from_max = the_damage / max_hp;

        HPChange?.Invoke();

        switch(data.damage_type) {
            case DamageType.Bleed:
                GameObject bleed =
ObjectPool.Instance.GetPooledObject(PoolObject.Bleed);
                bleed.transform.position = transform.position;
                bleed.SetActive(true);
                break;
            case DamageType.Ice:
                int agility_minus = (int) (percent_from_max * 50);
                Debug.Log(agility_minus);
                ChangeAttribute(Attribute.Agility, -agility_minus, agility_minus / 2);
                break;
        }

        if (show) {
            GameManager.ShowDamage(the_damage, percent_from_max, data.damage_type,
transform);
        }

        if (curr_hp <= 0) {
            Die(data, attacker);
        } else {
            AfterTakeDamageEvent?.Invoke(this, attacker, the_damage); // only if it
survives tho
        }
    }
    else if (the_damage < 0) {
        TakeHeal(-the_damage, show);
        HPChange?.Invoke();
    }
}

```

```

public void TakeHeal(float heal, bool show=true)
{
    if (curr_hp + heal > max_hp){
        if(show) {
            float h = max_hp - curr_hp;
            GameManager.ShowHeal(h, h / max_hp, transform );
        }
        curr_hp = max_hp;
    }else{
        if(show) {
            GameManager.ShowHeal(heal, heal / max_hp, transform);
        }
        curr_hp += heal;
    }
}

private void HandleDots(DotData[] dots)
{
    if(dots == null) return;
    foreach (DotData dot in dots) {
        int dot_type_index = (int)dot.damage_data.damage_type;

        int dot_index = current_dots[dot_type_index].FindIndex(d => d.Data.dot_name ==
dot.dot_name); // seek by name
        if (dot_index != -1){
            switch (dot.rebaff_type){
                case RebaffType.RestoreAmount:
                    StopCoroutine(current_dots[dot_type_index][dot_index].Coroutine);
                    current_dots[dot_type_index][dot_index].Coroutine =
StartCoroutine(
                        TakeDotDamageCoroutine(current_dots[dot_type_index][dot_in
dex], dot_type_index, false)
                    );
                    break;
                case RebaffType.IncreaseDamage:
                    current_dots[dot_type_index][dot_index].Data.damage_data.damage *=
1.5f;// += dot.damage_data.damage;
                    Debug.Log($"increase damage:
{current_dots[dot_type_index][dot_index].Data.damage_data.damage} in
{current_dots[dot_type_index][dot_index].Data.dot_name}");
                    break;
                case RebaffType.Nothing:
                    break;
            }
            continue;
        }else {
            if(dot.damage_data.damage_type == DamageType.Fire) {
                fire_particle = ObjectPool.Instance.GetPooledObject(PoolObject.Fire);
                fire_particle.transform.position = transform.position;
                fire_particle.transform.parent = transform;
                fire_particle.SetActive(true);
            }

            CurrentDot tmp = new CurrentDot(dot, null);
            tmp.Coroutine = StartCoroutine(TakeDotDamageCoroutine(tmp,
dot_type_index));
            current_dots[dot_type_index].Add(tmp);
            Debug.Log($"new dot: {tmp.Data.dot_name}");
        }
    }
}

```

```

    }
}
private System.Collections.IEnumerator TakeDotDamageCoroutine(CurrentDot current_dot,
int dot_type_index, bool wait_delay = true) {
    yield return new WaitForEndOfFrame(); // needed to get a link to a dot
    string dot_name = current_dot.Data.dot_name;
    if (wait_delay) {
        yield return new WaitForSeconds(current_dot.Data.delay);
    }
    DamageData d = current_dot.Data.damage_data;

    for (int i = 0; i < current_dot.Data.amount-1; i++)
    {
        // calculate damage with resistance
        d = current_dot.Data.damage_data;
        d.damage = current_dot.Data.damage_data.damage * (1 -
damage_resistances[dot_type_index] / 100);
        switch (current_dot.Data.dot_damage_type)
        {
            case DotDamageType.Default: // (1)
                break;
            case DotDamageType.PercentFromCurrent:
                d.damage = curr_hp / 100 * d.damage;
                break;
            case DotDamageType.PercentFromMax:
                d.damage = max_hp / 100 * d.damage;
                break;
        }
        TakeDamage(d, null, true, true); // (1)
        on_dot_tick_events[dot_type_index]?.Invoke(this, current_dot.Data);
{current_dot.Data.freq_incr}");
        current_dot.Data.frequency *= current_dot.Data.freq_incr;
{current_dot.Data.damage_incr}");
        current_dot.Data.damage_data.damage *= current_dot.Data.damage_incr;

        yield return new WaitForSeconds(current_dot.Data.frequency);

    }

    // calculate damage with resistance
    d.damage = current_dot.Data.damage_data.damage * (1 -
damage_resistances[dot_type_index] / 100);
    switch (current_dot.Data.dot_damage_type)
    {
        case DotDamageType.Default:
            break;
        case DotDamageType.PercentFromCurrent:
            d.damage = curr_hp / 100 * d.damage;
            break;
        case DotDamageType.PercentFromMax:
            d.damage = max_hp / 100 * d.damage;
            break;
    }
    TakeDamage(d, null, true, true);

    Debug.Log("removing" + dot_name);
    if(d.damage_type == DamageType.Fire && current_dots[dot_type_index].Count == 1) {
        fire_particle.SetActive(false);
        fire_particle.transform.parent = null;
        fire_particle = null;
    }
    Debug.Log(current_dots[dot_type_index].Remove(current_dot));
}

```

```

}
GameObject fire_particle = null;

private void UpdateHPBar()
{
    hp_bar.ChangeValue(curr_hp);
}
private void UpdateENBar() {
    en_bar.ChangeValue(curr_energy);
    if(curr_energy <= 0) {
        curr_energy = 0;
        OnEnergyEmpty?.Invoke(this, 0, 0);
    }
}

protected void InitAttributes(){
    for (int i = 0; i < attributes.Length; i++){
        UpdateAttributeStats((Attribute)i, 0, attributes[i]);
    }
}

protected virtual void UpdateAttributeStats(Attribute atr, int old_value, int
new_value) // it resets and then adds stats depends on atr
{
    float curr_percent_main;//, incr = 1.1f;
    switch (atr){
        case Attribute.Strength:
            BeforeStrengthCalculationEvent?.Invoke();
            curr_percent_main = curr_hp / max_hp;

            max_hp = START_MAX_HP + new_value * STRENGTH_ATTRIBUTE_MODIFIER;

            int old_resistances_strength_mod = old_value *
(int)STRENGTH_ATTRIBUTE_MODIFIER;
            int new_resistances_strength_mod = new_value *
(int)STRENGTH_ATTRIBUTE_MODIFIER;
            damage_resistances[(int)DamageType.Physical] -=
(old_resistances_strength_mod - new_resistances_strength_mod);
            damage_resistances[(int)DamageType.Ice] -= (old_resistances_strength_mod -
new_resistances_strength_mod);
            damage_resistances[(int)DamageType.Venom] -=
(old_resistances_strength_mod - new_resistances_strength_mod);
            damage_resistances[(int)DamageType.Bleed] -= (old_resistances_strength_mod
- new_resistances_strength_mod);

            hp_bar.SetMaxValue(max_hp);
            curr_hp = max_hp * curr_percent_main;
            curr_energy = max_energy * curr_percent_alt;
            InvokeHPChangeEvent();
            InvokeENChangeEvent();
            break;

        case Attribute.Agility:
            BeforeAgilityCalculationEvent?.Invoke();
            curr_percent_main = curr_run_speed / base_run_speed;

            base_run_speed = START_RUN_SPEED + new_value * AGILITY_ATTRIBUTE_MODIFIER;
            float speed_increased = base_run_speed / START_RUN_SPEED;
            base_run_anim_speed = START_RUN_ANIMATION_SPEED * speed_increased;

            int old_resistances_agility_mod = old_value *
(int)AGILITY_ATTRIBUTE_MODIFIER;

```

```

        int new_resistances_agility_mod = new_value *
(int)AGILITY_ATTRIBUTE_MODIFIER;
        Debug.Log($"old: {old_value}, new: {new_value}");
        Debug.Log($"old mod: {old_resistances_agility_mod}, new mod:
{new_resistances_agility_mod}");
        damage_resistances[(int)DamageType.Fire] -= (old_resistances_agility_mod -
new_resistances_agility_mod);

        curr_run_speed = base_run_speed * curr_percent_main;
        cur_run_anim_speed = base_run_anim_speed * curr_percent_main;

        if (anim_contr != null) {
            anim_contr.SetRunAnimationSpeed(cur_run_anim_speed);
        }
        break;

        case Attribute.Capabilities:
            BeforeCapabilitiesCalculationEvent?.Invoke();
            curr_percent_main = curr_energy / max_energy;

            max_energy = START_MAX_ENERGY + new_value *
CAPABILITIES_ATTRIBUTE_MODIFIER;

            int old_resistances_cap_mod = old_value *
(int)CAPABILITIES_ATTRIBUTE_MODIFIER;
            int new_resistances_cap_mod = new_value *
(int)CAPABILITIES_ATTRIBUTE_MODIFIER;

            damage_resistances[(int)DamageType.Rot] -= (old_resistances_cap_mod -
new_resistances_cap_mod);
            damage_resistances[(int)DamageType.Energy] -= (old_resistances_cap_mod -
new_resistances_cap_mod);

            en_bar.SetMaxValue(max_energy);
            curr_energy = max_energy * curr_percent_main;
            curr_hp = max_hp * curr_percent_alt;
            InvokeENChangeEvent();
            break;
        }
    }

protected void ShowBars(bool show){
    hp_bar.ShowBar(show);
    en_bar.ShowBar(show);
}

protected virtual void Die(DamageData data, FightingCharacter killer)
{
    OnDie?.Invoke();
    this.enabled = false;
    ChangeCanTakeHit(false);
    ShowBars(false);

    ChangeStateTo(State.Dead);

    StopAllCoroutines();
    StartCoroutine(DestroyCoroutine(2.5f));
}

private System.Collections.IEnumerator DestroyCoroutine(float delay) {
    yield return new WaitForSeconds(delay);
    RemoveParticles();
}

```

```

}

protected void RemoveParticles() {
    ParticleSystem[] particles = GetComponentsInChildren<ParticleSystem>();
    for(int i = 0; i < particles.Length; i++) {
        particles[i].transform.parent = null;
        particles[i].gameObject.SetActive(false);
    }
}

protected virtual void TakeDamageEffect(FightingCharacter attacker, AttackData
attack){
    GameObject go = ObjectPool.Instance.GetPooledObject(PoolObject.Blood);
    go.transform.position = transform.position;
    go.transform.rotation = attacker.transform.rotation;
    ParticleSystem ps = go.GetComponent<ParticleSystem>();
    ParticleSystem.EmissionModule e = ps.emission;
    short particles_to_burst = (short)(attack.damage_data.damage / max_hp *
max_blood_particle_count);
    e.SetBurst(0, new ParticleSystem.Burst(0f, particles_to_burst));
    go.SetActive(true);
    Debug.Log(particles_to_burst);
}

public void AddEnergy(float en) {
    if(en > 0) {
        if(curr_energy + en >= max_energy) {
            curr_energy = max_energy;
        }else {
            curr_energy += en;
        }
        InvokeENChangeEvent();
    }else if(en < 0){
        if(curr_energy + en <= 0) {
            curr_energy = 0;
        }else {
            curr_energy += en;
        }
        InvokeENChangeEvent();
    }
}

public void ChangeAttribute(Attribute atr, int value, float duration=0) {
    if(attributes[atr] + value <= 0) return;

    attributes[atr] += value;
    if(duration > 0) {
        StartCoroutine(Coroutines.DelayCoroutine(duration, ()=> {
            attributes[atr] -= value;
        }));
    }
}

```