

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 шутеру від першої особи»

Виконав студент групи ЗКН-19-1
ПИЛИПЧУК Сергій Олександрович

Керівник роботи:
ЛЕВЧЕНКО Антон Юрійович

Консультант:
КУЗЬМЕНКО Олександр Вікторович

Рецензент:
ГРАФ Марина Сергіївна

Житомир – 2023

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

ЗАТВЕРДЖУЮ

Завідувач кафедри комп'ютерних наук

Марина ГРАФ

«05» січня 2023 р.

ЗАВДАННЯ

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

Здобувач вищої освіти: **ПИЛИПЧУК Сергій Олександрович**

Керівник роботи: **ЛЕВЧЕНКО Антон Юрійович**

Тема роботи: **«Розробка шутеру від першої особи»,**

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

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

Вихідні дані роботи: об'єктом дослідження є розробка ігор з тривимірною графікою на основі існуючого ігрового рушія. Предметом дослідження є створення шутеру від першої особи на основі ігрового рушія Unity.

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

Розділ	Консультант	Дата	
		Завдання видав	Завдання прийняв
1	Кузьменко О.В.	02.02.2023р.	02.02.2023р.
2	Кузьменко О.В.	05.03.2023р.	05.03.2023р.
3	Кузьменко О.В.	10.05.2023р.	10.05.2023р.

Календарний план

№ з/п	Назва етапів випускної кваліфікаційної роботи	Термін виконання етапів роботи	Примітка
1	Постановка задачі. Пошук, огляд та аналіз аналогічних розробок. Формулювання технічного завдання. Опрацювання літературних джерел.	2 лютого 2023 – 8 березня 2023	Виконано
2	Проектування структури системи	9 березня 2023 – 15 Квітня 2023	Виконано
3	Написання програмного коду	16 Квітня 2023 – 9 травня 2023	Виконано
4	Тестування системи	10 травня 2023 – 16 травня 2023	Виконано
5	Оформлення пояснювальної записки	17 травня 2023 – 1 червня 2023	Виконано
6	Попередній захист	1 червня 2023	Виконано

Здобувач вищої освіти **Сергій ПИЛИПЧУК**
Керівник **Антон ЛЕВЧЕНКО**

РЕФЕРАТ

Кваліфікаційна робота бакалавра складається з програмного забезпечення та пояснювальної записки. Пояснювальна записка до кваліфікаційної роботи містить 66 сторінок, 20 ілюстрацій та 2 таблиці.

Метою роботи є дослідження особливостей проектування архітектури, розробки алгоритмів, реалізації та тестування гри в жанрі шутер від першої особи з використанням ігрового рушія Unity.

В роботі визначено основні завдання на розробку системи, проаналізовано аналоги розробленої системи. Наведено сценарії роботи програмного забезпечення, загальна структура програмного забезпечення, опис алгоритмів взаємодії окремих модулів системи, опис об'єктної структури системи (діаграма класів). Також наводиться результати тестування системи.

КЛЮЧОВІ СЛОВА: ШУТЕР ВІД ПЕРШОЇ ОСОБИ, ВОГНЕПАЛЬНА ЗБРОЯ, ІГРОВА МЕХАНІКА, ШТУЧНИЙ ІНТЕЛЕКТ, ТРИВИМІРНА ГРАФІКА.

					КН.КР.Б – 122 – 23 – ПЗ		
Змн.	Арк.	№ докум.	Підпис	Дата			
Розроб.	С.О. ПИЛИПЧУК				Розробка шутера від першої особи Пояснювальна записка	Літ.	Арк.
Керівник	А.Ю. ЛЕВЧЕНКО						3
Рецензент	М.С. ГРАФ						66
Н. контр.						Житомирська політехніка, група ЗКН-19-1	
Зав. каф.	М.С. ГРАФ						

ABSTRACT

Bachelor's qualification work consists of a software package and documentation. The documentation to the bachelor's graduate work contains 66 pages, 20 illustrations and 2 tables.

The goal of the graduate work is to study the features of architecture design, algorithm development, implementation and testing of the game in the genre of first-person shooter, using the game engine Unity.

The main tasks for system development are defined in the work, analogues of the developed system are analyzed. The use cases of the software, the general structure of the software, the description of the algorithms of interaction of individual modules of the system, the description of the object structure of the system (class diagram) are given. The results of system testing are also given.

KEYWORDS: FIRST-PERSON SHOOTER, FIREARMS, GAME MECHANICS, ARTIFICIAL INTELLIGENCE, 3D GRAPHICS.

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

ЗМІСТ

РЕФЕРАТ.....	3
ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ.....	6
ВСТУП.....	7
РОЗДІЛ 1. АНАЛІЗ НАПРЯМКІВ ВИКОРИСТАННЯ КОМП'ЮТЕРНИХ ІГОР В ЖАНРІ ШУТЕР ВІД ПЕРШОЇ ОСОБИ	9
1.1 Постановка задачі	9
1.2 Огляд існуючих рішень створення шутеру	14
1.3 Опис інструментів розробки.....	19
Висновки до розділу 1	26
РОЗДІЛ 2. ПРОЕКТУВАННЯ ГРИ В ЖАНРІ ШУТЕР ВІД ПЕРШОЇ ОСОБИ	27
2.1 Визначення варіантів використання системи.....	27
2.2 Проектування структури системи.	32
2.3 Проектування алгоритмів роботи системи	47
2.4 Реалізація програмного продукту	49
Висновки до розділу 2	50
РОЗДІЛ 3. Інтерфейс та порядок роботи з грою	51
3.1. Порядок встановлення та налаштування системи.....	51
3.2 Тестування роботи програмного продукту.....	54
Висновки до розділу 3	57
ВИСНОВКИ	59
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	61
ДОДАТКИ	62

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

- IDE – (англ. Integrated development environment) – комплексне програмне рішення для розробки програмного забезпечення
- UI – (англ. User Interface) – користувацький інтерфейс
- ОС – операційна система
- API – (англ. Application Programming Interface) – інтерфейс програмування застосунків

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

ВСТУП

Актуальність теми. Станом на 2023 рік комп'ютерні ігри є одним з найкрупніших секторів в індустрії розваг. Так загальна виручка від відеоігор за минулий рік склала 220 мільярдів доларів. Це в три рази більше, ніж у глобальної індустрії кінематографу, яка становить всього 77 мільярдів. По статистиці в світі 3,09 мільярда активних гравців, тобто майже 40% відсотків світового населення. В ігри грають далеко не тільки діти, але і велика кількість дорослих. Так вік 80% гравців становить старше 18 років.

Займатися розробкою ігор в наш час можуть не тільки великі компанії, але і маленькі студії, гра навіть може бути створена силами однієї людини. Поріг входу завдяки сучасному програмному забезпеченню значно знизився. На ринку існують різні безкоштовні рушії, які дозволяють швидко і зручно створювати ігри на самі різні платформи людям, які знайомі з програмуванням навіть на початковому рівні.

Зважаючи на вищеназвані факти, розробка ігор є перспективною, але й висококонкурентною діяльністю. Одним із найпопулярніших жанрів комп'ютерних ігор є шутери від першої особи. Ігри такого жанру почали розвиватися на початку 90, пройшли з тих пір великий шлях та не втратили актуальності. Такі ігри розраховані на доволі широку аудиторію.

Метою кваліфікаційної роботи є проектування та розробка ігри в жанрі шутер від першої особи. Встановлена мета обумовлює наступні завдання:

- вивчення можливостей ігрового рушія Unity;
- проектування архітектури програмного забезпечення;
- розробка алгоритмів гри;
- реалізація програмного продукту;
- тестування готової гри.

Об'єктом дослідження є розробка ігор з тривимірною графікою на основі існуючого ігрового рушія.

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

Предметом дослідження є створення шутеру від першої особи на основі ігрового рушія Unity.

В роботі було використано методи об'єктно-орієнтованого проектування та об'єктно-орієнтованого програмування.

Розроблена система може бути використана для розваги гравців.

Пояснювальна записка до кваліфікаційної роботи містить **66** сторінок, **20** ілюстрацій та **2** таблиці, список літературних джерел містить **12** найменувань.

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

РОЗДІЛ 1. АНАЛІЗ НАПРЯМКІВ ВИКОРИСТАННЯ КОМП'ЮТЕРНИХ ІГОР В ЖАНРІ ШУТЕР ВІД ПЕРШОЇ ОСОБИ

1.1 Постановка задачі

Шутер від першої особи - це популярний жанр відеоігор, де головна задача - це знищення ворогів з різної вогнепальної зброї, інколи зустрічається зброя ближнього бою на випадок, якщо закіняться патрони. Також в грі присутній пошук на карті різноманітних предметів, крім зброї, патронів, аптечок, також можуть зустрічатися ключі для дверей і різні сюжетні предмети. Графіка в грі повністю тривимірна або псевдо-тривимірна, що мало місце в старих іграх або зустрічається в сучасних ретро іграх, камера симулює вид «з очей», тобто гравець не бачить персонажа повністю, а на екрані показано лише його зброю. Управління відбувається за допомогою клавіатури і миші, можна також грати на геймпадах. Загалом шутер має досить динамічний ігровий процес, де від гравця потребується швидкість реакції та точність рухів.

Дана кваліфікаційна робота ставить ціллю створити гру з базовими механіками шутера. Розробка даної гри складається з наступних задач, які потрібно вирішити:

1. Пересування гравця по карті та керування камерою, можливість збирати ключі та відкривати двері.
2. Механіка стрільби різного типу зброї. Витрачання та збір боєприпасів. Влучання по об'єктам, які можуть отримувати пошкодження.
3. Створення та анімація тривимірних моделей ворогів.
4. Штучний інтелект для ворогів, який буде включати в себе стрільбу по головному герою та різні режими переміщення: очікування, патрулювання території, переслідування головного героя.
5. Створення декількох тривимірних рівнів та їх наповнення, яке полягатиме в тому, щоб наповнити карту ворогами, розташувати на ній різні предмети і прописати в коді умови перемоги.

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

6. Підбір звукового супроводження гри: фонова музика, звуки пострілів, відкриття дверей і т.д.
7. Головне меню, де можна почати нову гру, завантажити минулу та зробити інші налаштування.
8. Тестування програмного продукту на наявність багів.
9. Зборки гри під операційну систему Windows в вигляді окремої програми та по технології WebGL для гри в веб-браузерах.

Роль ігрового персонажу є ключовою для ігрового досвіду. Він є головним об'єктом ігрової механіки. Через ігрового персонажа відбувається взаємодія з оточуючим світом гри. Ігровий персонаж має переміщатися по карті, стріляти з різної зброї, взаємодіяти з об'єктами оточення. За допомогою клавіатури персонаж буде переміщатися в просторі, а завдяки миші міняти кут огляду камери. Ігровий персонаж може ходити, стрибати та присідати. Він буде мати можливість підбирати зброю та відкривати двері. У ігрового персонажа будуть окремі показники здоров'я та броні, які він буде втрачати під час пошкоджень від ворогів та поповнювати від взаємодії з спеціальними об'єктами.

Також надзвичайно важливим аспектом в грі є зброя, яка має великий вплив на ігровий досвід гравця. Зброя в грі буде різноманітною. Кожен тип зброї в грі буде мати унікальні характеристики ведення вогню, що дозволить обрати підходящу стратегію на той чи інший випадок.

Пістолет буде найпершою і найпростішою зброєю в грі. Пошкодження, яку він наносить суперникам, будуть незначні. Пістолет буде стріляти поодинокими пострілами, але мати високу швидкість стрільби, що дозволить зручно знищувати поодинокі цілі, але не велику кількість ворогів. Обойми і перезарядки в нього не буде. Вогонь буде вестися до тих пір, поки присутні патрони.

Дробовик – це друга зброя в арсеналі гравця. Він має розсіювання та малу дальність стрільби. Якщо в суперника поцілити впритул, то він отримає значні пошкодження, але для стрільби на відстані по одинокій цілі дробовик вже не такий ефективний, тому що більшість куль не влучить в ціль, але якщо суперників велика кількість, то з дробовика буде доцільно вести вогонь.

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

Дробовик буде мати великий час на перезарядку, щоб з нього не можна було вести швидкий вогонь.

Автомат – це третя стрілецька зброя в грі. Головною характеристикою автомата буде висока швидкість стрільби, він за короткий проміжок часу може випускати велику кількість куль, тому гравцеві потрібно бути обережним і стріляти влучно, щоб не змарнувати велику кількість патронів. В нього висока дальність стрільби, тому він може бути ефективним для ураження великої кількості ворогів на тих відстанях, куди не доставить дробовик.

Ракетниця – це четверта зброя. Важка та потужна. Вона випускає снаряд, за польотом якого можна спостерігати. При влучанні в ціль він вибухає та наносить пошкодження суперникам в зоні ураження. Для ракетниці не буде багато патронів та вона матиме великий час перезарядки. Вона добре діє як на великій, так і на середній дистанції. Використовувати її впритул буде небезпечно, тому що вона може нанести пошкодження і головному герою.

Плазмова гвинтівка – це останній футуристичний вид зброї. Вона подібна до автомату та має високу скорострільність, але відрізняється тим, що її кулі будуть не миттєво влучати в ціль, а видимо летіти по карті. Плазмова гвинтівка буде наносити більш значні пошкодження, чим автомат, але кількість боєприпасів буде більш обмеженою. Вона буде найбільш ефективною для швидкого знищення великої кількості ворогів, як на короткій, так і на дальній дистанції, хоча в останньому випадку деяка частина зарядів може промахнутися.

В цілому, потрібно буде приділити важливе значення балансу, тому що зброя повинна бути збалансованою, щоб уникнути ситуацій, коли деякий тип зброї стає надто сильним і гравець тільки його і використовує або зброя стає занадто слабкою та втрачає будь який сенс стріляти з неї. Баланс забезпечує справедливу гру та різноманітність варіантів стратегій.

Створення рівнів – це наступна важлива складова розробки шутеру. На ігрових рівнях будуть відбуватися усі дії гри. Переміщення, пошук предметів та боротьба з ворогами. Дизайн рівнів включатиме в себе розташування предметів, ворогів, які потрібно розмістити так, щоб вони створювали цікаві ситуації для

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

гравця, для розрішення яких потрібно буде підібрати свій підхід. Також на карті потрібно зробити приховані кімнати, які гравець зможе знаходити і отримувати нагороду в вигляді поповнення арсеналу зброї та боєприпасів. Рівні не повинні буде занадто короткими або занадто довгими.

Важливою частиною буде футуристична атмосфера. Потрібно буде показати, що гравець знаходиться на космічній станції. Для цього підібрати потрібні текстури, які відображатимуть інтер'єр космічної станції: стіни, двері, різні об'єкти. Показати види на відкритий космос, щоб гравець відчував, що дії справді відбуваються у далекому майбутньому на космічній станції, яку атакували вороги.

Наступним кроком в розробці шутеру буде створення ворогів. Цей етап розробки неможливо переоцінити, оскільки від нього багато в чому залежить те, наскільки гра буде цікавою. В даній грі буде представлено всього три типи ворогів, але вони будуть значно відрізнятися один від одного, та створювати різні ігрові ситуації.

Першим типом ворогу буде ворог з гвинтівкою. Він рухається в напрямку гравця та періодично стріляє, якщо між ворогом і гравцем немає перешкод, які би стали на заваді польоту кулі. Постріл відбувається миттєво та віднімає певний процент здоров'я у гравця. Ворог не може підходити занадто близько. Є дистанція, при досягненні якою, ворог зупиняється та просто слідує за гравцем, стріляючи з такою ж періодичністю.

Другий тип ворогу – це ворог з ракетницею. Він схожий за поведінкою до першого. Але головна різниця полягає в тому, що він стріляє не з гвинтівки, постріли якої відбувається миттєво та від них не можна ухилятися, а він запускає в напрямку гравця ракету, яку можна побачити, та вона має швидкість польоту, так що гравець має можливість ухилитися від снаряда, якщо він летить з великої дистанції.

Третій і останній тип ворогу – це вороги ближнього бою, які біжать до гравця та коли підходять до нього впритул атакують його. Швидкість їх переміщення вище, ніж у гравця, так що він не буде мати змоги від них втікати

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

безкінечно, а їх потрібно буде вчасно знищувати на підході, щоб вони не встигли нанести пошкоджень головному герою, тому що їх атака буде віднімати доволі багато здоров'я.

Для кожного типу ворога потрібно буде розробити свою модель та анімації. Вони мають зовні відрізнятися один від одного. Планується робити їх за допомогою технології вокселів, що значно спростить створення моделей.

Ці три типи ворогів, правильне їх розташування на карті та різні комбінації мають зробити гру насиченою та цікавою. Гравцю доведеться зіштовхуватися з різними ситуаціями та шукати правильні шляхи їх вирішення у вигляді доцільного підбору зброї для атаки та тактики ведення бою. Так, наприклад, будуть кімнати наповнені великою кількістю ворогів ближнього бою, і гравцеві потрібно буде зуміти швидко їх знищити, використовуючи зброю та боєприпаси, який він раніше знайшов та зберіг під час боїв, дотримуючись правильної тактики.

На карті будуть розташовані двері, які будуть розділяти зони. Гравець буде мати можливість їх відчиняти, коли підійде впритул та натисне клавішу взаємодії. Через деякий час двері будуть закриватися автоматично. Деякі двері потребуватимуть наявності спеціального ключа, який потрібно буде знайти на карті, щоб відкрити двері.

Також двері активуватимуть противників, які будуть знаходитись у режимові бездіяльності, поки гравець не підійшов до певної локації. І як тільки двері відчиняться, суперники пробудяться та почнуть рухатися, переслідувати героя, стріляти по ньому. Тобто, двері будуть взаємодіяти з програмуванням поведінки ворогів.

На карті будуть розташовані аптечки та броня, які будуть поповнювати здоров'я та, відповідно, броню героя, вони будуть мати ліміт. Так здоров'я не можна буде зібрати більше, ніж сто одиниць, а броні більше, ніж двісті одиниць.

Крім аптечок і броні, можна буде піднімати зброю та патрони. Зброя, якщо вона є в наявності у героя, буде збільшувати кількість патронів. Зброя та

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

боєприпаси інколи будуть випадати з тіл ворогів, а також будуть розташовані в секретних місцях на карті.

В грі буде наявний графічний інтерфейс, який буде відображати скільки у героя є здоров'я, броні та патронів до обраного типу зброї.

В грі буде головне меню, де можна буде почати нову гру, продовжити існуючу, обравши один з рівнів, або покинути гру. Під час проходження рівнів можна буде зайти в меню паузи, з якого можна буде повернутися в головне меню чи вийти з гри.

Сюжет гри буде простим, оскільки часто в шутерах він не є занадто важливою складовою, і нерідко він існує лише для галочки та пояснюється в декількох словах, що особливо часто зустрічалось на зорі розвитку шутерів. Сюжет даної гри полягатиме в тому, що на космічну станцію напали прибульці. Головному герою потрібно буде зачистити станцію від ворогів, подорожуючи по рівнях станції.

Вищезазначені складові дозволять створити повноцінну гру з основними механіками шутеру. Успіх її у гравців буде залежати не тільки від якості програмного коду, щоб не було великої кількості критичних помилок, які можуть зруйнувати будь-який проект, або від візуальної та звукової складової, які можуть заохочувати людей запустити гру, дивлячись на скріншоти та рекламні ролики, а успіх буде в великій мірі залежати від гейм-дизайну, тобто від того, наскільки цікаво буде побудований ігровий процес, чи зможе він утримати увагу та інтерес гравця.

1.2 Огляд існуючих рішень створення шутеру

Хоча комп'ютерна індустрія невпинно розвивається вже впродовж десятиліть, основні концепції багатьох ігор залишились незмінними, шутери не є виключенням. Приступаючи до створення гри, можна орієнтуватися на класичні ігри навіть тридцятирічної давнини. В багатьох випадках це навіть доречніше, оскільки сучасні AAA ігри є занадто складними, щоб відтворити

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

щось подібне силами одного розробника, тому як взірць для даної кваліфікаційної роботи буде взято гру Doom 2.

Звісно ігрові рушії з тих часів значно і навіть докорінно змінилися, якщо дивитися на таку застарілу технологію, як raycasting, що в DOOM використовувалась для побудови псевдо-тривимірної графіки, то зараз такий підхід може зустрічатися лише в розробці любителів ретро технологій, а в сучасних рушіях скрізь використовується справжня тривимірна графіка.

Але в даному випадку було обрано DOOM не як гру, з якої я буду запозичатись графічна частина, а як гру, де буде відбуватися орієнтування на елементи ігрового дизайну, тобто буде розглядатися те, які є види зброї, який штучний інтелект у противників, як побудовані карти і т.д. Ці елементи будуть взяті за основу кваліфікаційної роботи.

В DOOM є вісім типів зброї, кожна з якої значно відрізняються за своєю дією, це різноманіття призводить до того, що гравець має використовувати зброю з розумом, підходячи до кожної окремої ситуації. Наприклад, економити боєприпаси для потужної зброї, використовуючи більш просту, пістолет чи навіть бензопилу, обирати тип зброї залежно від того, на якій відстані від гравця знаходяться вороги та яка їх кількість. В цілому зброя дуже схожа на той арсенал, який був описаний в постановці задачі, хоча є деякі відмінності.

Кулак та бензопила – це зброя ближнього бою. Вони можуть бути використані в тому випадку, коли закінчились боєзаряди для вогнепальної зброї або їх залишилось в обмалі, а тому потрібно економити.

Пістолет є початковою зброєю гравця і має середні характеристики. Він зручний для стрільби на відстані та може бути корисним на початкових етапах гри.

Дробовик є потужною вогнепальною зброєю, здатною наносити значної шкоди на ближній дистанції. Він здатен розірвати ворогів та, як уже було сказано, найбільш ефективно знешкоджує суперників на короткій дистанції.

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

Супер дробовик це покращена версія дробовика. Він є ще більш потужнішою зброєю і наносить більше пошкоджень. Він має подвійні стволи, що дозволяє гравцеві вистрілювати двома патронами одночасно.

Кулемет це швидкострільна зброя, здатна роботи багато пострілів за короткий час. Ця зброя наносить швидкі, але не дуже великі пошкодження, і може бути ефективна проти ворожих сил, особливо коли вони знаходяться на відстані.

Ракетниця - це зброя, що стріляє вибуховими ракетами, за польотом яким можна спостерігати. При влучанні по цілі ракета наносить великі пошкодження. Вона зазвичай використовується для боїв на дальній дистанції та проти груп ворогів.

Плазмова гвинтівка є потужною зброєю, що вистрілює заряди плазми, за польотом яким, як і за ракетою, можна спостерігати. Плазмова гвинтівка наносить швидкі та потужні пошкодження, особливо проти броньованих ворогів.

БФГ 9000 є найсильнішою зброєю в грі. Вона випускає потужний енергетичний снаряд, який розриває ворогів на шматки та наносить значні пошкодження по великій зоні. Але вона витрачає багато зарядів, тому нею потрібно користуватися обмежено та з розумом.

Такого арсеналу зброї цілком достатньо для гри.

При створенні ігрових рівнів можна прислухатися до порад культового ігрового дизайнера Джона Ромеро, а серед його порад є наступні:

- завжди використовувати контраст в побудові рівнів, чергувати світлі і темні, вузькі та широкі локації;
- дуже важливо на кожному рівні добавляти декілька секретних локацій, які може відкрити гравець;
- створювати легко впізнаваних орієнтирів у кількох місцях для полегшення навігації;
- розмістити стратегічний запас предметів перед важкою битвою;
- не створювати пастки чи зони без виходу, які будуть змушувати гравця перезавантажувати гру.

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

Вороги є декількох типів. Ті, що атакують здалеку, та ті, що зблизька. Вороги, постріл яких миттєво влучає, та ті, які запускають снаряд, від якого можна ухилитися. Доцільно буде звернути увагу на наступних монстрів.

Зомбі є одним з основних ворогів, з якими гравець зустрічатиметься протягом гри.

Ці вороги в Doom 2 є незграбними противниками. Вони рухаються повільно та безцільно, а їхня атака - це в пострілі з пістолета, хоча на зображенні ворога присутня гвинтівка. Їх стрільба може бути точною, але низька швидкість дає гравцеві можливість ухилитися від атаки.

Зомбі мають обмежену кількість здоров'я, всього 20 очок, що робить їх вразливими для атак гравця. За декілька пострілів пістолетом або влучних ударів кулаком чи бензопилою зомбі можуть бути легко знищені. Для знищення великої кількості зомбі використовують дробовик.

Після смерті зомбі можуть залишити пістолет або патрони, які гравець здатен підбирати. Це може бути важливим джерелом додаткової амуніції для гравця.

Біс є одним з характерних ворогів DOOM 2, виглядає, як демонічне створіння, прибулець з аду, з якими гравцеві також доведеться зустрічатися постійно, починаючи з самого початку гри, але він дещо менш розповсюджений, ніж зомбі. В цілому їх можна побачити на всіх рівнях гри, часто присутні в великій кількості. Значну небезпеку для гравця не представляють.

Біси в Doom 2 мають гнучку поведінку та можуть рухаються досить швидко в різних напрямках. Вони намагаються знайти оптимальну позицію для атаки, після чого стріляють у гравця. Вони також можуть використовувати свої стрибки для ухилення від атак гравця.

Біси здалеку стріляють вогняними кулями, які вони запускають у гравця. Вогняні кулі наносять досить велике пошкодження здоров'ю гравця, тому від них бажано ухилятися. Впритул біси можуть атакувати кігтями, тому потрібно бути обережним на близькій дистанції.

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

Біси мають 60 очок здоров'я, це в 3 рази більше ніж у зомбі, тому застосовувати проти них пістолет не дуже ефективно, біс часто може витримати навіть постріл з дробовика.

Демони – це жахливі монстри з аду, які виглядають як здорові м'ясисті тварини з червоною шкірою та гострими зубами. Вони дуже небезпечні вороги, що гравцеві зустрічатимуться на його шляху.

Демони є досить швидкими та агресивними ворогами. Вони намагаються прорватися до гравця, оскільки не володіють далекобійними атаками. Вони атакують лише впритул своїми кігтями та зубами, чим можуть наносити великі пошкодження.

У демонів доволі багато здоров'я. 150 одиниць. Вони можуть бути вбиті не тільки на дистанції з декількох пострілів потужної зброї, але проти них також ефективна бензопила, яка є зброєю ближнього бою. Вона дозволить значно зекономити боєприпаси. Це демонструє гарний дизайн ворогів, що в Doom 2 потрібно використовувати різну зброю.

Аналізу цих трьох ворогів є достатньо, щоб їх взяти за основу для створення ворогів у кваліфікаційній роботі. Важливо звернути увагу на те, по якій траєкторії ходять суперники, з якою частотою атакують, як довго можуть переслідувати героя. Приблизно знати, де саме, скільки, і як розміщати ворогів. Кількість здоров'я та пошкодження, які вороги наносять, в DOOM відрізняють за рівнем складності, це також потрібно взяти на замітку. Оскільки в кваліфікаційній роботі не потрібно буде робити занадто важку або занадто легку складність гри, коли не буде ніякого змагання.

Інтерфейс в грі Doom 2 складається з різних елементів, які допомагають гравцю контролювати гру та отримувати необхідну інформацію. Інтерфейс Doom 2 складається з наступних елементів:

1. Показники здоров'я та броні продемонстровано у вигляді цифр, які відповідно показують залишок очок здоров'я та броні у гравця. Також зрозуміти, в якому стані знаходиться головний герой, можна по обличчю, що розташовано в центрі панелі інтерфейсу. Чим більше

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

пошкоджень отримав гравець, тим більш це відображається на стані героя у вигляді кровотечі та виразу обличчя.

2. Показники боєприпасів. Кількість патронів до активної зброї відображається в лівому нижньому кутку, а загальні показники амуніції у правому нижньому кутку екрана, де вказано не тільки скільки є тих чи інших снарядів, але й те скільки їх можливо всього зібрати.
3. Наявна зброя. За зброю відповідає панель з цифрами. Якщо число жовте, то зброя присутня в арсеналі, а якщо число сіре, то її нема та її потрібно ще знайти в майбутньому.
4. Карти-ключі. Вони бувають трьох кольорів. Сині, жовті та червоні. Коли їх гравець знаходить, то вони відображаються на панелі, коли їх нема, то там пусте місце. Карти-ключі потрібні для того, щоб відкривати двері.

Були описані головні елементи гри DOOM 2. В цілому, якщо намагатися слідувати описаним механікам DOOM 2, то можна створити гру з цікавим та збалансованим ігровим процесом. Звісно не буде відбуватися копіювання один в один, а ряд елементів буде змінено, доповнено або вирізано.

1.3 Опис інструментів розробки

Для створення Шутеру будуть використовуватись різні інструменти. Багатоплатформовий рушій Unity. Ігрова логіка буде писатися на мові програмування C#. Моделі суперників будуть створені в MagicaVoxel, карти побудовані в редакторі тривимірної графіки Blender, текстури намальовані в програмі Aseprite.

Як вже було зазначено, програмний продукт буде створюватись на ігровому рушії Unity. Хоч Unreal Engine, інший лідируючий на ринку рушій, історично був створений для шутерів, але він занадто важкий, розмір простого проекту без оптимізації може сягати 1 гігабайта. Тоді як ціла гра на Unity може помістися в 100-200 мегабайт, що має величезне значення для мобільних ігор.

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

Також перевагою Unity є можливість робити зборки WebGL, що дозволяє грати прямо в браузері. Ті складні налаштування графіки, шейдерів та спецефектів, які доступні в Unreal Engine, не будуть використовуватися при створенні даного проекту.

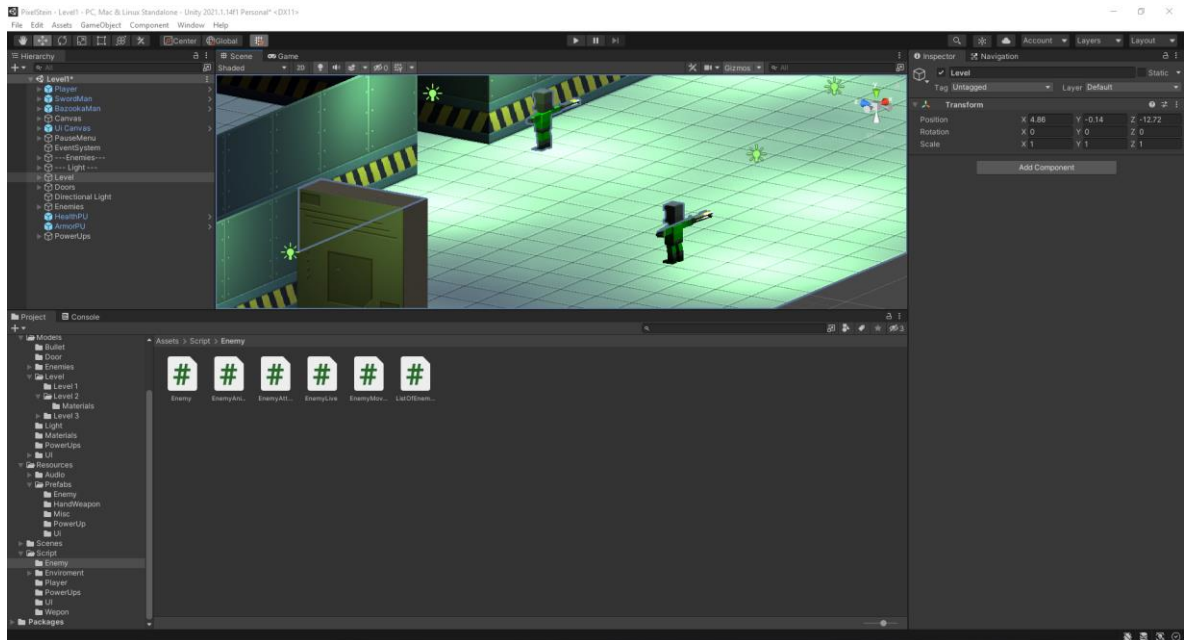


Рис. 1.1 Користувацький інтерфейс рушія Unity

Unity гарно документований рушій та має велику та активну спільноту розробників, які на різних форумах дають відповіді на самі різноманітні задачі та проблеми, які можуть зустрітися під час розробки. Це забезпечує швидкий доступ до важливої інформації. На форумах є можливість обговорювати проблеми та отримувати поради від інших розробників. Також є великий вибір плагінів, які можуть знадобитися в ході створення програмного продукту.

Unity повністю задовольняє вимогам розробки шутеру. Рушій дозволяє без проблем імпортувати тривимірні моделі, текстури та анімації. Присутнє їх тонке налаштування. Для роботи з анімацією існує зручний аніматор, а також система Ragdoll.

Потужний та інтуїтивно зрозумілий візуальний редактор сцен допоможе розробнику зручно створювати, налаштовувати та візуально редагувати ігрові сцени. Drag-and-Drop інтерфейс дозволяє просто перетягувати та розміщувати об'єкти на сцені: будь то ігрові персонажі, чи джерела світла, які динамічно

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

відображаються на сцені. Таким чином легко створити потрібну композицію ігрового рівня.

Ще однією важливою частини розробки гри є наявність компонента NavMesh, який відповідає за пошук шляху на карті та має різноманітні налаштування. Він створює навігаційні мережі для штучного інтелекту, які дозволяють агенту рухатись до цілі, минаючи перешкоди, можливе динамічне оновлення карти. Присутність NavMesh дуже важлива річ, оскільки написання власної логіки пошуку шляху - це дуже непросте завдання, яке далеко не обмежується одним алгоритмом A*, а потребує вміти аналізувати тривимірну карту, розбивати її на графи, створювати зони з різною ціною переміщення та інше. Тому використання NavMesh розумне рішення.

Все це та інше повністю виправдовує вибір Unity в якості рушія. Він надає розробнику широкий спектр функціональності, гнучкості та підтримку спільноти.

Мовою програмування буде обрано C#. Це сучасна об'єктно-орієнтована мова програмування з безпечною системою типізації для платформи .NET. C# був розроблений та підтримується компанією Microsoft. Це одна з найпопулярніших мов програмування у світі, входить в п'ятірку найбільш вживаних мов. C# дуже схожа до C++, але є дещо простішою та зручнішою для нового користувача. На C++ написаний runtime Unity.

На даний момент в рушії Unity C# – це єдина мова програмування для написання скриптів, хоча раніше використовувались ще Unity Script та Boo. C# інтегрується безпосередньо з Unity API, що дозволяє розробнику отримати доступ до всіх можливостей і функцій рушія. Сюди відноситься робота зі сценами, об'єктами, анімацією, фізикою, аудіо та інше.

Також в Unity присутні корутини. Це спеціальний тип функцій в C#, які дозволяють створювати асинхронні операції та відкладати виконання коду. Вони корисні для створення анімацій, затримок часу, асинхронних завдань та багато чого іншого.

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

Головною альтернативою є візуальний скриптинг Bolt, хоча він і дозволяє вирішити переважну кількість задач, але він підходить більше для початківців, які не мають досвіду з мовами програмування. Bolt не є таким потужним та гнучким, як C#, де розробник має повний контроль над своїм кодом, може створювати складні структури даних, використовувати класи, інтерфейси, наслідування та інші ООП-концепції, що дає можливість писати більш ефективний та оптимізований код. Тому в даному випадку візуальний скриптинг Bolt не розглядається як варіант, яким можна замінити традиційний скриптинг на мові програмування C#.

По замовченню разом з рушієм Unity встановлюється IDE MonoDevelop. Це легке та швидке середовище розробки, яке не споживає багато ресурсів, але більш надійним та функціональним є Microsoft Visual Studio, який має повну інтеграцію з Unity. В налаштуваннях інсталяції Visual Studio є пункт, де можна обрати, що потрібна підтримка Unity.

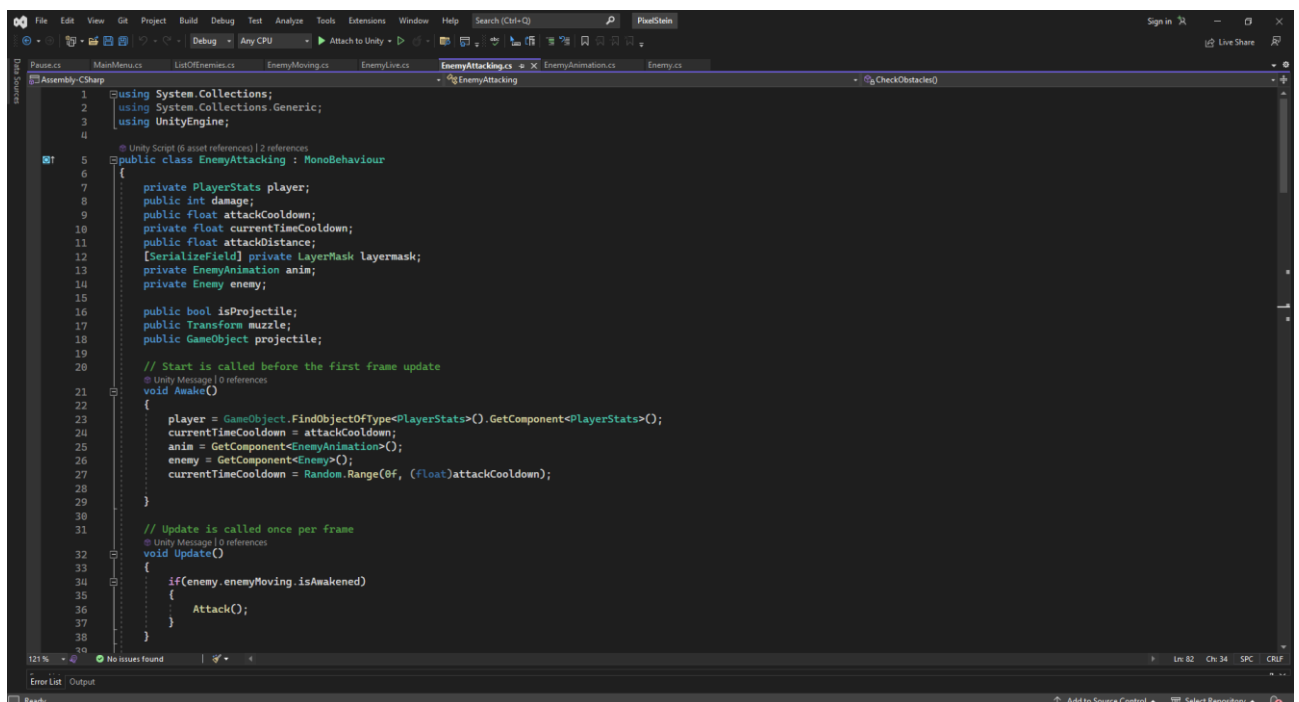


Рис. 1.2 Користувацький інтерфейс Microsoft Visual Studio

Штучний інтелект IntelliCode під час написання програмного коду надає зручну функцію, яка, зважаючи на контекст, використовуючи машинне навчання, автоматично доповнює строки коду, що може значно пришвидшити

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

роботу. Також наявні інші розширені можливості редагування коду, такі як підсвічування синтаксису, перейменування по всім файлам.

Середовище Rider не розглядається, тому що воно не є безкоштовним.

В якості редактора по створенню тривимірних рівнів та анімацій персонажів було обрано Blender. Це легкий, швидкий та потужний редактор. Але головним чинником в обранні саме цієї програми стало те, що Blender це безкоштовна програма з відкритим кодом, а ціна має велике значення, тому що, наприклад, 3ds Max коштує дуже дорого.

Відкритий програмний код свідчить про наявність великої кількості розширень та плагінів, які створюються користувачами по всьому світові. Розширення та плагіни можуть значно полегшити роботу та будуть використовуватися в даній кваліфікаційній роботі.

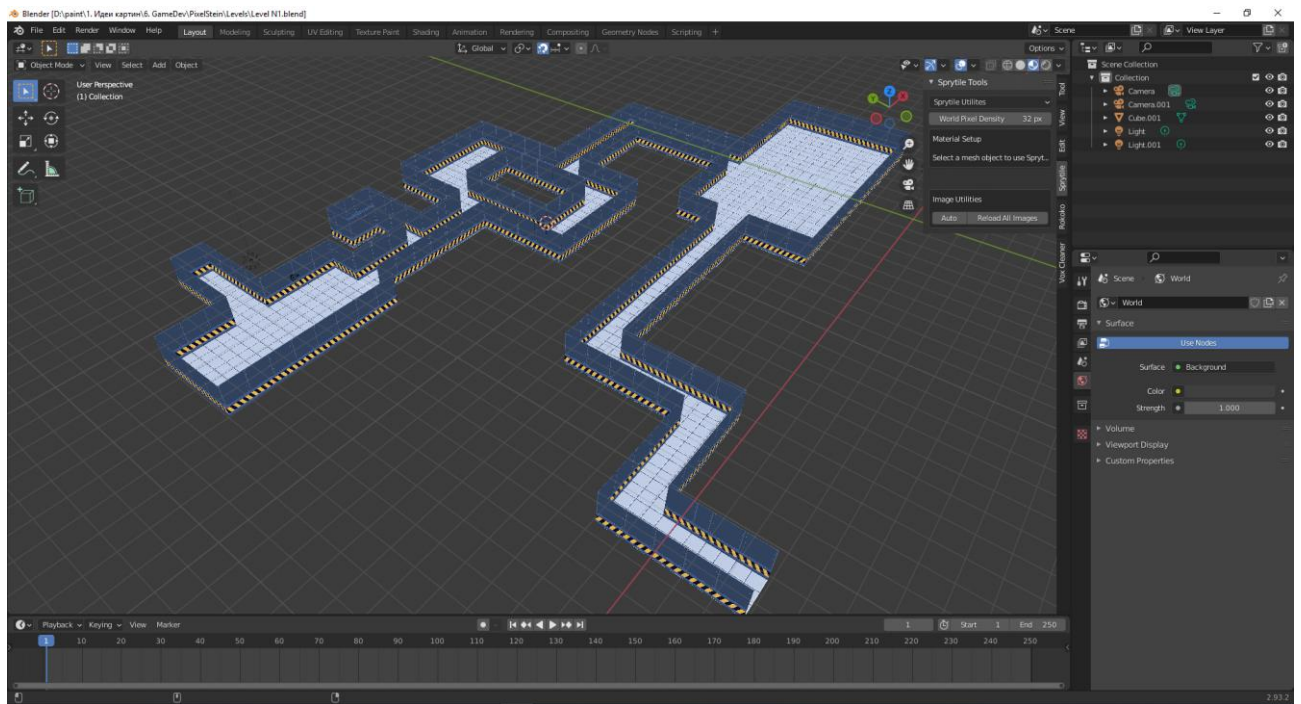


Рис. 1.3 Користувацький інтерфейс Blender

Оскільки Blender популярний редактор, тому він має велику та активну спільноту користувачів, що дозволяє отримувати доступ до різноманітних навчальних матеріалів, що допоможе швидко та зручно вирішувати ті проблеми та задачі, які виникнуть у процесі роботи.

Серед безлічі додатків до Blender для простого та швидкого створення рівнів буде застосований безкоштовний додаток Sprytle, який використовує

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

тайлову систему об'єктів в стилі двовимірних ігор. Достатньо завантажити текстуру та вказати розмір тайлу, після чого можна приступити до малювання рівня, використовуючи зручну систему прив'язок. Готовий об'єкт експортується до Unity без зайвих складних налаштувань текстур, тільки потрібно правильно вказати нормалі, щоб не було чорних текстур. Звісно, недоліком такої системи є те, що вона не дозволяє створювати складні об'єкти, але для цілей даної кваліфікаційної роботи можливостей Sprytle буде достатньо.

Blender дозволяє зручно створювати, налаштовувати та експортувати анімації до Unity. Для цих цілей буде використана система скелетної структури, яка реалізована через створення кісток, визначення їх ієрархії та інші елементи. Можливе застосування інверсної кінематики, яка полегшує анімацію персонажу. Скелети прив'язуються до готових моделей та налаштовується те, наскільки сильно модель буде деформуватися під час руху. Далі відбувається створення анімацій завдяки ключовим кадрам, які встановлюються для різних частин тіла, визначаючи позицію, поворот, масштабування та інші властивості. Після чого програма автоматично налаштовує плавний перехід між кадрами. Це дуже зручно. Різні анімації можна експортувати одним файлом.

Персонажі та зброя для кваліфікаційної роботи будуть змодельовані в безкоштовній програмі MagicaVoxel. Це зовсім проста та легка програма для воксельного моделювання. Voxel походить від комбінації слів об'єм (volumetric) і піксель (pixel). З цієї назви можна зрозуміти, що вокселі – це аналог пікселів для тривимірного простору. Моделі персонажів будують з кубиків, кожен з яких має свій колір. Це більш швидкий та зручний спосіб моделювання, ніж ті, які наявні в Blender.

					КН.КР.Б – 122 – 23 – ПЗ	Арк.
						24
Змн.	Арк.	№ докум.	Підпис	Дата		

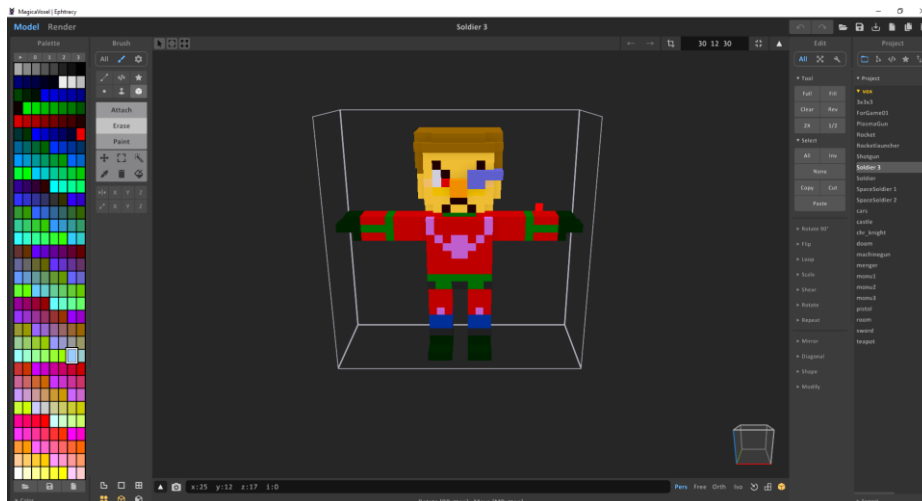


Рис. 1.4 Користувацький інтерфейс MagicaVoxel

Хоч будувати такі моделі досить просто, вони можуть містити дуже багато полігонів та не завжди придатні для анімації, тому моделі потребуватимуть оптимізації, щоб об'єднати воксели в суцільну форму, де це потрібно. Текстура для воксельної моделі не буде потрібна, так як кольори будуть назначатися через властивості вершини моделі.

Загальний вигляд воксельних моделей може здатися занадто простим, але є багато ігор, які містять таку або схожу графіку. Найвідомішим прикладом в світі є гра Minecraft, яка має такі ж прості моделі та використовує подібну технологію для випадкової генерації ландшафту.

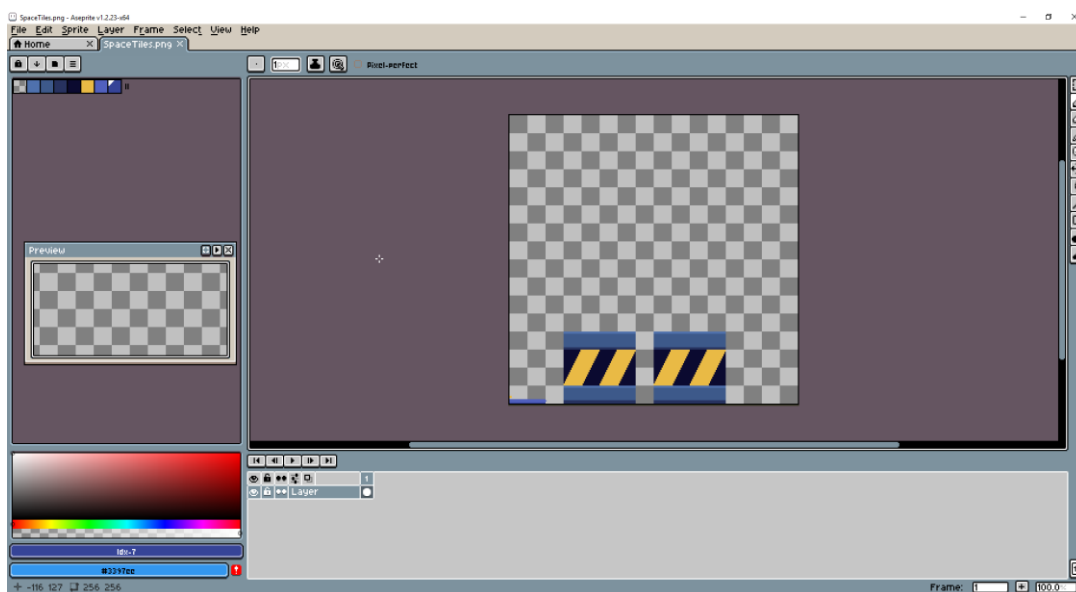


Рис. 1.5 Користувацький інтерфейс Aseprite

Для малювання текстур буде використано графічний редактор Aseprite. Ця програма призначена для створення піксельної графіки та містить широкий та

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

зручний набір інструментів для малювання та редагування таких зображень. Серед інструментів пензлі, заливки, градієнти, кольорові палітри. Можливе використання шарів. Присутнє зручне виправлення помилок.

Вибір обґрунтовується тим, що в грі не буде фото-реалістичних текстур, а персонажі ворогів та моделі зброї будуть воксельними, тому з ними буде добре гармоніювати піксельні текстури, створення яких може бути доволі простим та повністю виконувати задачу по приданню ігровому оточенню футуристичного стилю. Готові зображення експортуються в різних форматах, які повністю сумісні з Blender та Unity.

Висновки до розділу 1

В даному розділі були поставлені задачі по розробці шутера від першої особи, що містило в собі опис зброї, ворогів, користувацького інтерфейсу та принцип побудови карт. Також був розібраний аналогічний програмний продукт, на який буде проведено орієнтування під час розробки гри. А також були розібрані програми, завдяки яким буде створюватись програмний продукт. Були перелічені переваги цих програм, що послугували причиною того, що обрали саме їх.

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

РОЗДІЛ 2. ПРОЕКТУВАННЯ ГРИ В ЖАНРІ ШУТЕР ВІД ПЕРШОЇ ОСОБИ

2.1 Визначення варіантів використання системи

При постановці задачі до кваліфікаційної роботи були висунуті різні функціональні вимоги, що описують, як повинна себе поводити гра. Наявний функціонал гри можна поділити на окремі логічні частини, що відповідають, наприклад, за керування персонажем чи стрільбу. Кожна така логічна частина має свою діаграму прецедентів, яка дозволяє описати систему на концептуальному рівні.

Діаграма прецедентів є одним з інструментів в аналізі та проектуванні системи. Вона представлена у вигляді графа, складовими частинами якого є актори (сутності, що взаємодіють з системою) та варіанти використання, що описують окремі функціональні можливості, дії системи, які можуть бути викликані акторами.

Використання діаграм прецедентів при проектуванні проекту має ряд переваг серед яких:

Чітке визначення функціональності. Діаграми прецедентів допомагають у лаконічній формі визначити функціональні можливості системи з точки зору її користувачів.

Планування та пріоритети. Діаграми прецедентів використовуються на початковому етапі для планування розробки системи та визначення пріоритетів функціональності.

Комунікація між учасниками проекту. Діаграми прецедентів є візуальним засобом комунікації між розробниками, дизайнерами, тестувальниками та замовниками. Вони допомагають уточнити вимоги та налагодити зрозуміле спілкування між усіма сторонами проекту.

На всіх наведених нижче діаграмах прецедентів постійним актором буде гравець, який по різному взаємодіє з системою. Іншими акторами будуть виступати різні складові частини гри, такі наприклад як персонаж, яким буде

					КН.КР.Б – 122 – 23 – ПЗ	Арк.
						27
Змн.	Арк.	№ докум.	Підпис	Дата		

керувати гравець, чи зброя, яку гравець буде використовувати в бою. Взаємодія між цими акторами буде представлена у вигляді варіантів використання.

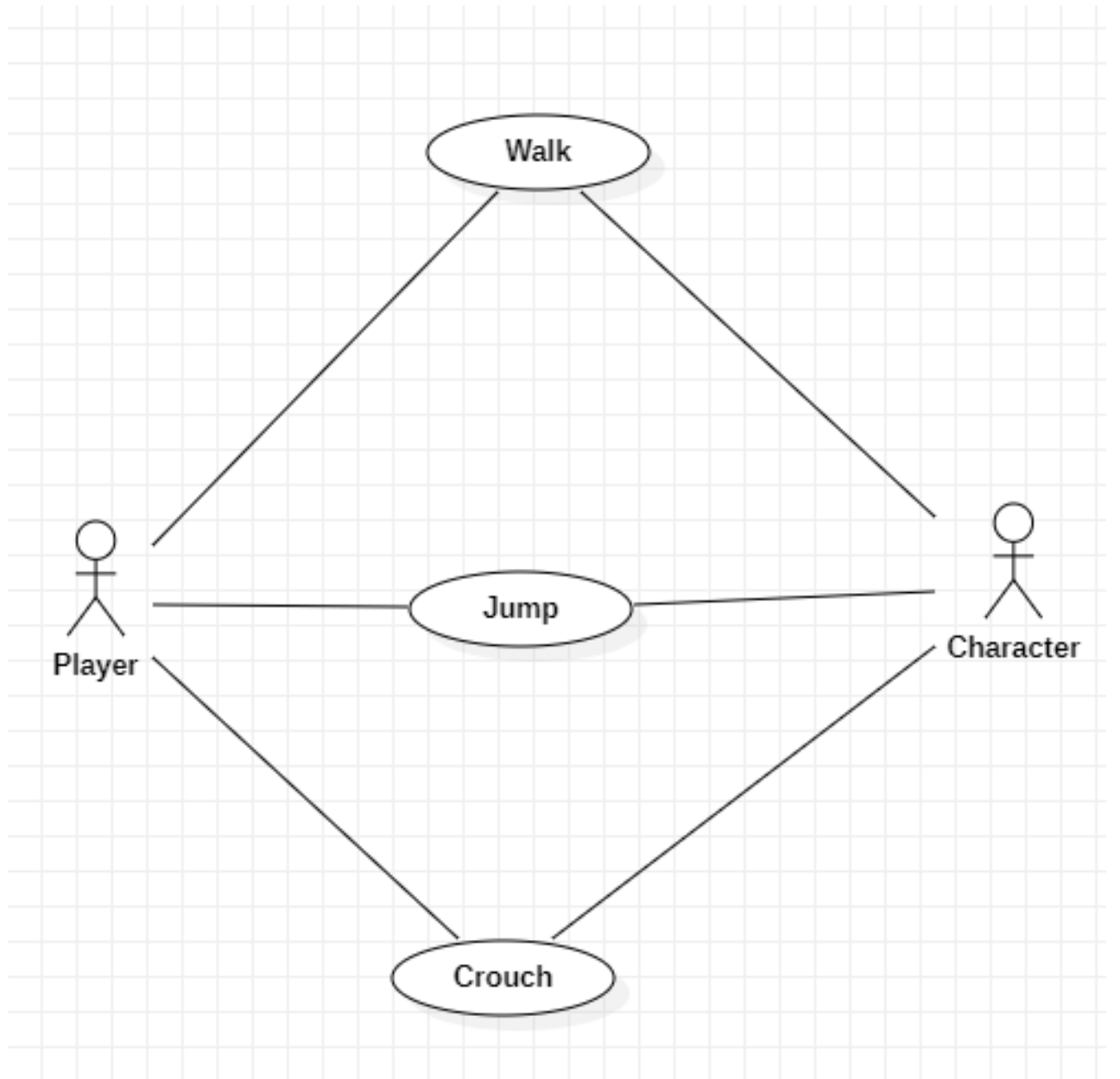


Рис 2.1 Діаграма прецедентів персонажу

Перша функціональність, яка розглядається, - це персонаж, тобто головний герой, яким керує гравець. В нього є три базові функціонали, які відповідають за переміщення по карті. Ходьба, яка завжди відбувається з однаковою швидкістю. Стрибки, які потрібні для подолання перешкод. І останній прецедент – це присідання, яке необхідне, щоб гравець міг пройти в різних низьких місцях.

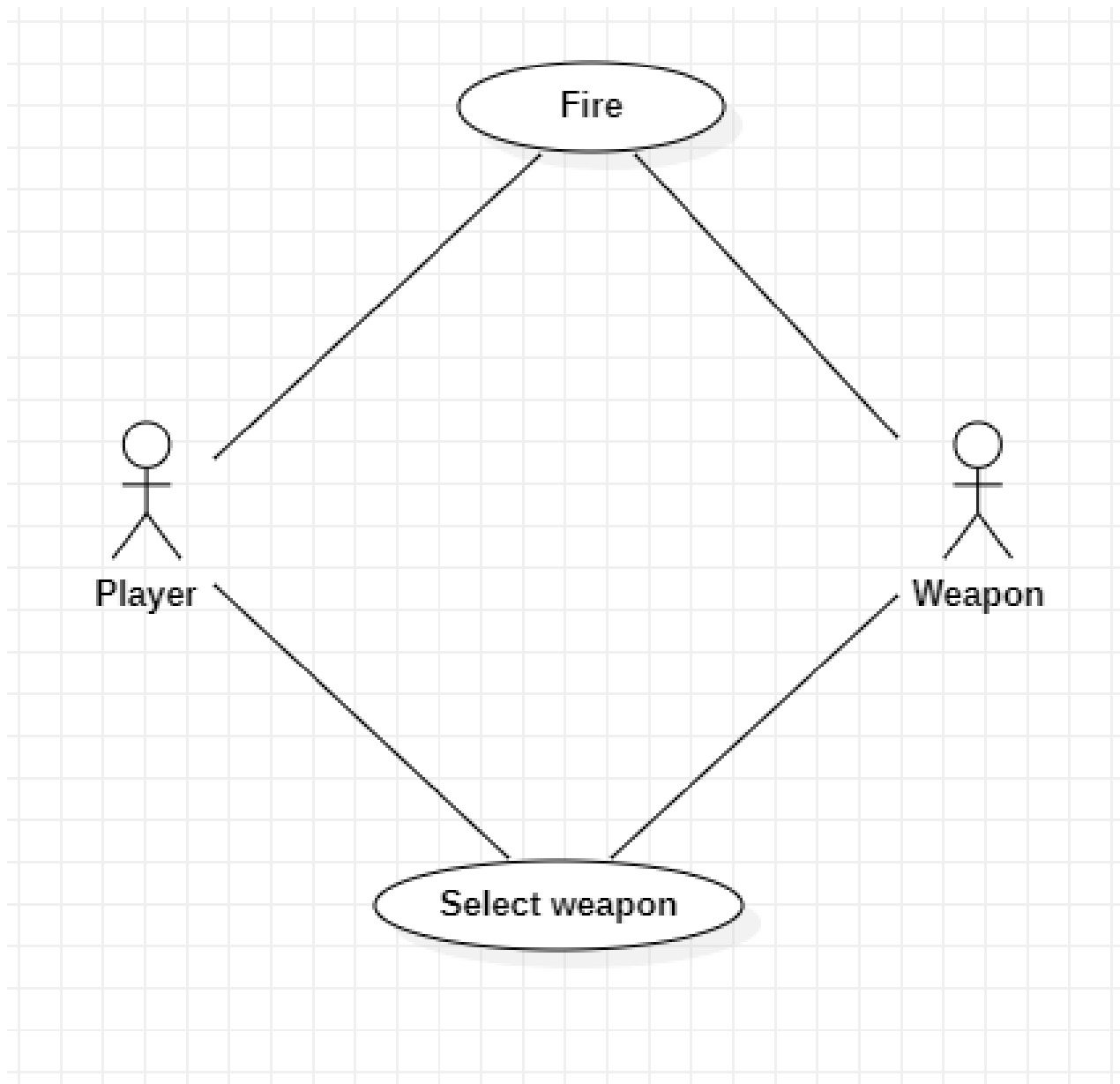


Рис 2.2 Діаграма прецедентів зброї

Друга функціональність – це зброя, яку використовує гравець для знищення суперників. Вона вся вогнепальна. В грі не передбачені перезарядки, які би міг виконувати гравець за своїм бажанням, тому не існує прецеденту, який би відповідав за перезарядку. Зброя стріляє до тих пір, поки в магазині залишаються боєприпаси. Кожна зброя має свій інтервал пострілів. За стрільбу відповідає прецедент Fire. Інший прецедент – це вибір зброї, оскільки в грі наявні декілька видів зброї, які гравець знаходить під час проходження рівнів, він може їх обирати завдяки натисканню на відповідні клавіші клавіатури.

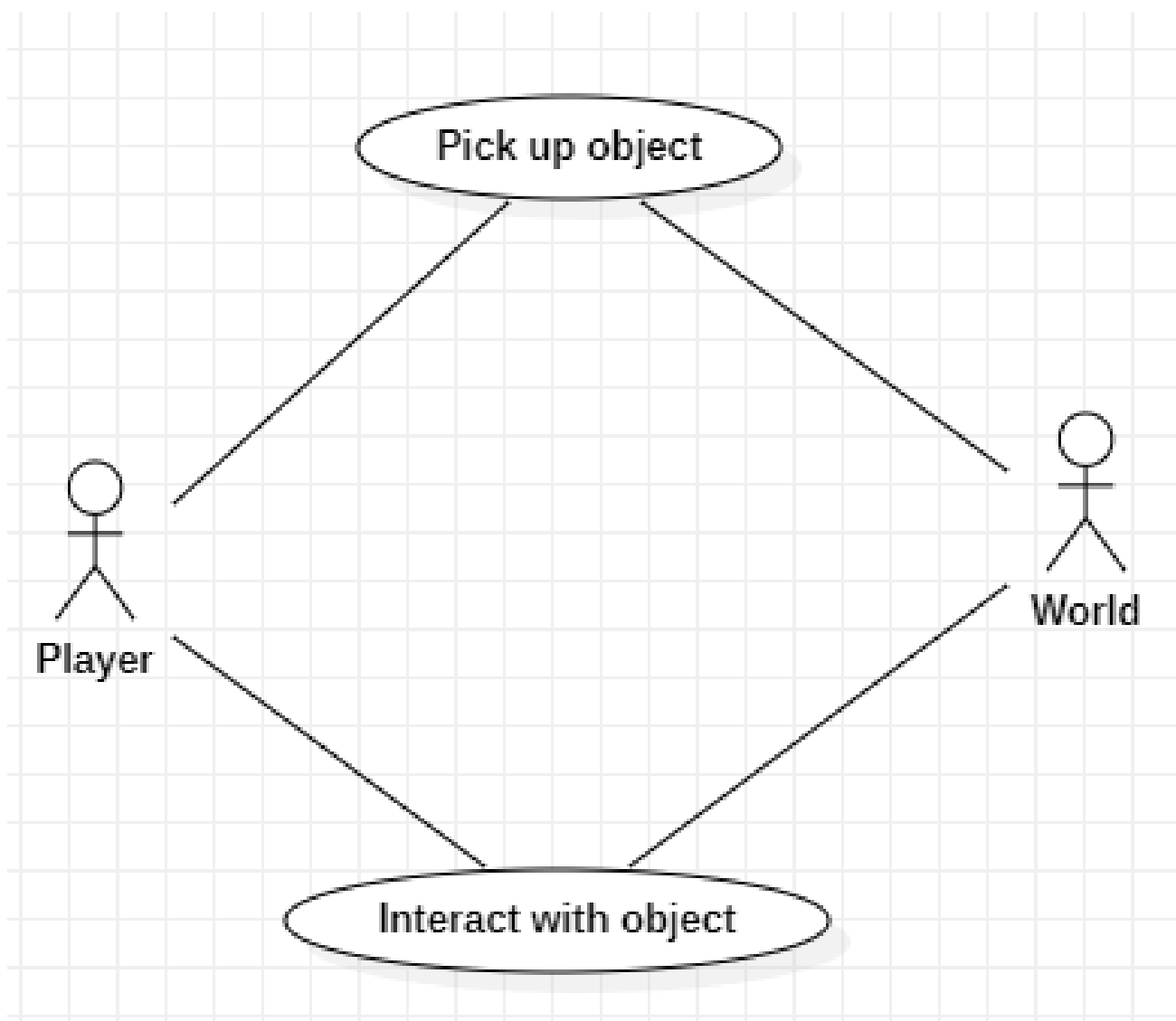


Рис 2.3 Діаграма прецедентів ігрового рівня

На ігровому рівні розміщені різні об'єкти, з якими може взаємодіяти гравець. Гравець може піднімати об'єкти, просто підійшовши до них на достатньо близьку відстань, після чого вони зникнуть з карти, то піднімуть характеристики героя, до таких об'єктів відносять патрони, аптечки, броню, ключі. За цю дію відповідає прецедент Pick Up. А також гравець може взаємодіяти з такими об'єктами, як двері, підійшовши до них та натиснувши відповідну клавішу, двері будуть відчинятися. І це другий прецедент.

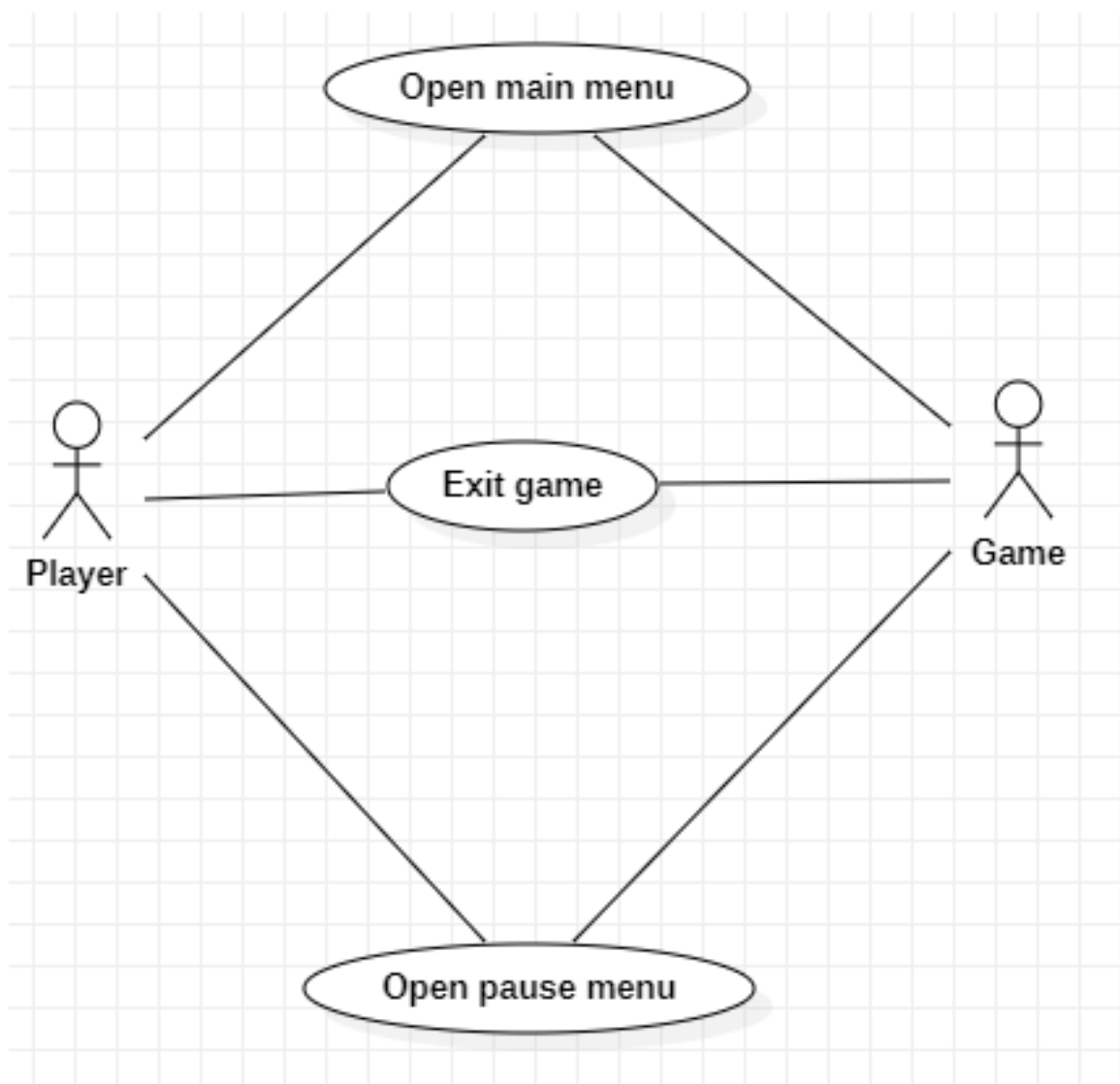


Рис 2.4 Діаграма прецедентів меню гри

Під час проходження рівнів гравець може натиснути паузу, за що відповідає прецедент Open pause menu. Пауза не тільки зупиняє гру, але й дозволяє гравцю повернутися в основне меню, якщо він хоче розпочати гру занову чи завантажити інший рівень. Таку можливість надає прецедент Open main menu. А також при потребі можна вийти з програмного продукту, зачинивши його. За це відповідає прецедент Exit game.

Загальна діаграма Use Case об'єднала в собі всі попереднє визначені варіанти використання. Це допоможе оцінити в цілому ті задачі, які потрібно виконати та визначити загальну складність проекту. Уникнути повторення варіантів використання.

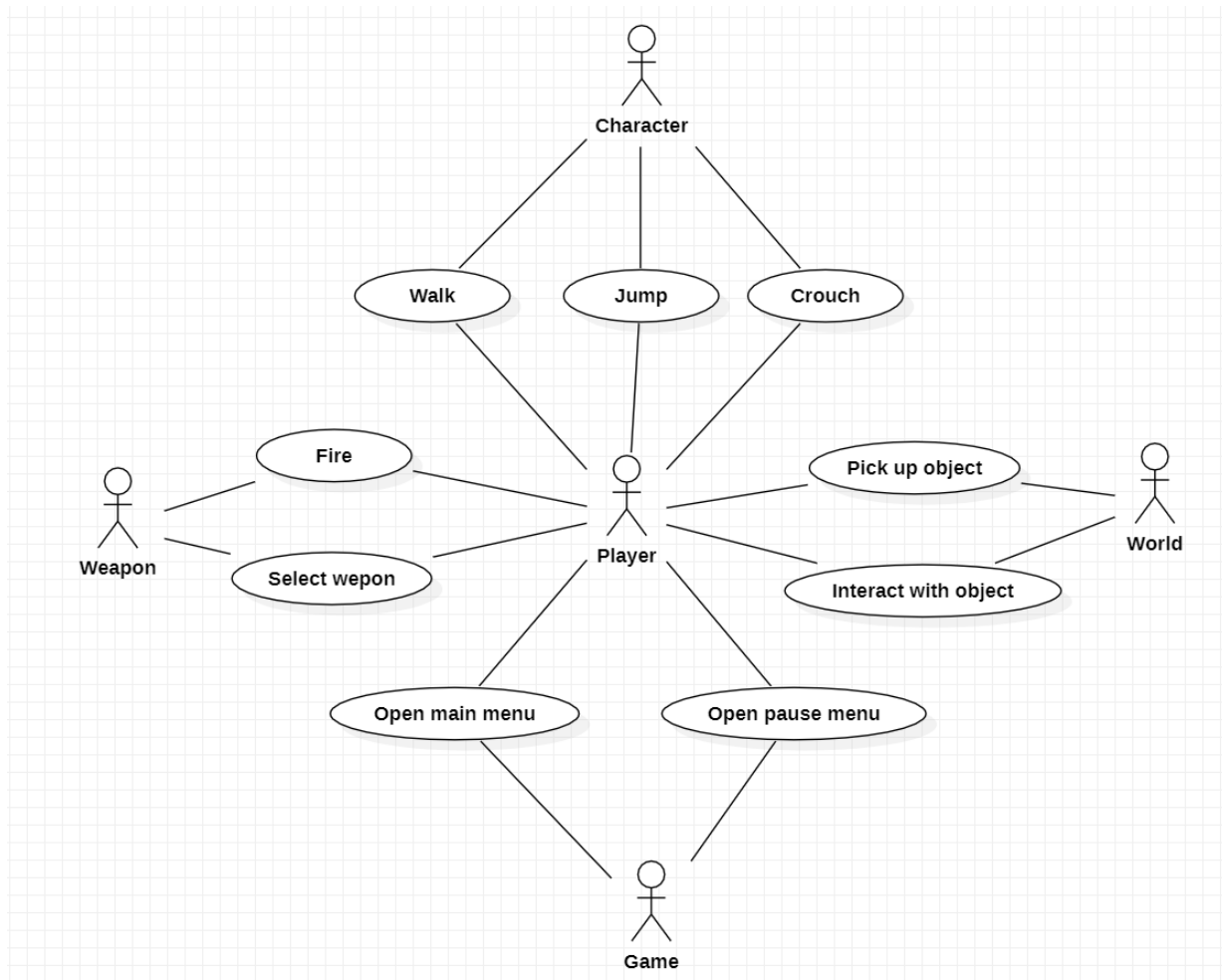


Рис 2.5 Загальна діаграма прецедентів

2.2 Проектування структури системи.

Розробка шутера включатиме в себе наявність різних ігрових механік, які потрібно грамотно спроектувати.

Проектування структури системи є процесом визначення організації та взаємозв'язків різних компонентів системи. Це важлива фаза в розробці програмного забезпечення, де визначається архітектура системи, ієрархія компонентів, способи взаємодії між ними.

Проектування структури системи допомагає планувати роботу над проектом, визначати і уточняти вимоги до гри, краще розуміти, які функції має виконувати гра та як вони пов'язані між собою, адже різних функцій у грі доволі багато та без чіткої організації можна легко заплутатись.

Шутери мають багато різних аспектів, таких як управління персонажем, штучний інтелект ворогів, обробка зіткнень, зброя, анімація тощо. Проектування структури системи допомагає організувати ці різноманітні функції в окремі компоненти, що полегшує розробку та підтримку коду.

Використання модульної структури дозволяє розділити функціональність на окремі модулі, які можуть бути розроблені та тестовані незалежно. Наприклад, можна мати окремий модуль для обробки ворогів, інший для гравця, третій для елементів оточення і так далі. Це спрощує розробку, тестування та розширення гри.

У шутерах часто потрібна взаємодія різних компонентів, таких як персонажі, вороги, зброя, оточення тощо. Проектування структури системи дозволяє визначити, як ці компоненти будуть взаємодіяти, які дані та події будуть передаватися між ними. Це робить код більш зрозумілим та легким для управління. Від якості цієї структури системи залежить ефективність розробки, супровід та майбутнє розширення гри.

Шутери можуть розростатися з часом, розробники можуть додавати нові рівні, персонажів, зброю та функціональність. Якщо структура системи добре спроектована, то додавання нових елементів стає значно простішим. Можна легко додати новий тип ворога, нову зброю або нову функціональність, не змінюючи весь існуючий код.

Для проектування будуть використані діаграми класів, які є потужним інструментом для візуалізації та проектування структури системи. Вона допомагає розробникам легше розуміти систему, виявляти проблеми та приймати рішення щодо архітектури проекту.

Діаграма класів допомагає ідентифікувати класи, їх атрибути, методи та зв'язки між ними. Це дозволяє визначити, які класи потрібні для реалізації функціональності гри та як вони пов'язані між собою.

Діаграма класів сприяє розумінню того, як класи взаємодіють між собою за допомогою асоціацій, агрегацій, композицій та інших типів зв'язків. Це

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

дозволяє вирішувати питання про обмін даними, спільне використання ресурсів та взаємодію компонентів гри.

Діаграма класів також дає змогу визначити, які компоненти системи можуть бути легко модифіковані або розширені без впливу на інші частини системи. Це сприяє гнучкості та повторному використанню коду.

Загалом, проектування структури системи є важливим етапом у розробці шутера, оскільки воно сприяє організації функціональності, полегшує розробку та підтримку коду, забезпечує модульність та розширюваність гри. Відповідна структура системи робить проект більш гнучким, зрозумілим та підготовленим для майбутнього розвитку.

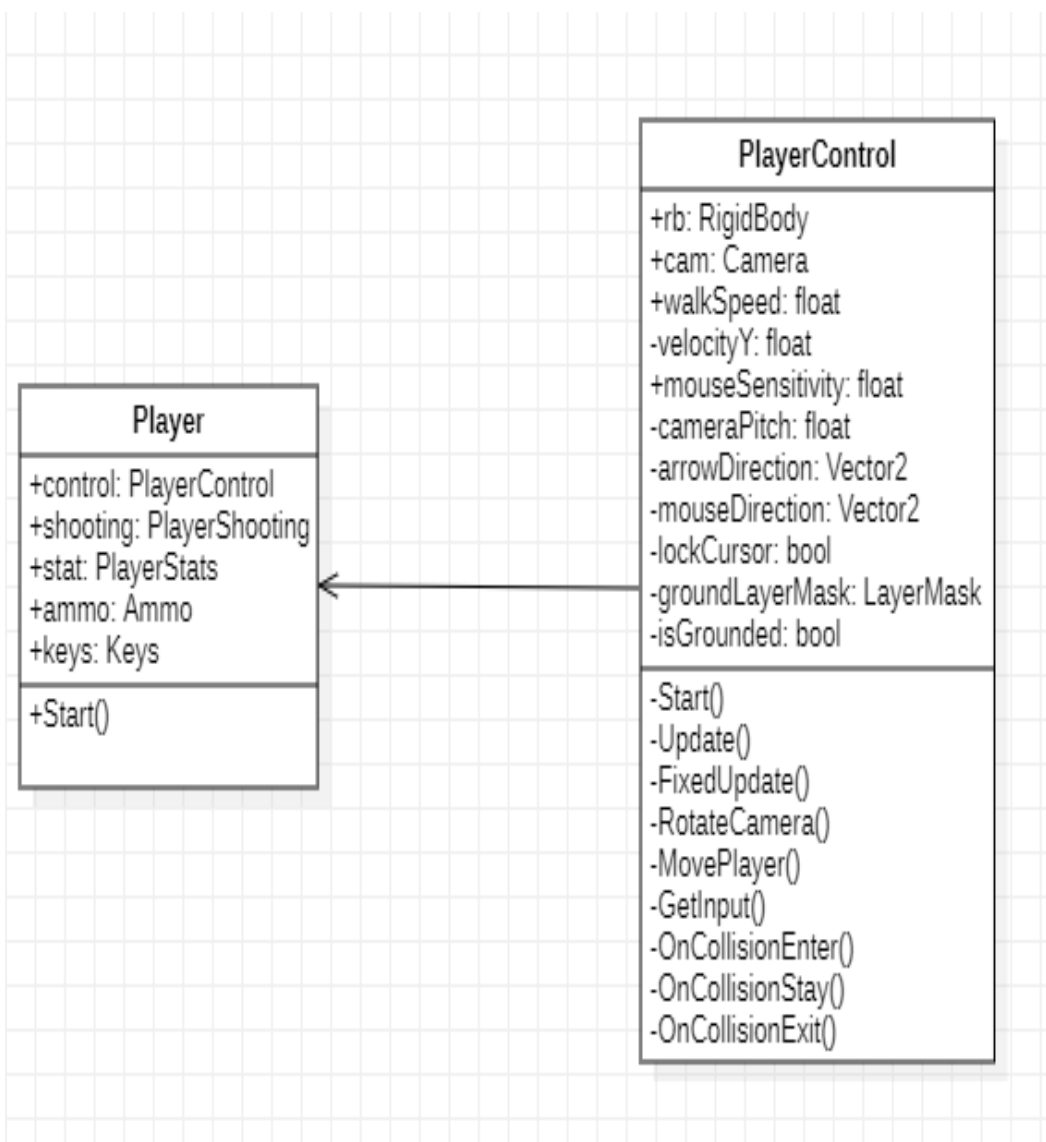


Рис 2.6 Діаграма класів керування персонажем

Клас `PlayerControl` реалізує керування гравцем, яке можна поділити на дві частини: перше – це керування пересуванням тіла персонажу в просторі на площині карти, тіло ігрового персонажа являє з себе пустий об’єкт, до якого прикріплені коллайдер, щоб відбувалися колізії з підлогою, стінами, противниками, та компонент `RigidBody`, який використовується для симуляції фізичного руху в просторі та саме його властивості будуть змінюватися через код; друге – це за допомогою рухів миші реалізовані повороти головною камерою, яка відображає вид з очей героя. До камери, як дочірній об’єкт, буде прикріплена зброя, як буде рухатися разом з камерою. Її буде завжди видно на екрані. Її позиція фіксована.

За фізику тіла персонажу відповідає, як вже було сказано вище, компонент `RigidBody`. Для того, щоб змінювати позицію персонажу в просторі використовуються властивість `Velocity`, яка вказує швидкість руху по кожній із осей тривимірного простору. Вектор руху на площині отримується методом `GetInput()`, який зчитує натиснення клавіш руху в змінну `arrowDirection`, після чого вони передаються як одиниці або нулі в показники осей `X` та `Z` властивості `Velocity` та множаться на задану швидкість пересування персонажу `WalkSpeed`. Таким чином реалізуються рух у вісьмох напрямках, по діагоналі швидкість така ж сама, як по вертикалі чи горизонталі, для цього використовується метод `Normalize()` для заданого вектору, який робить його довжину (*magnitude*) завжди рівний одиниці, що забезпечує однакову швидкість у всіх напрямках.

Стрибок реалізований в методі `Jump()`. Якщо гравець натиснув кнопку стрибка, а персонаж знаходиться на землі, що перевіряє булева змінна `isGrounded`, значення якої визначають два наступні вбудовані методи коллайдера `OnCollisionStay(Collision collision)`, коли гравець знаходиться на підлозі, та `OnCollisionExit(Collision collision)`, коли гравець втратив контакт з землею, а друга зміна, яка також перевіряється, - це та, чи персонаж не в положенні присідання, бо з такого положення не можна здійснити стрибок, якщо по цим двом булевим змінним можна здійснити стрибок, то до тіла героя буде задіяна сила по напрямку вгору, що реалізовано через метод `AddForce()` класу `RigidBody`.

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

Сила додається кожен кадр по вектору, що вказує напрямок вгору. Вона буде додаватися до тих пір, поки гравець не досягне граничної висоти, після чого швидкість стане нульовою, та гравець почне падати з лінійним прискоренням, що прописано в формулі. Ці обчислення проводяться вручну через особливості формули руху, де потрібно завжди вказувати швидкість по осі Y, інакше гравець при стрибку просто зависне в повітрі.

Метод Crouch() відповідає за те, щоб гравець присідав до тих пір, доки буде натиснена відповідна клавіша. Для того, щоб гравець міг присісти, перевіряється булева змінна isGrounded, тобто якщо гравець знаходиться у повітрі, наприклад, під час стрибку, то присідання не буде виконуватись. Щоб здійснити присідання, відбуваються наступні маніпуляції з коллайдером та камерою. В методі Crouch() позиція камери, яка реалізує вид від першої особи, опускається по осі Y, а також зменшується висота коллайдеру, який відповідає за зіткнення тіл, щоб персонаж міг проходити в низьких місцях, куди він не пройде в позиції стоячи. При відпусканні клавіші присідання положення камери та розміри коллайдеру повертаються до початкових, які є в персонажу під час ходьби.

Метод RotateCamera() відповідає за повороти камери, що відбуваються завдяки пересуванню мишки, та дозволяє герою дивитися в різні сторони по горизонталі та вертикалі. Швидкість поворотів камери визначаються параметром mouseSensitivity, яка множиться на вектор руху миші та той час між кадрами в грі. Персонаж може робити повний оберт навколо осі X, але в нього є обмеження по осі Y на 90 та -90 градусів, за що відповідає змінна cameraPitch, яка через метод Mathf.Clamp() встановлює обмеження для руху камери, щоб крайніми точками були погляд прямо в гору та прямо під ноги, але не повний оберт, що суперечить можливостям людини.

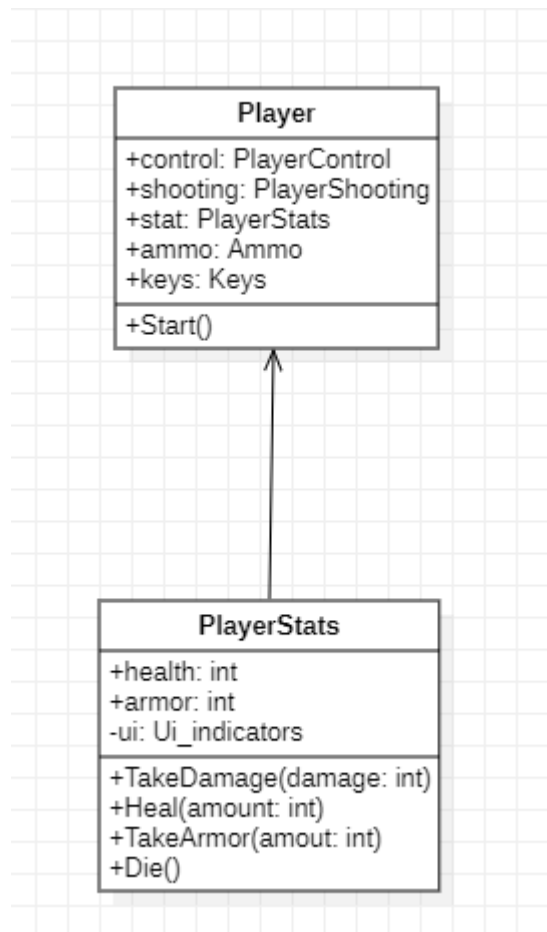


Рис 2.7 Діаграма класів показників персонажа

Клас `PlayerStat` відповідає за такі показники, як здоров'я та броня персонажу. Ці два показники змінюються під час боїв з суперниками, які наносять головному герою пошкодження своїми дальнобійними атаками з вогнепальної зброї та атаками на короткій відстані холодною зброєю. Також здоров'я буде відніматись, якщо власний снаряд ракетниці розірветься поруч з головним ігровим героєм. Показники бронні та здоров'я можуть не тільки зменшуватись від пошкоджень, що завдають суперники, але і збільшуватися під час того, як гравець знаходить на карті аптечки та броню, які він може підбирати.

В методі `TakeDamage(int damage)`, який викликається зовні під час влучання в героя ворожої кулі чи розриву поряд снаряду ракетниці, в зону ураження якої попадає гравець, реалізовано отримання пошкоджень вхідним аргументами, метод не знає, яка зброя була використана, він лише отримує цілочисельне значення, яке вказує на те, скільки гравець має отримати пошкоджень. Далі по формулі віднімаються здоров'я та броня персонажа, якщо

вона в нього пристуня. Броня поглинає три четверті пошкоджень, а залишок віднімається напрямку від здоров'я. Якщо показник здоров'я персонажу падає нижче одиниці, то він помирає.

Метод Heal (int amount) поповнює здоров'я персонажу, яке передається в якості аргументу, цей метод викликається зовні під час того, як гравець піднімає аптечки, які він знаходить на карті. Здоров'я не може збільшуватися нескінченно, в нього є гранична величина, що становить 100 одиниць. Якщо ігровий персонаж має повне здоров'я та піднімає аптечку, то нічого не відбувається, хоча аптечка використовується..

Метод TakeArmor(int amount) додає броню до персонажу. Це значення також, як і показник здоров'я, має граничне значення, вище якого не може бути кількість броне, але воно вдвічі вище, ніж показник здоров'я, та становить 200 одиниць. Якщо гравець втрачає всю броню, то нічого не відбувається, крім того, що всі пошкодження будуть поглинати одиниці здоров'я головного героя.

Також всі ці три методи, TakeDamage(int damage), Heal (int amount) TakeArmor(int amount), викликають зміни в ігровому інтерфейсі, де відображаються показники здоров'я та броні ігрового персонажу, щоб гравець міг слідкувати за тим, скільки в нього є запасів броні та здоров'я. Для цього в скрипті є посилання на клас ui_indicators, який відповідає за вивід на екран даних про ігрового персонажа, і в цей клас в кінці кожного з вищеназваних методів передаються оновлені значення відповідних показників.

Метод Die() викликається з методу TakeDamage(int damage). Це відбувається, як вже було сказано, в тому випадку, коли персонаж в процесі гри, отримуючи пошкодження від суперників, втратив повністю всю броню та показник його здоров'я опустився нижче одиниці. В гравця нема декількох життів, щоб ігровий персонаж міг відродитися на місці загибелі та була можливість продовжити гру з цієї самої точки. Замість цього програма повертає гравця в головне меню, де він може розпочати гру з самого початку або обрати один з відкритих рівнів.

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

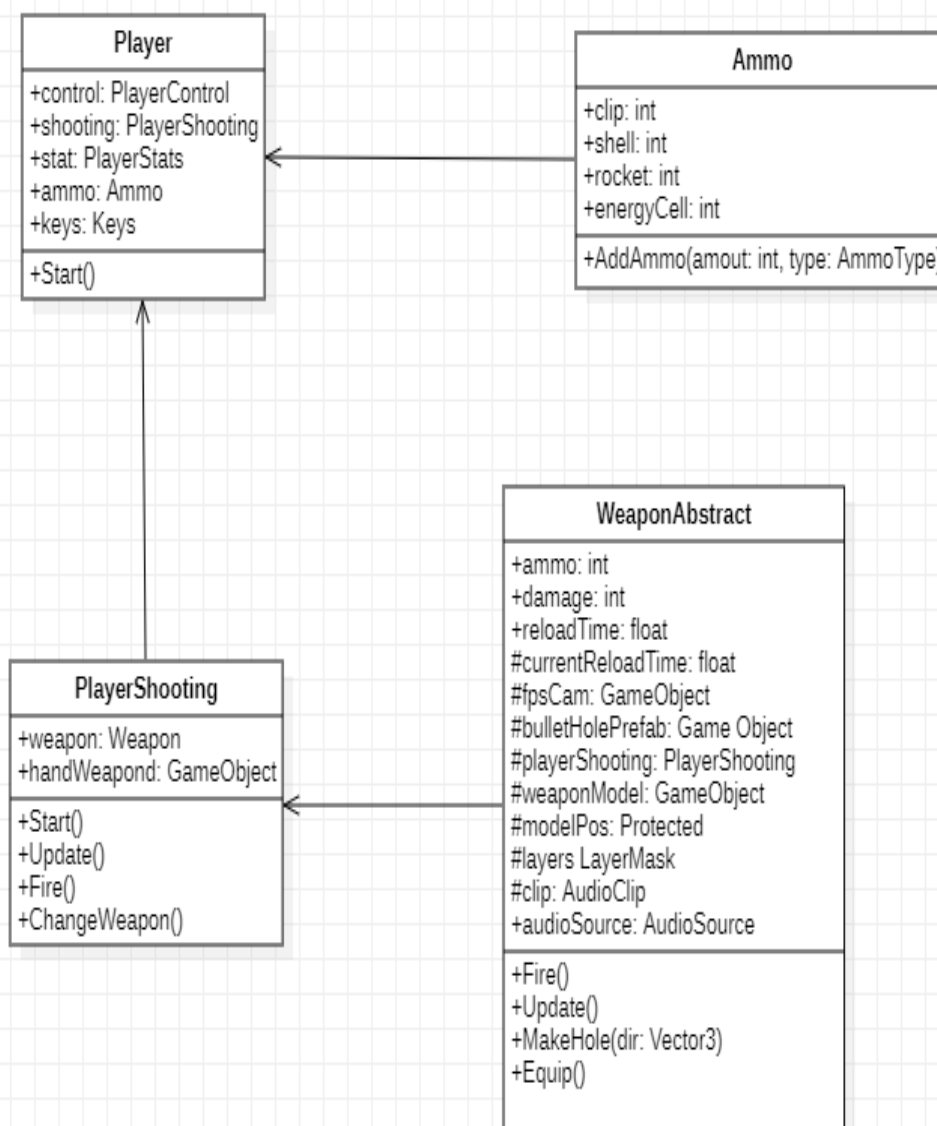


Рис 2.8 Діаграма класів стрільби персонажа

Клас `PlayerShooting` відповідає за вибір зброї та ведення вогню.

Через метод `ChangeWeapon()` реалізується зміна зброї. В залежності від того, яку кнопку натиснув гравець, базовому класові `weapon`значається один з похідних класів, який відповідає за певний тип зброї: пістолет, дробовик, ракетниця і т.д. Для цієї зброї завантажується кількість боєприпасів, інформація про які відповідно відображаються в інтерфейсі гри, до якого доступ отримується через змінну `ui_indicators`, яка посилається на відповідний клас.

Через метод `Fire()` відбувається стрільба по натисненню кнопки вогню, що зчитується в методі `Update()`. Стрільба є двох типів: поодинокі, коли постріл

відбувається щоразу при натисканні кнопки миші, та чергою, коли постріли йдуть один за одним, доки гравець утримує натиснутою кнопку миші, що відповідає за стрільбу. Звісно, тут враховується швидкість стрільби зброї та наявність боєприпасів в магазині. Після кожного пострілу боєприпаси віднімаються, що відповідно відображається на індикаторах ігрового інтерфейсі.

Абстрактний клас `Weapon` відповідає за зброю, якої в грі п'ять типів: пістолет, дробовик, кулемет, ракетниця та плазмова гвинтівка. Всі ці типи зброї будуть походити від базового. Клас `Weapon` містить цілий ряд змінних, що характеризують зброю: `damage` вказує на те, скільки зброя віднімає здоров'я у противників при влучанні; `reloadTime` - як швидко перезаряджається, а `currentReloadTime` потрібен для того, щоб підраховувати час, що залишився до перезарядки; `ammo` відповідає за те, скільки патронів в магазині зброї, кількість боєприпасів зчитуються з класу `PlayerShooting` при екіпіруванні зброї. А також клас `Weapon` містить моделі зброї, що зберігаються в папці ресурсів, як `prefabs`. Цю модель буде видно в руках головного героя, а позиція зброї на екрані, яка в кожній моделі зброї своя, визначається координатами `modelPos`, в наявності також є також змінна звуку пострілів `AudioClip` та посилання на компонент `AudioSource`, який відповідає за програвання звуків пострілу.

Віртуальний метод `Fire()` описує логіку пострілу, вона є двох типів. Через посилання променів, що викликається через метод `Physics.Raycast()`, де вказується початок променя, його напрямок та довжина, а також шари (`Layer`), з якими буде врахована фізика взаємодії. Таким чином, якщо кінцева точка польоту уявної кулі, так як постріл при такому методі відбувається миттєво, являється ворогом, то в нього віднімається кількість здоров'я, яка вказана в змінній `damage`. Другий тип пострілів - це створення об'єктів кулі, за рухом яких можна спостерігати. Такий снаряд має свою швидкість польоту та може не влучити в суперника, якщо він встигне ухилитися від кулі. Такі снаряди зберігаються, як `prefabs`, шлях для яких прописується в конструкторі зброї. За об'єкт кулі відповідає поле `projectile`. Під час пострілу куля створюється в місці, що приблизно відповідає виходу зі ствола зброї, за цю позицію відповідає пустий

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

дочірній об'єкт зброї. Після чого рух кулі та взаємодія її з об'єктами оточення реалізуються відповідним скриптом, який прив'язаний до кулі.

Віртуальний метод Equip() відображає поточну модель зброї в руках персонажу. Кожна модель зберігається в полі weaponModel, значення якої прописується в конструкторі класу. Перед тим як завантажити зброю, потрібно перевірити чи існує стара, якщо вона є, то її необхідно знищити. Зброя є дочірнім об'єктом головної камери, що дозволяє їй рухатися синхронно з камерою, що відповідає за вид від першої особи.

Метод MakeHole() залишає сліди пострілів на об'єктах оточення, наприклад, на стінах та підлозі. Через метод Physics.Raycast() посилається промінь, який визначає точку влучання. Шар, на якому розташовані вороги, не враховується, тому на них слідів пострілів не буде видно. А за слід пострілу, який буде залишатися на стінах, відповідає ігровий об'єкт, що вказаний в полі bulletHolePrefab. Цей ігровий об'єкт сліду пострілу створюється в точці влучання та йому задається поворот, яка відповідають нормалі того об'єкту, в який було влучено.

Для зберігання кількості боєприпасів існує окремий клас Ammo, оскільки зброя в класі PlayerShooting при змінні завжди створюється в якості нового екземпляру, тому вона не може зберігати кількість зарядів, так як ця інформація буде втрачена, як тільки гравець вибере інший тип зброї, тому інформація про заряди зберігається в окремому класі та передається зброї при кожному її створенні.

Клас Ammo зберігає кількість патронів для кожного типу зброї. Це публічні поля цілочислового типу. Також додатково є перелічуваний тип даних AmmoType, де вказано кожен тип боєприпасів. Клас Ammo має декілька методів.

Метод AddAmmo(int amount, AmmoPowerUp.AmmoType type) добавляє вказану кількість патронів певного типу зброї, що передаються аргументами методу. При цьому кількість патронів для кожного типу зброї не може перевищувати значення 999. Додавання патронів викликається зовні, коли гравець на карті знаходить боєприпаси.

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

Метод `int GetAmmo(Weapon weapon)` повертає кількість патронів, залежно від того, який тип зброї був вказаний. В даному випадку можна передавати ту зброю, яка є в руках гравця, щоб знати скільки в неї патронів та, наприклад, відображати це на екрані.

Метод `RemoveAmmo(Weapon weapon)` віднімає кількість боєприпасів. Він викликається зовні при кожному пострілі зброї. В якості аргументу використовується клас зброї, а не тип патронів, щоб можна було зручно передавати ту зброю, з якої в даний момент ведеться вогонь.

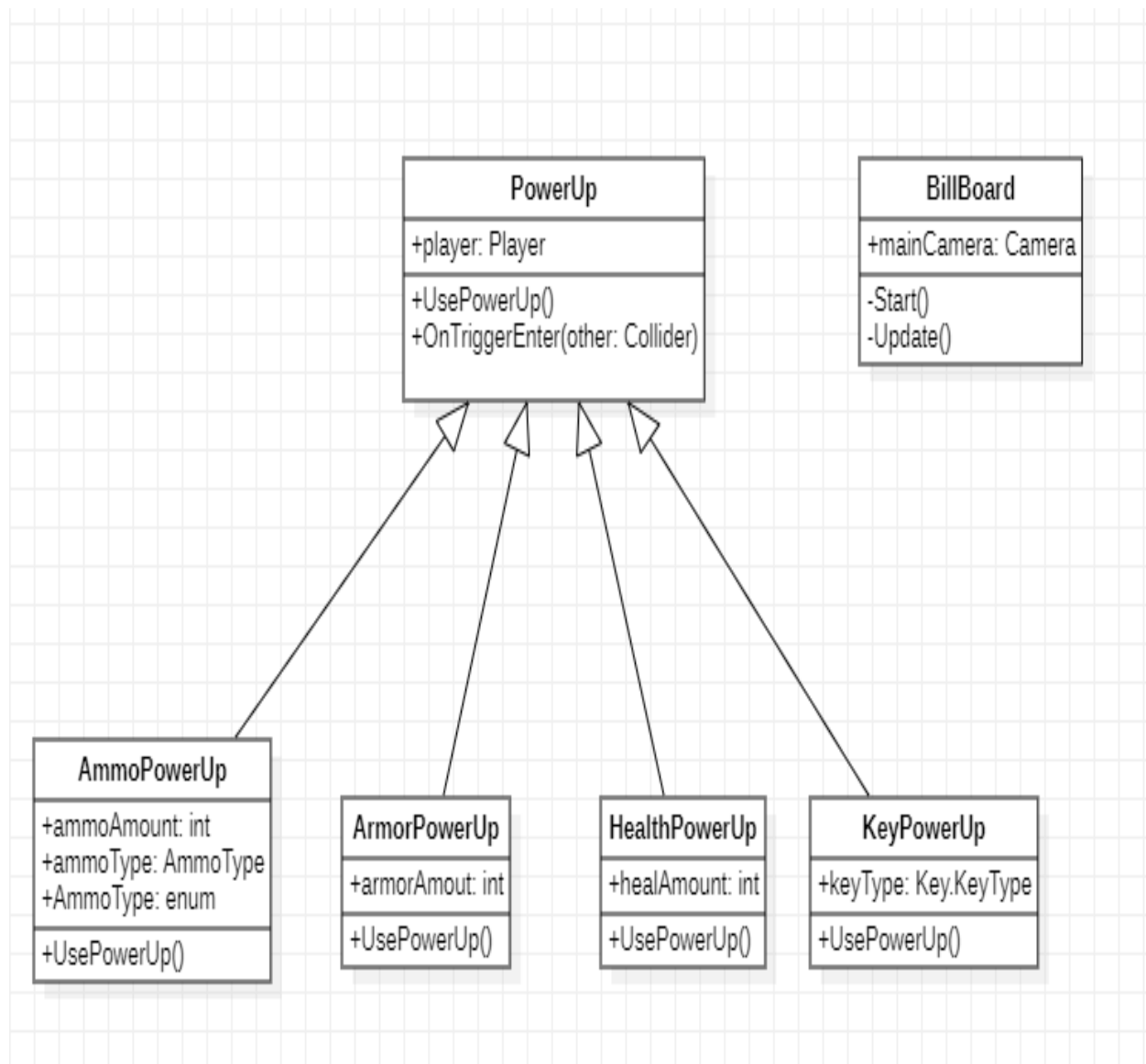


Рис 2.8 Діаграма класів предметів, що можна піднімати

Абстрактний клас **PowerUp** відповідає за ті корисні предмети, які гравець може знайти та підібрати на карті. До предметів буде прив'язаний `trigger collider`,

який не буде враховуватись в фізичній моделі світу, як перешкода, тобто гравець буде проходити скрізь нього, але буде викликатися метод `OnTriggerEnter(Collider other)`, де буде перевірятися, чи саме з гравцем відбувається взаємодія.

При взаємодії з гравцем буде викликатися абстрактний метод `UsePowerUp()`, який буде перевизначатися в кожному похідному класі. Які в свою чергу будуть передавати потрібну інформацію класу `player`, який зберігає всю необхідну інформацію про гравця, а саме кількість його здоров'я та броні, кількість боєприпасів до кожного типу зброї.

Клас `AmmoPowerUp` відповідає за підбір боєприсів. Один скрипт використовується зразу для всіх можливих видів боєприсів. Тип патронів визначається в змінний типу `AmmoType`, що належить класу `Ammo`. Кожен виклик метода `UsePowerUp()` буде додавати зазначену кількість патронів в полі `ammoAmount` до відповідної зброї в арсеналі гравця. Кількість патронів, що дається, буде вказана в `prefab` індивідуально для кожного типу зброї.

Клас `HealthPowerUp` пов'язаний зі здоров'ям персонажу. Коли гравець піднімає на карті предмет, до якого прив'язаний цей скрипт, то кількість його здоров'я збільшується на вказану величину, що зазначена в полі `healAmount`. Метод `UsePowerUp()` звертається до класу `player`, де зберігається вся інформація про гравця, та передає туди кількість одиниць, на яку потрібно поповнити здоров'я героя.

Клас `ArmorPowerUp` відповідає за броню героя та діє точно таким же чином, як і клас `HealthPowerUp`. Коли гравець знаходить предмет на карті, до якого прив'язаний скрипт `ArmorPowerUp`, то кількість його броні збільшується на величину вказану в полі `armorAmount`, що передається класові `Player`.

Клас `KeyPowerUp` відповідає за ключі, які гравець може знаходити на карті під час проходження рівнів. Ці ключі необхідні для того, щоб відкривати певні двері, які збігаються з типом ключа. Для ключа існує окремий клас `Key`, який зберігає інформацію про тип ключа, який вказується в конструкторі класу. А також є клас `Keys`, який відповідає за всі ключі, що є в наявності у головного

					КН.КР.Б – 122 – 23 – ПЗ	Арк.
						43
Змн.	Арк.	№ докум.	Підпис	Дата		

героя, вони зберігаються в списку `List<Key> keys`. Також в цьому класі передбачені два публічні методи, за допомогою яких можна добавляти та видаляти ключі зі списку.

Клас `Billboard` буде застосовуватись до кожного об'єкту типу `PowerUp`. Оскільки вони на карті будуть відображатися не як тривимірні тіла, а як площини, то потрібно щоб вони були завжди повернені до гравця своєю лицевою стороною, інакше не можна буде розгледіти текстуру, що вказує на те, до якого типу належить саме цей `PowerUp`. В класі є посилання на головну камеру, що відповідає за вид з очей головного героя, та відносно неї повертається текстура.

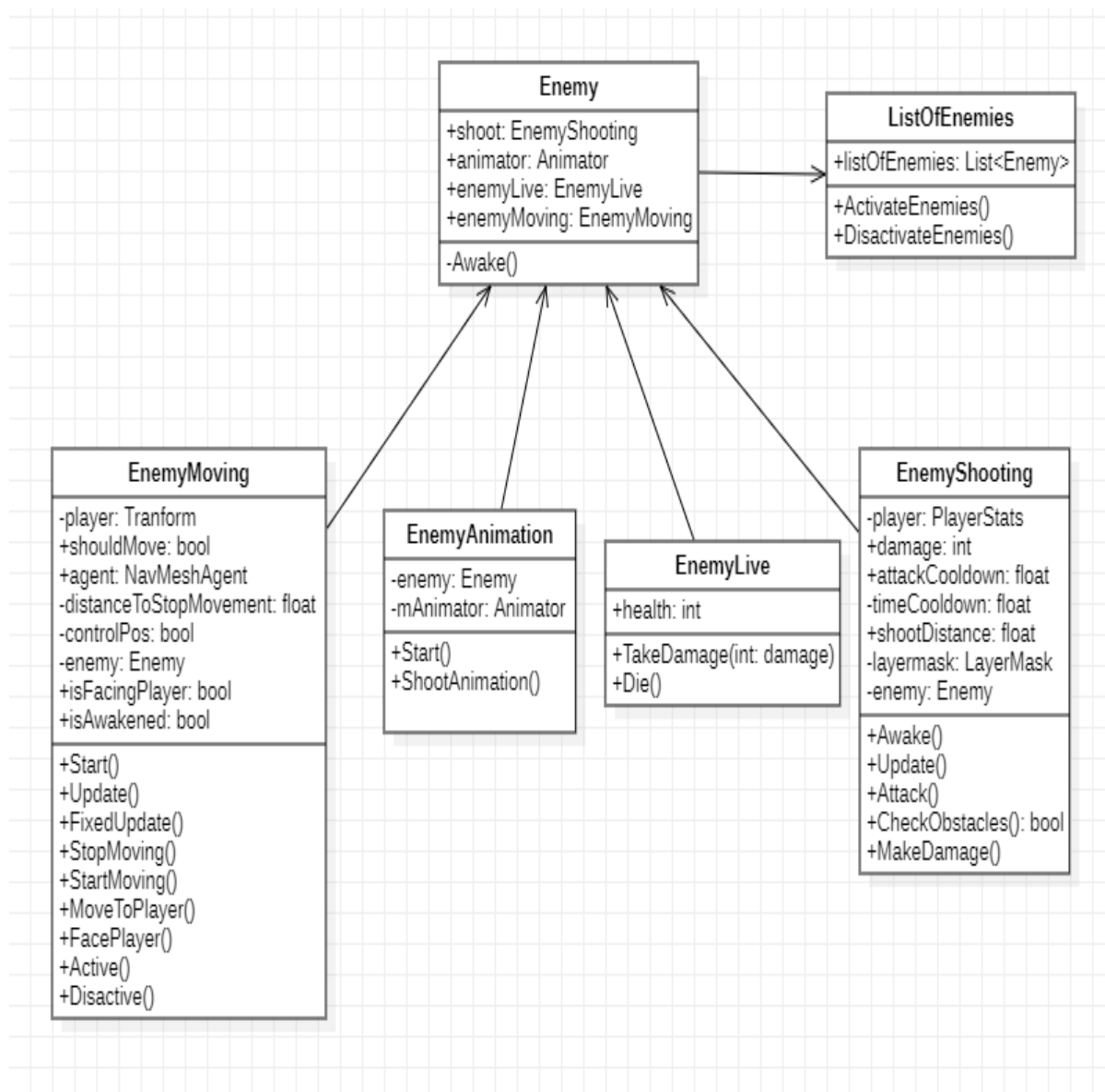


Рис 2.9 Діаграма класів суперника

Клас Enemy зберігає в собі всі класи, які описують поведінку суперника, а саме ті, що відповідають за переміщення ворога по карті, його стрільбу по ігровому персонажу, управляють анімаціями суперника та містять інформацію про кількість його здоров'я. Також існує клас ListOfEnemies, який використовується для зберігання груп суперників, які потрібно буде активувати чи дезактивувати під час пересування по зонам, наприклад, коли гравець відкриває двері.

Клас EnemyAttacking реалізує логіку атак суперника. В грі у ворогів є три типи атаки: перша - суперники стріляють здалеку та попадання по головному герою відбувається миттєво, така атака відбувається через посилення променю; другий тип атаки – це коли суперник запускає в сторону гравця снаряд, який має свою швидкість та у гравця є деякий час, щоб ухилитися від снаряду, який не міняє траєкторію свого польоту; і останній, третій тип атаки, - це атака зброєю ближнього бою, яка відбувається тоді, коли суперник підходить на достатньо коротку дистанцію до головного героя, якщо гравець встигне відбігти, то суперник може промахнутися.

Поле damage містить ту величину пошкоджень, яку отримує герой при влучанні по ньому ворога. Поле attackCooldown відповідає за загальний час між атаками суперника, а currentTimeCooldown відраховує поточний час, скільки залишилось до наступної атаки, щоб всі суперники не стріляли одночасно, час першого пострілу визначається випадковою генерацією числа, attackDistance – це величина, що вказує на мінімальну дистанцію між ворогом та ігровим персонажем, з якої суперник може проводити свої атаки. isProjectile – це булева змінна, що відповідає за тип атаки, яку буде наносити суперник, чи то через технологію посилення променів, чи через створення ігрового об'єкту кулі в заданій позиції, яка зберігається в полі muzzle.

Приватний метод void Attack() веде підрахунок часу між атаками і якщо він досягнув необхідної позначки, то перевіряється чи дистанція між гравцем та ворогом задовільняє умовам атаки, а також чи нема між ними перешкод, якщо всі умови виконано, то відбувається одна з трьох атак. Атака ближнього бою та

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

запуск снаряду реалізуються через співпрограми (Coroutine), що синхронізувати анімацію з атакою.

Метод `bool CheckObstacles()` перевіряє, чи є перешкоди польоту кулі між суперником, який збирається стріляти, та гравцем, в якого цілять. Для цього використовується посилення променю, якщо він влучає в гравця, то перешкод немає, та метод повертає відповідне значення.

Клас `EnemyMoving` відповідає за пересування противника. Логіка пошуку шляху відбувається завдяки вбудованому компоненту `NavMesh`, який створює на рівні карту, а кожному ворогу прикріплюється компонент `NavMeshAgent`, через методи якого віддаються вказівки, куди супернику рухатись або чи потрібно йому зупнитися.

Поле `isAwakened` відповідає за те, чи суперник активований та буде виконуватися логіка прописана в даному скрипті, в іншому випадку ворог буде стояти бездіяльно. Для зміни цього поля існує два методи `Activate()`, який активує суперника, та `Disactivate()`, який дезактивує суперника. Вони будуть викликатися зовні з класу `ListOfEnemies` в залежності від того, як пересувається гравець по карті.

Поле `shouldMove` вказує на те, чи потрібно супернику рухатись в сторону головного героя, що реалізовано в методі `MoveToPlayer()`, де в якості цілі для `NavMeshAgent` вказується поточне положення гравця, або ж суперник може стояти, досягнув граничної відстані між ним та героєм. Для стояння також існує свій метод `Standing()`. Поле `isFacingPlayer` вказує на те, чи супернику потрібно дивитися в сторону героя, реалізація чого прописана в методі `FacePlayer()`.

Клас `EnemyAnimation` відповідає за анімацію та її синхронізацію з пострілами, які реалізовані в класі `EnemyShooting`. Він містить посилення на клас `Enemy`, який зберігає всю інформацію про суперника, а також посилення на компонент `Animator`. Публичний метод `Shoot(float time)` приймає значення часу, яке потрібне для затримки між анімаціями. Ці дані передаються в співпрограму, яка виконується паралельно.

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

2.3 Проектування алгоритмів роботи системи

Головний алгоритм в кваліфікаційній роботі – це штучний інтелект суперників. Це важливий елемент гри, який прописує розумну поведінку ворогам, їх взаємодію з гравцем та навколишнім середовищем. Цей алгоритм буде визначати різні аспекти поведінки суперників: рух, атаку, отримання ушкоджень.

Для графічного зображення буде використана блок-схема, де кожен блок представляє окрему дію або умову для прийняття рішення. Значення блок-схеми полягає у легкості читання та аналізу. Блок-схема дозволяє розробнику візуалізувати послідовність дій та рішень, які здійснює штучний інтелект суперників.

В блок-схемі поведінки суперника буде дві точки входу: ініціалізація та утримання ушкоджень.

Ініціалізація, тобто поява на карті суперника, запускає відразу перевірку, чи суперник активізований, якщо ні, то ворог буде бездіяльно стояти на місці до тих пір, поки його не активізують, що в грі буде відбуватися, наприклад, через відкриття дверей.

Після активізації суперник буде атакувати гравця через певний проміжок часу при виконанні потрібних умов, для суперників ближнього бою – це знаходження героя в зоні ураження, а для суперників з дальнобійною атакою – це відсутність перешкод на шляху польоту кулі, які можуть їй завадити влучити в ціль.

Також відбувається переслідування героя, суперники рухаються в його сторону до тих пір, доки не скоротять дистанцію до вказаного мінімального значення, після чого їх рух зупиняться до тих пір, поки дистанція знову не збільшиться.

При отриманні пошкоджень, віднімається певна кількість здоров'я у суперника, якщо в результаті в нього здоров'я стає нуль або нижче, то цей суперник помирає та відключаються його скрипти.

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

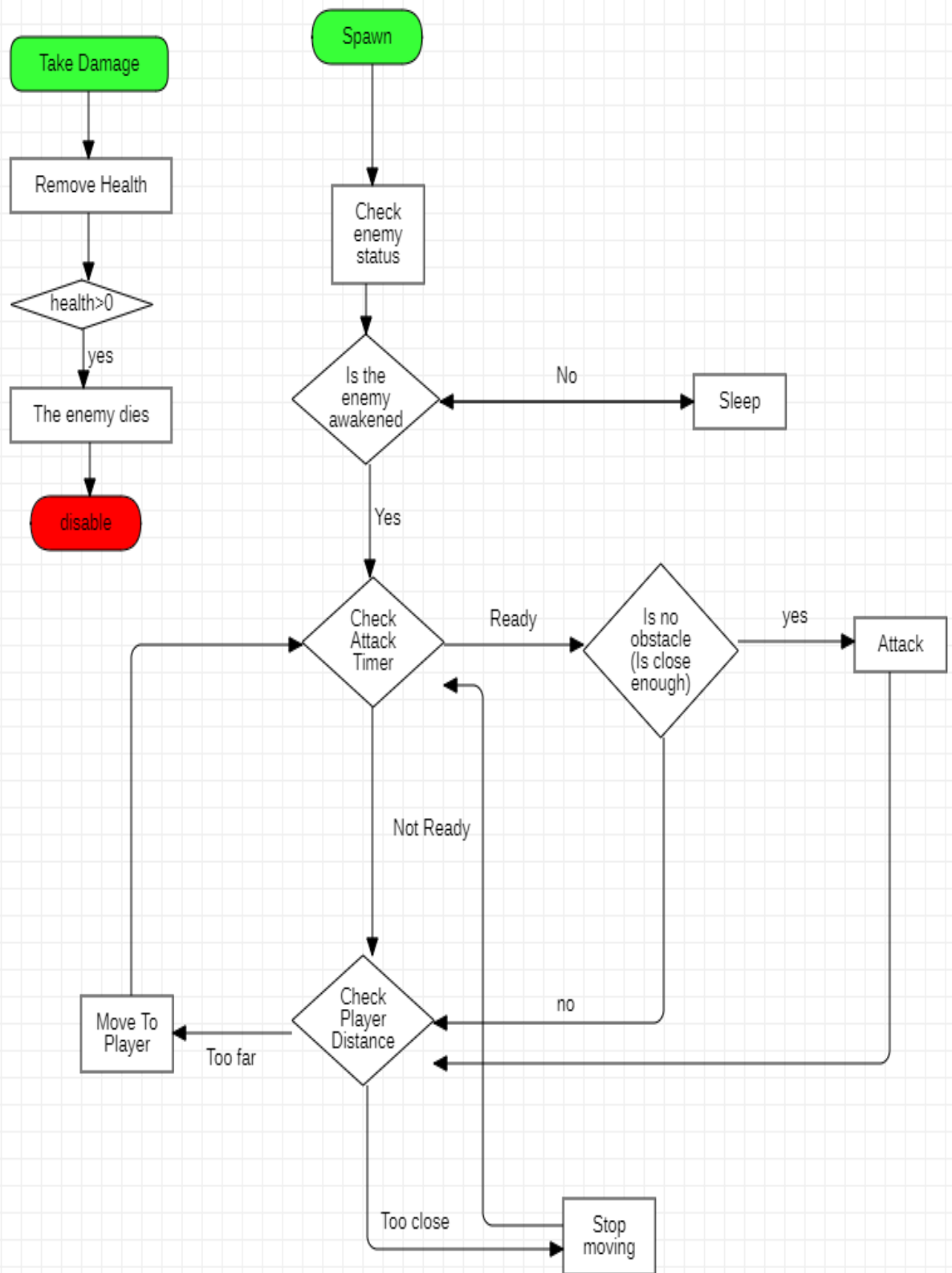


Рис 2.10 Діаграма діяльності алгоритму штучного інтелекту

2.4 Реалізація програмного продукту

При розробці гри активно використовувалась офіційна документація ігрового рушія Unity, що дозволяло в короткий час уточнювати незначні питання, а також вирішувати більші проблеми, які виникали на шляху створення програмного продукту. Питання стосувалися використання стандартних класів та структур рушія Unity, а також роботи з стандартними компонентами рушія.

Наприклад, у великій нагоді стало використання компонентку пошуку шляху. Навігації по рівню відбувалася завдяки NavMesh, який створює на рівні карту з різними зонами, головні з них – це ті по яким ворог може ходити та ті, де пересування заборонено, но також можуть бути зони з різною ціною пересування, що буде враховуватись при обчисленні оптимальної дистанції. Для того, щоб ворог міг пересуватися по створенній карті, йому потрібно присвоїти компонент NavMeshAgent та налаштувати його, де вказати його розміри та швидкість.

Також в розробці гри допомогли інші джерела інформації, де можна було розглянути приклади створення шутерів від першої особи, звідки були запозичені такі механіки гри, як управління мишкою або пересування по карті, але деякі функціональності були розроблені самостійно, наприклад, фізика стрибків.

Крім рушія Unity та мови програмування C#, для створення гри використовувались і інші програмні продукти та інтернет ресурси. З безкоштовних бібліотек було завантажено звукове супроводження для гри: звуки пострілів, відкриття дверей, фонова музика.

В редакторі MagicaVoxel були створені примітивні моделі ворогів та зброї, як в подальшому експортувалися до редактора тривимірної графіки Blender, де моделі підлягли обробці, яка їх оптимізувала для гри та для того, щоб до них приєднати скелет. Завдяки скелету, який був прив'язаний до моделі, та налаштуванню карти вагів, які відповідає за деформацію об'єкта, створювалася анімація ворогів: ходьби та стрільби.

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

Також в редакторі тривимірної графіки Blender були створені карти усіх трьох рівнів. Для цього використовувалися текстури намальовані в графічному редакторі Aseprite. Допомігав зручно створювати рівні add-on Sprityla. Джерела світла розташовувались на рівні вже у рушії Unity, що набагато зручніше.

Висновки до розділу 2

В даному розділі були наведені варіанти використання системи, які базувалися на основі поставленої задачі. Також були наведені діаграми класів та описано, яку вони виконують роль у програмному продукті. Далі був зроблений короткий опис діяльності алгоритму штучного інтелекту. І в завершення розділу розповідалося про реалізацію програмного продукту, які навички довелось задіяти, до яких джерел інформації звернутись.

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

РОЗДІЛ 3. Інтерфейс та порядок роботи з грою

3.1. Порядок встановлення та налаштування системи

Програмний продукт не потребує інсталяції. Збірка гри йде в архіві, який після завантаження на комп'ютер потрібно лише розархівувати та запустити «game.exe». Для нормальної роботи гри апаратне забезпечення комп'ютера має відповідати наступним системним вимогам:

- OS: Windows 7 (SP1+) 64-bit version,
- Процесор: Intel(R) Core(TM) i5-7400 CPU @ 3.0GHz (4 CPUs)
- Відеокарта: 1 GB Video RAM,
- Оперативна пам'ять: 4 GB of RAM,
- Вільне місце 100 MB
- DirectX: 11

Запустивши програмний продукт, гравець опиняється в головному меню, де можна розпочати нову гру з першого рівня, або, обравши інший пункт, продовжити гру з останнього рівня, на якому гравець загинув. Цей рівень буде не відновлений з точки загибелі, а буде розпочинатися з самого початку.



Рис 3.1 Вид головного меню

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

При виборі рівнів зникають пункти основного меню, але є можливість до них повернутися.



Рис 3.2 Вид вибору рівнів

Для керування ігровим персонажем використовується стандартний набір клавіш для шутеру, який добре знайомий всім гравцям, та не потребує того, щоб проводити навчання нового користувача, де би розповідалося про те, як правильно керувати ігровим персонажем, рухатися по карті, стріляти по ворогам, обирати зброю, виходити в меню.

Таблиця 3.1

Керування

Дія	Клавіша
1.	2.
Рухатись вперед	W або стрілочка вгору
Рухатися назад	W або стрілочка вниз
Рухатися вправо	W або стрілочка вправо
Рухатися вліво	W або стрілочка вліво
Стрибнути	Пробіл
Присісти	Left Ctrl
Стріляти	Ліва кнопка миші
Відкрити двері	E
Обрати пістолет	1
Обрати дробовик	2
Обрати автомат	3

1.	2.
Обрати базуку	4
Обрати плазмову гвинтівку	5
Вийти в меню паузи	Esc

На екрані в верхньому лівому кутку зображена інформація про головні характеристики ігрового персонажу, а саме кількість його здоров'я, яка позначена символом червоного хрестика, кількість його броні, яка позначена символом зеленого щита, та вказана кількість боєприпасів до поточної зброї, для кожної зброї є свій унікальний символ патронів.

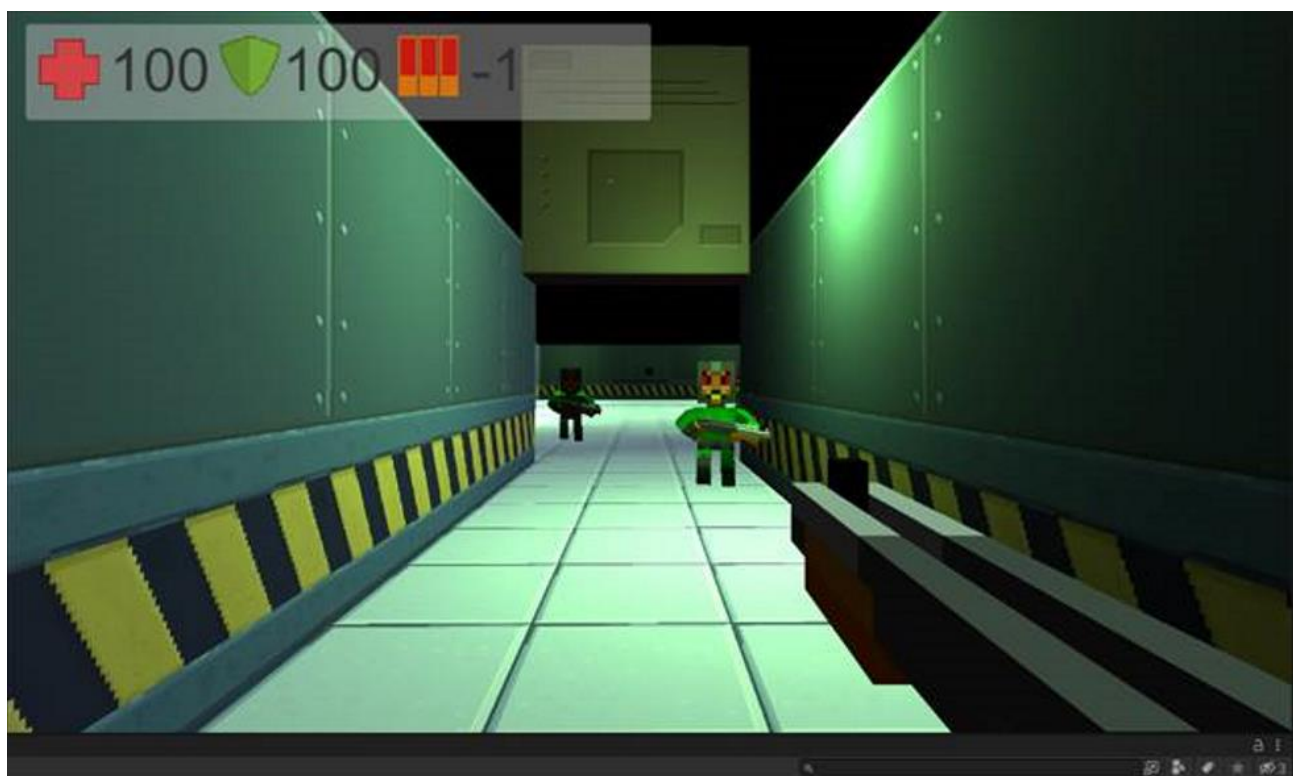


Рис 3.3 Кадр гри з ігровим інтерфейсом

Для того, щоб потрапити в меню паузи, гравцю потрібно натиснути клавішу Escape. Ігровий час зупиниться, вороги перестануть рухатися, гравець також не зможе ні рухатись, ні стріляти, а буде можна тільки обрати один з трьох пунктів меню. Чи продовжити гру, що також буде відбуватися по повторному натисненню клавіши Escape, чи вийти в головне меню, де можна буде розпочати гру з початку або вибрати один з доступних рівнів, чи повністю вийти з гри, закривши програмний продукт та повернувшись до операційної системи.

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

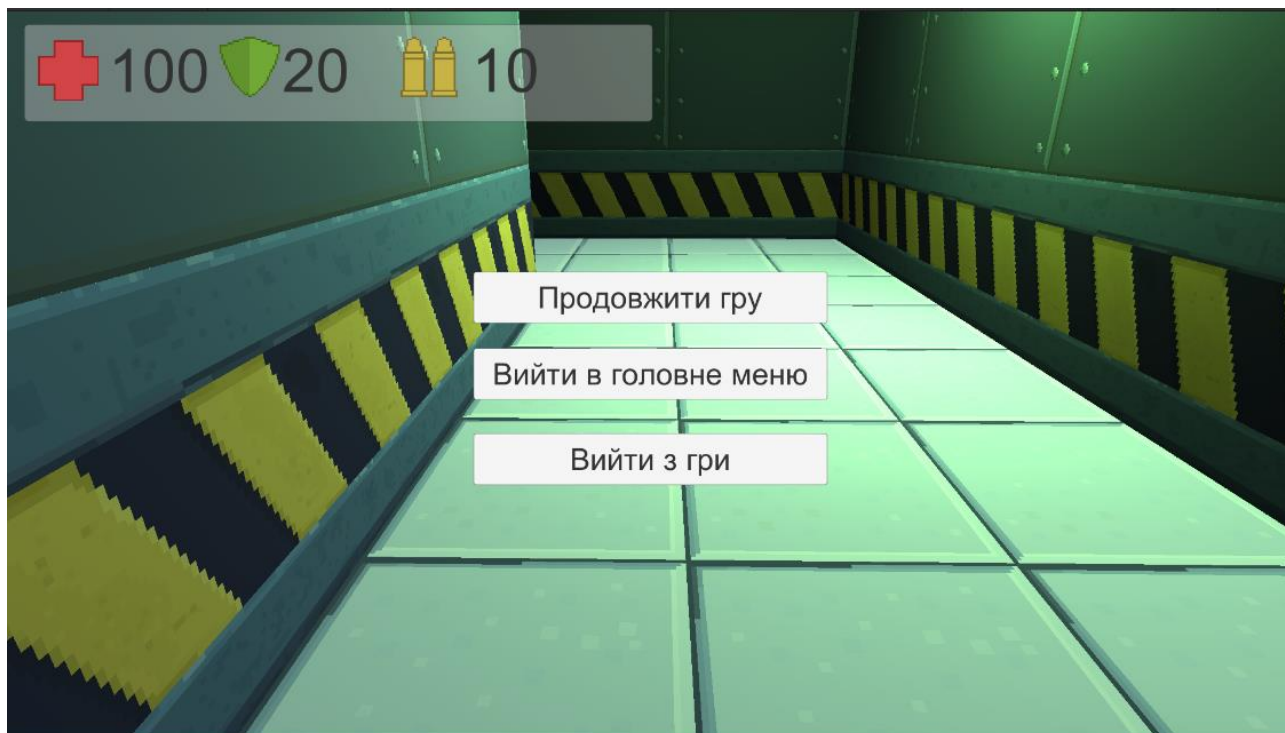


Рис 3.4 Меню паузи

3.2 Тестування роботи програмного продукту

Пошук та усунення багів відбувалися вже під час розробки гри, але коли програмний продукт був повністю готовий, було проведено функціональне тестування, щоб переконатися, що основний функціонал програмного продукту працює правильно, відповідно з поставленими задачами.

Функціональне тестування є одним з основних видів тестування, яке зазвичай проводиться після завершення розробки програмного забезпечення та перед його випуском. Функціональне тестування спрямоване на перевірку функціональності системи. Його основна мета полягає в перевірці того, чи відповідає програмне забезпечення визначеним вимогам та очікуванням користувачів.

Щоб перевірити основний функціонал на працездатність, була створена таблиця тестів.

Функціональні тести

Дія	Очікувана реакція	Отримана реакція
1.	2.	3.
Натискання на кнопки головного меню	Кнопки діють згідно призначення: початок нової гри, вибір рівня, вихід з програмного продукту	Кнопки діють згідно призначення: початок нової гри, вибір рівня, вихід з програмного продукту
Виклик меню паузи	Зупиняється час в грі, користувач втрачає контроль над ігровим персонажем, кнопки діють згідно призначення: повернутися до гри, вийти в головне меню, вийти з програмного продукту	Зупиняється час в грі, користувач втрачає контроль над ігровим персонажем, кнопки діють згідно призначення: повернутися до гри, вийти в головне меню, вийти з програмного продукту
Стрільба зі зброї, в якій закінчились патрони в магазині	Постріли не відбуваються, не зважаючи на те, що гравець натискає кнопку	Постріли не відбуваються, не зважаючи на те, що гравець натискає кнопку
Вибір зброї по натисканню відповідної кнопки на клавіатурі	Герой міняє поточну зброю, змінюється кількість боєприпасів, що знаходяться в магазині	Герой міняє поточну зброю, змінюється кількість боєприпасів, що знаходяться в магазині
Стрільба з кожного типу зброї по супернику	У суперників віднімається кількість здоров'я, яка відповідає силі пошкодження кожної зброї. Ракетниця наносить пошкодження по області. Залишаються сліди від куль. Віднімаються боєприпаси. Працюють звуки.	У суперників віднімається кількість здоров'я, яка відповідає силі пошкодження кожної зброї. Ракетниця наносить пошкодження по області. Залишаються сліди від куль. Віднімаються боєприпаси. Працюють звуки.
Підбір power-up	Power-up коректно відображається з усіх сторін, power-up зникає, коли ігровий персонаж достатньою близько до нього підходить, змінюється показники здоров'я, броні, боєприпасів відповідно до типу power-up.	Power-up коректно відображається з усіх сторін, power-up зникає, коли ігровий персонаж достатньою близько до нього підходить, змінюється показники здоров'я, броні, боєприпасів відповідно до типу power-up.
Відкриття дверей	Двері відчиняються, коли гравець до них підходить та натискає відповідну клавішу.	Двері відчиняються, коли гравець до них підходить та натискає відповідну клавішу.

1.	2.	3.
Штучний інтелект суперників	Суперники переслідують головного героя, атакують його через певний інтервал. Суперники з далекобійною атакою не підходять до героя ближче, ніж вказана мінімальна дистанція. Групи суперників, розташовані на усіх трьох рівнях, активізуються, коли герой робить необхідну для цього дію.	Суперники переслідують головного героя, атакують його через певний інтервал. Суперники з далекобійною атакою не підходять до героя ближче, ніж вказана мінімальна дистанція. Групи суперників, розташовані на усіх трьох рівнях, активізуються, коли герой робить необхідну для цього дію.
Закінчення ігрового рівня	Коли персонаж наближається до фінальної точки, гра переходить до наступного рівня, а якщо це останній рівень, то повертається в головне меню	Коли персонаж наближається до фінальної точки, гра переходить до наступного рівня, а якщо це останній рівень, то повертається в головне меню

Під час функціонального тестування не було виявлено недоліків, що пов'язано з тим, що гра не являється складною, а більшість багів була виявлена та усунена в процесі розробки програмного продукту, але це не виключає можливості того, що в подальшому будуть виявлені помилки в грі. Але це є нормою, оскільки вичерпне тестування неможливе.

Також було проведено тестування, що перевіряло, яке навантаження гра дає на систему. Для цього гра, яка не є довгою, була пройдена з початку до кінця, щоб упевнитись, що не виникає ситуацій, де перевантажується система. Для замірів використовувався головним чином інструмент Profiler, який слідкує не тільки за тим, яке дає навантаження на систему, а точніше на центральний процесор, на оперативну пам'ять та на відеокарту, програмний продукт, но також дозволяє відслідкувати, який саме метод та скрипт навантажує систему.

Тестування проводились на комп'ютері з наступними технічними характеристиками:

– ОС: Windows 10 x64

– Процесор: AMD Ryzen 5 1600 Six-Core 3.2 GHz

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

- Оперативна пам'ять: 16 Гб
- Відеокарта: Geforce GTX 1050 ti
- DirectX: 12

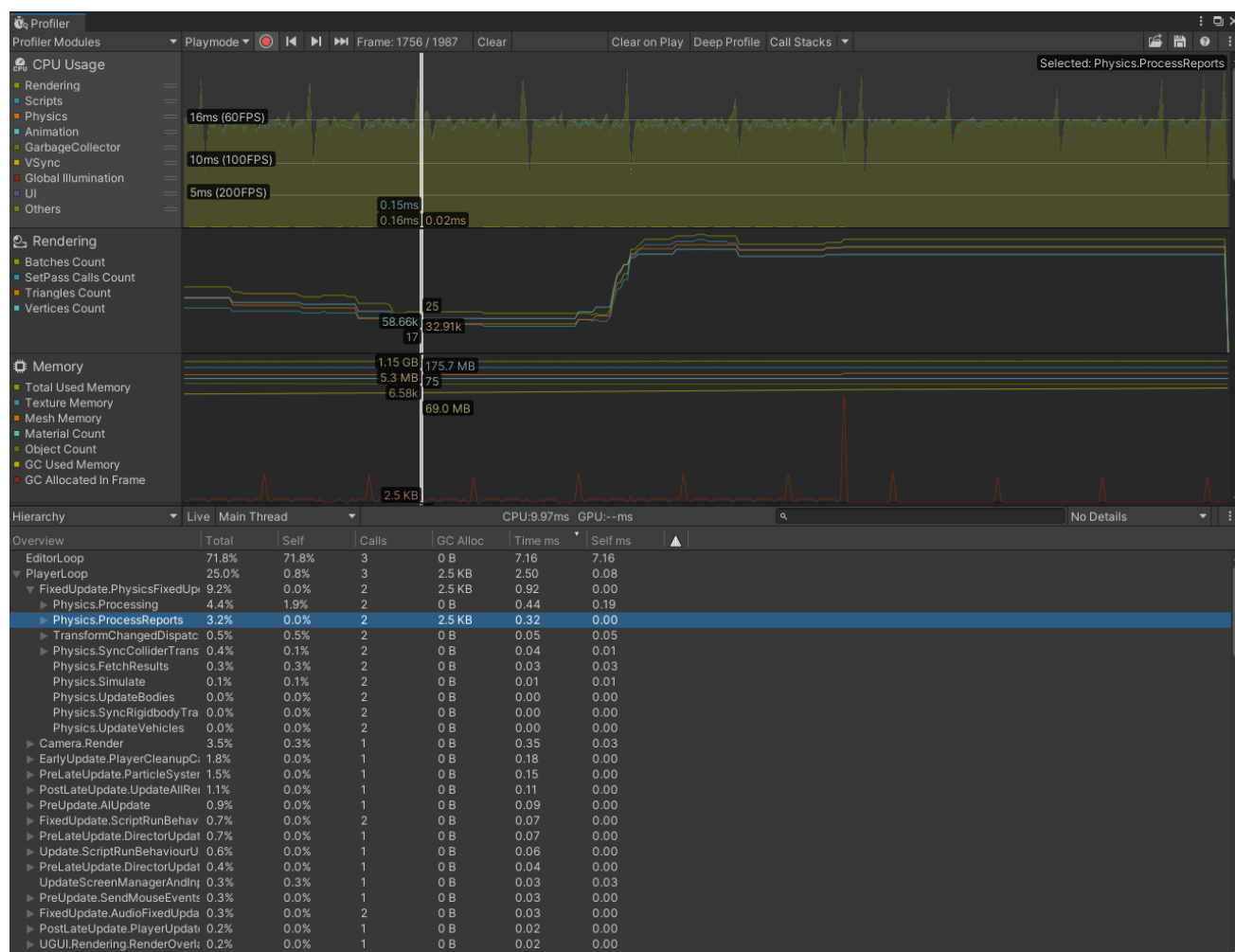


Рис 3.5 Показники Profiler

Проведене тестування показало, що під час гри немає критичних навантажень на систему. Гра працює в 60 кадрів в секунду з незначними відхиленнями, які не мають суттєвого впливу на ігровий процес. Оптимізація на даному етапі не є пріоритетною задачею.

Висновки до розділу 3

В останньому розділі працювали з готовим програмним продуктом. Було розглянуто користувацький інтерфейс гри: головне меню, меню паузи, відображення характеристик героя на екрані. Також були наведені клавіші

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

управління ігровим персонажем. Далі були проведені функціональні тести, де перевірялося, чи гра працює згідно поставлених цілей, чи немає в грі критичних помилок, які би унеможливлювали ігровий процес або псували його. Також було проведено тестування на те, яке навантаження на систему дає гра. Було виявлено, що кількість кадрів в секунду майже завжди тримається на рівні 60, що є гарним показником.

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

ВИСНОВКИ

У ході роботи над кваліфікаційною роботою була створена комп'ютерна гра в жанрі шутер від першої особи. Для розробки був використаний ігровий рушій Unity, тривимірні редактори Blender та MagicaVoxel, графічний редактор Aseprite та мова програмування C#.

У вступній частині була розкрита актуальність розробки комп'ютерних ігор, показано, що це прибуткова, але висококонкурентна справа у сучасному світі.

У першому розділі була поставлена задача по розробці шутеру від першої особи, яка передбачала написання програмного коду та створення тривимірних моделей. Для аналізу був обраний аналогічний продукт, щоб рівнятися на нього під час роботи над грою. В заключенні розділу були перелічені програмні продукти, які використовувалися під час створення гри, а також дана коротка характеристика цього інструментарію та наведено обґрунтування вибору.

В другому розділі, спираючись на раніше поставлену задачу, були наведені основні варіанти використання програми, що описували поведінку системи у відповідь на дію гравця. Архітектура програмного забезпечення була представлена у вигляді діаграми класів, де були описані їх атрибути (поля) та операції (методи), показані взаємозв'язки між класами. Коротко розповідалось про призначення та роль кожного класу.

Останній, третій розділ показував, як працювати з готовим програмним продуктом, встановлення та запуск гри. Було проведено тестування програмного продукту на наявність помилок та зазначені його результати. Показано, яке навантаження на систему дає гра, чи достатньо добре оптимізований код та чи не призводить від до перенавантажень.

Програмний продукт, який був створений, в цілому відповідає поставленій задачі. Звісно, в рамках кваліфікаційної роботи не могла бути розроблена гра високої якості, яка буде мати фінансовий успіх на ринку. Такі ігри зазвичай розробляються досвідченою командою впродовж місяців або навіть років. Але дана гра має в наявності весь базовий функціонал шутеру від першої особи: вид

					КН.КР.Б – 122 – 23 – ПЗ	Арк.
						59
Змн.	Арк.	№ докум.	Підпис	Дата		

камери «з очей», переміщення по карті, стрільба з різної зброї, взаємодія з об'єктами оточення, вороги зі штучним інтелектом, перехід між рівнями.

Для реалізації усіх цих функцій потрібно було продемонструвати володіння рушієм Unity, різноманітними його компонентами, як, наприклад, системою навігації, та володіння мовою програмування C#, застосування ООП. А також були створені найпростіші воксельні моделі зброї та суперників, проведена їх анімація. Були намальовані текстури та спроектовані ігрові рівні.

					КН.КР.Б – 122 – 23 – ПЗ	Арк.
						60
Змн.	Арк.	№ докум.	Підпис	Дата		

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Felicia P. Unity From Zero to Proficiency (Intermediate): A step-by-step guide to coding your first FPS in C# with Unity. 2016. 665 p.
2. Smith M., Ferns S., Gegan C. Unity 2021 Cookbook: Over 140 recipes to take your Unity game development skills to the next level. Packt Publishing, 2021.
3. Ferrone H. Learning C# by Developing Games with Unity: Get to grips with coding in C# and build simple 3D games in Unity 2022 from the ground up. 7th ed. Packt Publishing, 2022. 460 p.
4. Booch G., Rumbaugh J., Jacobson I. The Unified Modeling Language User Guide. 2nd ed. Addison-Wesley Professional, 2005. 475 p.
5. Fowler M. UML Distilled: A Brief Guide to the Standard Object Modeling Language. 3rd ed. Addison-Wesley Professional, 2003. 208 p.
6. Buttfield-Addison P., Manning J., Nugent T. Unity Game Development Cookbook: Essentials for Every Game. O'Reilly Media, 2019. 405 p.
7. Whitaker R. The C# Player's Guide. 5th ed. 2022. 494 p.
8. Miles R., Hamilton K. Learning UML 2.0: A Pragmatic Introduction to UML. O'Reilly Media, 2006. 284 p.
9. P. Doran J. Building an FPS Game with Unity. Packt Publishing, 2015. 404 p.
10. W. Murray J. C# Game Programming Cookbook for Unity 3D. 2nd ed. CRC Press, 2021. 302 p.
11. Video game industry - Statistics & Facts. URL: <https://www.statista.com/topics/868/video-games/#topicOverview>.
12. Unity Manual. URL: <https://docs.unity3d.com/Manual/index.html>.

ДОДАТКИ

					КН.КР.Б – 122 – 23 – ПЗ	Арк.
						62
Змн.	Арк.	№ докум.	Підпис	Дата		

Лістинг файлу EnemyMoving

```

using System.Collections;
using System.Collections.Generic;
using Unity.VisualScripting;
using UnityEngine;
using UnityEngine.AI;

public class EnemyMoving : MonoBehaviour
{
    private Transform player;
    public bool shouldMove;
    public NavMeshAgent agent;
    public float distanseToStopMovement = 5;
    private bool controlPos = false;
    public Vector3 previousPos;
    private Enemy enemy;
    public bool isFacingPlayer = false;
    public bool isAwakened = false;

    // Start is called before the first frame update
    void Start()
    {
        enemy = GetComponent<Enemy>();
        player = GameObject.FindObjectOfType<PlayerControl>().gameObject.transform;
        agent = GetComponent<NavMeshAgent>();
        shouldMove = true;
        if (isAwakened)
        {
            MoveToPlayer();
        }
        agent.isStopped = false;
        previousPos = agent.transform.position;
    }

    void Update()
    {
        if (controlPos && agent.isStopped == true)
        {
            agent.velocity = Vector3.zero;
        }
    }

    private void FixedUpdate()
    {
        if (isAwakened)
        {
            MoveToPlayer();
            if (isFacingPlayer)
            {
                FacePlayer();
            }
        }
    }

    public void Disactivate()
    {
        isAwakened = false;
        StopMoving();
    }

    public void Activate()
    {

```

					КН.КР.Б – 122 – 23 – ПЗ	Арк.
						63
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        isAwakened = true;
    }

    public void StopMoving()
    {
        controlPos = true;
        //print("stop");
        agent.isStopped = true;
        agent.velocity = Vector3.zero;
        shouldMove = false;
    }

    public void StartMoving()
    {
        shouldMove = true;
        enemy.Animator.SetBool("isStanding", false);
        isFacingPlayer = false;
    }

    public void MoveToPlayer()
    {
        if (shouldMove == true)
        {
            if (Vector3.Distance(player.position, agent.gameObject.transform.position) >
distanseToStopMovement)
            {
                agent.isStopped = false;
                agent.SetDestination(player.position);
                enemy.Animator.SetBool("isStanding", false);
            }
            else
            {
                Standing();
            }
        }
    }

    private void Standing()
    {
        FacePlayer();
        enemy.Animator.SetBool("isStanding", true);
        agent.isStopped = true;
    }

    public void BlockAreaMovent()
    {
        agent.areaMask = (1 << 0) + (0 << 1) + (0 << 2) + (0 << 3);
    }

    public void UnblocAreaMovent()
    {
        agent.areaMask = (1<<0)+(0<<1)+(0<<2)+(1<<3);
    }

    public void FacePlayer()
    {
        Vector3 dir = player.transform.position - transform.position;
        Quaternion lookRotation = Quaternion.LookRotation(dir);
        Vector3 rotation = lookRotation.eulerAngles;
        agent.transform.rotation = Quaternion.Euler(0f, rotation.y, 0f);
    }

```

Лістинг файлу EnemyAttacking

```

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

public class EnemyAttacking : MonoBehaviour
{
    private PlayerStats player;
    public int damage;
    public float attackCooldown;
    private float currentTimeCooldown;
    public float attackDistance;
    [SerializeField] private LayerMask layermask;
    private EnemyAnimation anim;
    private Enemy enemy;

    public bool isProjectile;
    public Transform muzzle;
    public GameObject projectile;

    // Start is called before the first frame update
    void Awake()
    {
        player = GameObject.FindObjectOfType<PlayerStats>().GetComponent<PlayerStats>();
        currentTimeCooldown = attackCooldown;
        anim = GetComponent<EnemyAnimation>();
        enemy = GetComponent<Enemy>();
        currentTimeCooldown = Random.Range(0f, (float)attackCooldown);
    }

    // Update is called once per frame
    void Update()
    {
        if(enemy.enemyMoving.isAwakened)
        {
            Attack();
        }
    }

    private void Attack()
    {
        currentTimeCooldown -= Time.deltaTime;
        if(currentTimeCooldown < 0 && attackDistance >=
Vector3.Distance(transform.position,player.transform.position))
        {
            if(isProjectile)
            {
                anim.Shoot(1f);
                StartCoroutine(ShootProjectile());
            }
            else if(attackDistance < 4)
            {
                anim.Shoot(0.5f);
                StartCoroutine(Melee());
            }
            else
            {
                anim.Shoot(1f);
                MakeDamage();
            }
            currentTimeCooldown = attackCooldown;
        }
    }
}

```

					КН.КР.Б – 122 – 23 – ПЗ	Арк.
						65
Змн.	Арк.	№ докум.	Підпис	Дата		

```

    }
}

private IEnumerator ShootProjectile()
{
    yield return new WaitForSeconds(0.5f);
    Vector3 pos = muzzle.position;
    Transform t = muzzle.transform;
    GameObject bullet = GameObject.Instantiate(projectile, pos, Quaternion.identity);
    bullet.GetComponent<RocketBullet>().isByEnemy = true;

    bullet.transform.rotation = Quaternion.Euler
        (-270,
        muzzle.transform.parent.transform.parent.rotation.eulerAngles.y,
        0);
}

private IEnumerator Melee()
{
    yield return new WaitForSeconds(0.5f);
    if(attackDistance >= Vector3.Distance(transform.position,
player.transform.position))
    {
        MakeDamage();
    }
}

private void MakeDamage()
{
    player.TakeDamage(damage);
}
}

```

					КН.КР.Б – 122 – 23 – ПЗ	Арк.
						66
Змн.	Арк.	№ докум.	Підпис	Дата		