

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

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

Кафедра програмних засобів
(повне найменування кафедри)

Пояснювальна записка

до дипломного проекту (роботи)

бакалавр
(ступінь вищої освіти)

на тему РОЗРОБКА ТЕХНІЧНОЇ ДЕМОНСТРАЦІЙНОЇ ВЕРСІЇ
КОМП'ЮТЕРНОЇ ГРИ
DEVELOPMENT OF A DEMO VERSION OF A COMPUTER GAME

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

Освітня програма (спеціалізація)
Комп'ютерні науки

Власова К.А.

(прізвище та ініціали)

Керівник Зайко Т.О.

(прізвище та ініціали)

Рецензент Гофман Є.О.

(прізвище та ініціали)

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

Інститут, факультет ІРЕ, ФКНТ

Кафедра програмних засобів

Ступінь вищої освіти бакалавр

Спеціальність 122 Комп'ютерні науки

(код і найменування)

Освітня програма (спеціалізація) Комп'ютерні науки

(назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.

С.О. Субботін

“ ” 2021 року

З А В Д А Н Н Я

НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

Власової Карини Андріївни

(прізвище, ім'я, по батькові)

1. Тема проєкту (роботи) Розробка технічної демонстраційної версії комп'ютерної гри. Development of a Demo Version of a Computer Game.

керівник проєкту (роботи) Зайко Тетяна Андріївна, к.т.н., доцент,

(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

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

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

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

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз проблеми та постановка завдань дослідження. 2. Аналіз вимог до програмного забезпечення. 3. Проєктування програми. 4. Керівництво програміста. 5. Керівництво користувача

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

6. Консультанти розділів проєкту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-5 Основна частина	Зайко Т.О., доцент		
Нормоконтроль	Липовець М.В., асистент		

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

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Постановка завдання роботи.	1 тиждень	Завдання, ТЗ
2	Аналіз предметної області.	2–3 тижні	Розділ 1
3	Дослідити існуючі аналоги програмного продукту і виявити їхні переваги і недоліки;	4–5 тижні	Розділ 2
4	Зпроектувати алгоритми для реалізації ігрового програми з урахуванням переваг і недоліків існуючих аналогів;	6 тиждень	Розділ 3
5	Реалізувати складені алгоритми на мові програмування C #;	7–8 тижні	Розділ 3
6	Розробити керівництво програміста	9 тиждень	Розділ 4
7	Розробити керівництво користувача	10 тиждень	Розділ 5
8	Оформлення висновків	11 тиждень	Висновки
9	Оформлення пояснювальної записки та документів до неї.	12-13 тиждень	Додатки
10	Нормоконтроль та рецензування.	14–15 тижні	
11	Захист роботи.	16 тиждень	

Студент(ка)

(підпис) Власова К.А.
(прізвище та ініціали)

Керівник проєкту (роботи)

(підпис) Зайко Т.А.
(прізвище та ініціали)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи бакалавра: 86 с., 1 табл., 34 рис., 2 дод., 14 джерел.

КОМП'ЮТЕРНА ГРА, RPG, UNITY, OPENGL, ІГРОВИЙ ПРОЄКТ, КОМП'ЮТЕРНА ГРАФІКА, МОВА C#, CRPG.

Об'єкт дослідження – процеси реалізації демонстраційної версії комп'ютерної гри.

Предмет дослідження – методи розробки комп'ютерних ігор.

Мета роботи – розробка технічної демонстраційної версії комп'ютерної гри.

Матеріали, методи та технічні засоби: об'єктно-орієнтована мова програмування C#, JavaScript, Boo, міжплатформене середовище розробки комп'ютерних ігор Unity, персональний комп'ютер з процесором Intel Core 7 під управлінням операційної системи Microsoft Windows 10.

Результати. Розроблено технічну демонстраційну версію комп'ютерної гри. Реалізовано підсистеми пошуку шляху, графічного інтерфейсу користувача і взаємодії об'єктів.

Висновки. Розроблено програмну реалізацію технічної демонстраційної версії комп'ютерної гри.

Галузь використання – комп'ютерні ігри.

ABSTRACT

Explanatory note to the diploma qualifying work of the bachelor: 86 pages, 34 figures, 1 tables, 2 appendixes, 14 sources.

COMPUTER GAME, RPG, UNITY, OPENGL, GAME PROJECT, COMPUTER GRAPHICS, C # LANGUAGE, CRPG.

Object of study is is the processes of realization of the demonstration version of the computer game.

Subject of study is methods of computer game development.

The purpose of the work is to develop a technical demonstration version of a computer game.

Materials, methods, and tools: bject-oriented programming language C #, JavaScript, Boo, cross-platform environment for computer game development Unity, personal computer with Intel Core 7 processor running Microsoft Windows 10 operating system.

Results. A software implementation of a technical demonstration version of a computer game has been created.

Conclusions. Software implementation of technical demonstration version of computer game is developed, subsystems of search of a way, the graphical user interface and interaction of objects are realized.

The field of use – computer games.

ЗМІСТ

С.

Вступ.....	8
1 Аналіз проблеми та постановка завдань дослідження	9
1.1 Жанри комп'ютерних ігор	10
1.2 Класифікація комп'ютерних ігор	12
1.2.1 Опис класифікації ігор за ігровою платформою	14
1.2.2 Опис класифікації комп'ютерних ігор за графікою	16
2 Аналіз вимог до програмного забезпечення	18
2.1 Вимоги до програмного забезпечення	18
2.1.1 Вибір рушії	19
2.1.2 Мова програмування C #.....	21
2.1.3 Середовище розробки графічних об'єктів	23
2.2 Аналіз даних	25
3 Проєктування програми	28
3.1 Аналіз алгоритмів.....	28
3.2 Архітектура програми	34
4 Керівництво програміста	38
4.1 Призначення розробки	38
4.2 Вимоги до програми.....	40
4.3 Вимоги до вхідних та вихідних даних.....	40
5 Керівництво користувача.....	43
5.1 Установка, запуск, мінімальні вимоги і склад програми	43
5.2 Елементи інтерфейсу та управління.....	45
Перелік джерел посилання	48
Додаток А	50
Додаток Б.....	80

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

CRPG – Computer Role-Playing Game;

GUI – Graphical User Interface;

HP – Hit Points;

RPG – Role-Playing Game;

БД – база даних;

ІТ – інформаційні технології;

ПЗ – програмні засоби;

ПК – персональний комп'ютер.

ВСТУП

Комп'ютерна гра – комп'ютерна програма, яка необхідна для організації ігрового процесу, організації зв'язку з гравцями.

Комп'ютерні ігри з кожним днем охоплюють все більшу аудиторію по всьому світу та використовуються на різних ігрових пристроях. Дані розробки з часом стають складнішими та масштабнішими: покращилася обробка візуального простору, обробка фізики об'єктів та штучний інтелект. У віртуальному просторі може перебувати величезна кількість різноманітних об'єктів, але їх кількість, як і раніше залежить від технічних можливостей пристроїв, на які розрахована гра. Інтерактивність в такій програмі зводиться до взаємодії між об'єктами. Вони можуть пересилати інформацію одне одному, навколишньому простору і користувачеві в тому числі.

Основною проблемою є завантаженість коду об'єктів взаємодії, що виконують, закріплений за ними, певний сценарій [1]-[2]. Виходячи з вищевикладеного було прийнято рішення створити технологічну комп'ютерну гру з використанням сучасних середовищ і способів розробки.

Об'єкт дослідження – процеси реалізації демонстраційної версії комп'ютерної гри.

Предмет дослідження – методи розробки комп'ютерних ігор.

Мета роботи – розробка технічної демонстраційної версії комп'ютерної гри.

У зв'язку з поставленою метою вирішуються такі завдання:

- дослідити існуючі аналоги програмного продукту і виявити їхні переваги і недоліки;
- спроектувати алгоритми для реалізації ігрової програми з урахуванням переваг і недоліків існуючих аналогів;
- реалізувати складені алгоритми на мові програмування C #;
- провести тестування візуального представлення алгоритмів.

1 АНАЛІЗ ПРОБЛЕМИ ТА ПОСТАНОВКА ЗАВДАНЬ ДОСЛІДЖЕННЯ

Досягнення комп'ютерних технологій впливають на всі галузі комп'ютерного мистецтва, відкриваючи перед цифровими художниками все більше можливостей і втілюючи в реальність вражаючих істот. В наш час комп'ютерні ігри стають все більш популярними. Кожен день створюється багато ігор, програм, які розвиваються та стають популярними.

Комп'ютерні ігри відносяться до числа актуальних проблем сучасної науки, тому що представляють собою унікальний продукт розвитку техніки і сучасної особистості. Ігри допомагають змодельовати різні життєві ситуації, проблеми та допомагають знайти можливі шляхи їх вирішення. Гра містить у собі всі необхідні передумови для природного розвитку особистості і культури суспільства. З вдосконаленням комп'ютерів також вдосконалювалися і гри, залучаючи все більше і більше людей. На сьогоднішній день комп'ютерна техніка досягла такого рівня розвитку, що дозволяє програмістам розробляти дуже реалістичні ігри з гарним графічним і звуковим оформленням.

Розробка комп'ютерних ігор – це процес створення комп'ютерних ігор. Розробленням відеоігор займається розробник, який може бути представлений як однією людиною, так і цілою командою. Зазвичай масштабні комерційні ігри розробляються командами розробників в межах компанії, які спеціалізується на іграх для персонального комп'ютера (ПК) або консолей. Як правило, розроблення фінансує велика компанія-видавець, яка після розроблення займається виданням гри та пов'язаними з ним витратами. Рідше компанії-видавці можуть містити внутрішні команди розробників, також компанія-розробник може розробляти ігри за свій рахунок і поширювати їх без участі видавців, наприклад, засобами цифрової дистрибуції.

Завдяки розвитку ринку незалежних або інді-ігор, велика кількість розробників отримали можливість працювати над ігровими проєктами без фінансових та юридичних зобов'язань перед компаніями-видавцями. Незалежні ігри – це комп'ютерні ігри, які створені окремими розробниками

або невеликими колективами без фінансової підтримки видавця комп'ютерних ігор. Поширення даних розробок здійснюється за допомогою каналів цифрової дистрибуції.

Технічна демонстрація, або технічна демонстраційна версія, техно-демоверсія – це прототип, наближений приклад або неповна версія продукту, яка створена з метою продемонструвати ідею, продуктивність, метод або особливості будь-якого програмного продукту. Технічні демонстраційні версії можуть використовуватися як демонстрації для інвесторів, партнерів, журналістів або навіть потенційних споживачів для того, щоб переконати їх в життєздатності даного продукту, матеріального товару або ідеї.

1.1 Жанри комп'ютерних ігор

Можна з самого початку уявляти в деталях свою готову гру, а можна по мірі розроблення доповнювати сюжет, стиль та особливості гри. У даній справі не обов'язкова зайва точність, але, як мінімум, потрібно задати напрямок розвитку ігрового проекту. Жанр гри необхідно вибрати на самому початку в обов'язковому порядку. Жанр і буде основним напрямком розвитку гри.

Прийнято класифікувати ігри по жанрах, які визначають сукупність характерних рис, притаманних тій чи іншій грі. Цей підхід не стандартизований, але популярний і його дотримуються як представники ігрової індустрії, так і споживачі ігрових додатків. Основні жанри приведені нижче:

– аркади – це категорія комп'ютерних ігор з простим ігровим процесом, що зводиться до виконання обмеженого набору дій, націлених на досягнення певного результату. Завдання ігор цього жанру, як правило, зводяться до набору максимальної кількості очок і / або мінімального часу проходження гри;

- квести – здійснюють подорож одного або декількох персонажів до поставленої мети шляхом подолання різноманітних труднощів;
- екшн (action) – ігри від першої особи – популярні дії-стрілянина;
- рольові ігри або RPG (role-playing game) – гравець виконує роль певного персонажа і виконує поставлені перед ним завдання;
- стратегії та логічні ігри наслідують діяльності управління;
- симулятори імітують керування автомобілем, космічним кораблем, літаком та ін.
- азартні та логічні добре розвивають розумову діяльність.

Також комп'ютерні ігри можна згрупувати за такими ознаками як наявність персонажа вибору.

Існуючі платні і безкоштовні ігри, в яких є персонаж, різноманітні: це екшн, RPG, деякі види стратегій, квести.

Ігри без персонажа - логічні, симулятори. Вони не мають сюжету, не викликають сильного звикання і не впливають на психіку. Часто пропонують інтелектуальні завдання, що добре розвиває мислення.

Стратегії – ігри з відсутністю персонажа, де процесом гри управляє людина. Відмінною особливістю стратегії є особлива система внутрішньоігрових факторів. Спочатку гравець приймає певні рішення і спостерігає за їх наслідками, а потім вибудовує логіку подальших дій.

Найвідоміші гри, в яких є право вибору - це RPG, іноді екшн і квести. Гравець самостійно вибирає персонажа, одягає його, наділяє здібностями. У грі передбачена свобода дій, яка впливає на процес і результат гри. Даний жанр гри викликає стійке звикання, гравець ототожнює себе з персонажем, розвиває його за своїм образом і подобою. Ігровий процес сприймається як реальність.

Ігри з відсутністю вибору мають лінійний сюжет (стрілялки). Гравець ототожнює себе з готовим персонажем, вже наділеним характеристиками. Розгалуження сюжету гри залежать не від вибору гравця, а від успіху чи випадковості.

1.2 Класифікація комп'ютерних ігор

Можна виділити три основних ієрархічних рівня, які дозволяють побудувати схему гри:

- оперативний рівень (локальні мережі, що ускладнюють гри);
- тактичний рівень (поточне управління клавішами);
- стратегічний рівень (кінець гри, фіксація результатів).

На першому, оперативному рівні відбувається зміна об'єктів в ігровому просторі за допомогою натискання клавіш клавіатури, миші або іншого пристрою, що управляє (джойстик). В результаті на екрані дисплея користувачеві повинні бути відображені всі переміщення і зміни.

Тактичний рівень включає в себе оперативний. Всі дії на цьому рівні ведуть до досягнення деякої певної мети. На цьому етапі відбувається зміна рівня складності, темпу гри.

Стратегічний рівень містить в собі тактичний з додаванням кількох самостійних блоків, таких як введення на ігровий простір всіх об'єктів для визначення завдання і візуалізації їх початкових параметрів. Крім цього, на даному рівні відбувається перевірка критеріїв закінчення гри, фіксації та візуалізації результатів всієї гри в цілому і підведення підсумків минулих ігор.

За структурою ігри діляться на три блоки:

- блок ігрового середовища (правило гри);
- блок оцінки ігрової ситуації (аналіз);
- блок взаємодії з гравцем (інтерфейс).

Загальна структура комп'ютерних ігор представлена на рис. 1.1

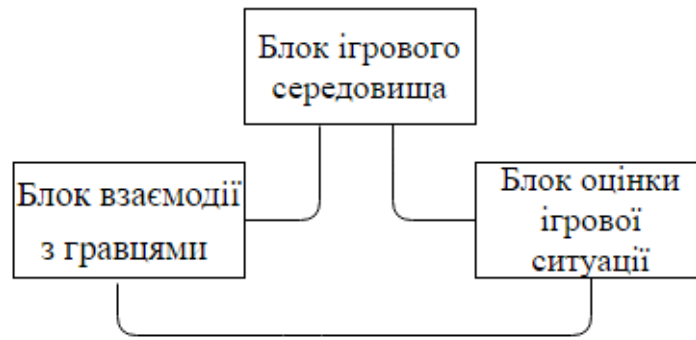


Рисунок 1.1 – Загальна структура комп'ютерних ігор

Блок ігрового середовища - це той простір, в якому відбувається ту чи іншу дію у відповідності зі змістом, задумкою розробника і законами самої гри.

Блок взаємодії з гравцями – це складові програми, які дозволяють гравцеві змінювати хід гри.

Блок оцінки – це умови, створені гравцем до об'єкту гри всередині ігрового простору. Це такі засоби, як підрахунок очок, опис умов, показ початкової або кінцевої ігрової ситуації.

Ігри можуть належати одній платформі або бути мультиплатформенними.

Класифікація по платформах:

- персональний комп'ютер;
- ігрові консолі;
- мобільні пристрої.

Також за кількістю платформ, на яких може запуститися гра:

- мультиплатформенні – ігри, здатні запуститися на декількох платформах;
- одноплатформенні – ексклюзивні ігри, створені тільки для однієї певної платформи [1].

Класифікація комп'ютерних ігор ускладнюється тим, що багато ігор випускаються в декількох варіантах з різними сюжетними лініями, завданнями і можливостями.

Також існує класифікація ігор за критерієм виконуваних дій[2]:

- ігри спілкування: отримання інформації, спілкування, вивчення світу;
- ігри дій: переміщення в просторі, використання зброї і техніки;
- ігри контролю: командування, управління, розподіл матеріальних засобів.

Всередині груп задаються кордони, що визначають подальшу деталізацію за допомогою уточнюючих критеріїв і з використанням усталених назв категорій. Такий підхід здається більш раціональним і системним, хоча і більш складним.

1.2.1 Опис класифікації ігор за ігровою платформою

Основним способом поділу відеоігор на категорії є поділ за платформами, який вказує, на якому пристрої можна запустити гру. Якщо у користувача немає платформи, для якої призначена гра, то і запустити її не вдалося [2].

Одноплатформена гра або ексклюзивна - гра, розроблена для однієї платформи. Проте, дуже часто розробники намагаються охопити найбільшу аудиторію, тому випускають відразу кілька варіантів однієї гри. Кожен з варіантів гри підходить для своєї апаратної платформи.

Мультиплатформенна гра - гра, розроблена відразу для декількох платформ. До основних платформ можна віднести: персональний комп'ютер, ігрова консоль, мобільний пристрій, ігровий автомат,браузер.

Персональний комп'ютер (ПК). Завдяки своїй ігровій спрямованості, персональні комп'ютери оснащені відеокартами, звуковими картами, процесорами великої потужності.

ПК в свою чергу поділяються на кілька підплатформ за операційними системами (ОС). У кожній ОС розроблені свої інструменти для обробки

відеоігор, тому не всі ігри можуть підійти відразу до всіх ОС. Перелік найбільш популярних серій комп'ютерних ОС: Windows, Mac OS, Linux.

Ігрові консолі - це, ті ж комп'ютери, але лише з однією функцією – грою. Для роботи більшості консолей необхідно під'єднувати їх до телевізора, тому їх ще часто називають ігровими приставками. Але є незалежні портативні консолі, з вбудованим екраном. На відміну від комп'ютерів, консоль - готовий нерозбірний пристрій. До найбільш популярних консолей можна віднести: Sony PlayStation (PSP, PSOne, PS2, PS3, PS4), Microsoft Xbox (Xbox, Xbox 360, Xbox One), Nintendo 3DS, Wii.

Мобільний пристрій: телефон, планшет, кишеньковий комп'ютер (КПК, PDA). Мобільні пристрої за технічними характеристиками набагато слабкіше стаціонарних комп'ютерів, тому мобільні гри виглядають дещо простіше звичайних ігор, але ситуація поступово поліпшується.

Аркадний ігровий автомат – пристрій у вигляді шафи з екраном, що має всередині себе одну встановлену гру, і вимагає оплату за кожен сеанс гри. Подібні автомати, як правило, встановлюють в часто відвідуваних публічних місцях.

Браузерна гра – гра, яка запускається в браузері, з обов'язковим створенням профілю, що дозволяє грати разом з іншими гравцями. Це різновид ММО-ігор, в яких відсутня повноцінний режим відео, замість нього використовується графіка, намальована на інтернет-сторінках, з використанням флеш-відеовставок. Браузерні ігри можуть бути дуже високої якості, але, не дивлячись на це, поширюються вони безкоштовно. найчастіше такі ігри влаштовані за схемою free2play з мікротранзакції коли в грі можна купувати якісь віртуальні речі за реальні гроші.

1.2.2 Опис класифікації комп'ютерних ігор за графікою

Зовнішній вигляд — головна прикраса ігри. Багато недосвідчених гравців при виборі гри орієнтуються саме на графіку, тому існує поділ ігор за типом і якістю графічного зображення.

Відсутність графіки — не велика потужність комп'ютерів перших поколінь ставила перед розробниками ігор масу обмежень. Через це у багатьох старих іграх застосовувалася графіка, та текстове оформлення. Ігри такого виду більше схожі на інтерактивну книгу, а не на відеогру.

Двомірна— найбільш природний вигляд графіки. Зображення складені з окремих пікселів - кольорових квадратів. Також існує технологія векторних зображень, при якій об'єкти складаються з пікселів, а з точних геометричних координат, з'єднуються лініями. Такий вид зображень дозволяє малювати більш плавні лінії, без пікселізації. При збільшенні зображення не псується його зовнішній вигляд.

Тривимірна графіка. Завдяки застосуванню тригонометричних формул у розробників ігор з'явилася можливість створювати ілюзію тривимірного світу, що відображається на двомірній площині екрану. У комп'ютері обчислюються справжні 3D-моделі, а на екран виводиться математично обчислювані 2D-проекції цих тривимірних об'єктів. На сьогоднішній день 3Dграфіка - найпопулярніший формат графіки в комп'ютерних іграх.

Об'ємне зображення — це невелике поліпшення 3D-графіки. Удавана об'ємність об'єктів досягається за рахунок того, що глядачеві надходять різні зображення для лівого і правого ока. За рахунок різниці зображень людський мозок може обчислити приблизну відстань до об'єкта, і виходить ефект об'ємності.

Доповнена реальність - подібні ігри доступні лише на мобільних пристроях з відеокамерою. Через екран об'єктива під час відеозйомки відображається реальний світ, але з додаванням віртуальних об'єктів. Гравець водить відеокамерою, знаходить об'єкти або прицілюється і знищує їх [4].

Доповнена реальність - цікава і незвичайна ідея, але широкої популярності вона не отримала.

Віртуальна реальність – повне занурення гравця у віртуальний світ, коли гравець відчуває, що в відеогру поміщено все його тіло, всі його органи чуття. розвиток гри до повноцінної віртуальної реальності (VR) на сьогоднішній день неможливо. На даний момент існують шоломи віртуальної реальності, в які транслюється стерео-графіка. Сенсори, що закріплюються на руках, можуть відображати реальні рухи рук у віртуальному просторі.

Дану роботу можна віднести до незалежної розробки, яка розвивається за рахунок коштів розробників.

Завданням даної роботи є розробка технічної демонстраційної версії комп'ютерно рольової гри.

2 АНАЛІЗ ВИМОГ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Аналіз вимог – частина процесу розробки програмного забезпечення, що включає в себе збір вимог до програмного забезпечення, їх систематизацію, виявлення взаємозв'язків, а також документування.

В даному розділі розглянемо: вибір засобів реалізації та аналіз даних.

2.1 Вимоги до програмного забезпечення

Для вибору засобів реалізації було визначено ряд вимог до рушія, мови програмування та середовища розробки графічних об'єктів.

Основними вимогами до рушія є:

- можливість розробляти двох-трьох вимірні застосування;
- підтримка різних OCL;
- міжплатформена підтримка;
- модульна система компонентів.

Мова програмування для створення повинна бути об'єктно-орієнтованою та мати вбудовану підтримку:

- узагальнень;
- подій;
- делегатів;
- великої кількості бібліотек та шаблонів.

Середовище розробки графічних об'єктів повинно відповідати наступним вимогам:

- простота виведення, відображення графіки не повинно занадто навантажувати процесор комп'ютера, дана вимога необхідна для швидкого виведення зображення;
- висока точність і достовірність передачі зображень;
- підтримка стандартних форматів файлів.

2.1.1 Вибір рушії

Ігровий рушій - це базове ПЗ, на якому розробляється та функціонує гра, загальна назва для сукупності декількох підсистем (звукова, графічна, фізична та інші). В сучасних ігрових рушіїх дані підсистеми - модульні, і якість опрацювання у них різниться в різних рушіїх.

Розглянемо три найпопулярніші відкриті ігрові рушії: Unreal Engine 4, Cryengine 5 та Unity і оцінимо їх за наступними пунктами:

- доступність;
- підтримувані платформи;
- підтримка користувача;
- доступність ігрових ассетів та інструкцій.

Unreal Engine 4 – підходить для будь-яких завдань. Інтерфейс рушії інтуїтивно зрозумілий і не вимагає довгого звикання. Це велика перевага, тому що дозволяє швидко перейти від вивчення ПЗ до творчого процесу. Гнучкість у створенні проєкту дозволяє охопити майже будь-які платформи і механіки, а технологія Blueprints (побудова ігрової логіки за допомогою наочних схем) дозволяє створювати ігри без знання мов програмування.

Переваги:

- безкоштовний для некомерційного використання;
- можливість роботи на Windows та Linux-подібних системах;
- активна служба підтримки користувача на ринку та спільнота;
- через масове використання рушії ігрові ресурси та бібліотеки доступні в великій кількості;
- майже повна відсутність помилок.

Недоліки:

- розмір більш ніж в 20 Гб;
- завантаження Windows-версії відбувається через магазин.

Рушій володіє багатьма перевагами, однак це професійний інструмент, що працює з сотнями тисяч доповнень і розширень.

Cryengine 5 – має невиразний інтерфейс, що більше підходить закритим рушіям внутрішніх студій, а не масового продукту. Але продукт цілком логічний, що дозволяє освоїти рушій за лічені місяці. Незважаючи на позиціонування рушія як універсального, залишається найкращим засобом для створення шутерів, для інших цілей придатний мало. Знання мови програмування C ++ необхідно.

Переваги:

- безкоштовний для некомерційного використання;
- гарна графіка;
- малий розмір.

Недоліки:

- не активна служба підтримки;
- відсутність повної документації;
- непоширеність ігрових ассетів та повноцінних гайдів;
- мало придатний для проєктів, крім шутерів;
- присутня плутанина в інтерфейсах різних версій.

Unity – це ігровий рушій для розробки двох-або тривимірних застосувань та ігор, призначений для операційних систем Windows та OS X. Створене за допомогою Unity ігрове програмне забезпечення працює на таких операційних системах, як Windows, OS X, Android, Apple iOS, Linux, а також на ігрових приставках Wii, PlayStation, Xbox 360. Комп'ютерні ігри та інша продукція, створені за допомогою Unity, підтримують набір бібліотек DirectX і OpenGL.

Комп'ютерні ігри та інша продукція, які були створені за допомогою Unity, підтримують набір бібліотек DirectX та OpenGL.

Даний рушій дозволяє створити сцену, на якій будуть розміщені імпортовані з 3D's Max об'єкти взаємодії.

Основні особливості рушія:

- кілька сценарних мов програмування: C #, JavaScript (модифікація) і Boo;

- можливість миттєвого запуску гри;
- проста робота з ресурсами через Drag-and-Drop;
- широкі можливості імпорту;
- повністю налаштовується та доступний для більшості людей інтерфейс;
- кроссплатформенність;
- гнучкість та нескінченна розширюваність;
- наявність безкоштовної версії з деякими обмеженнями.

Недоліки:

- не досить висока продуктивність готових проєктів;
- низький рівень графіки в порівнянні з конкурентами.

На даний момент Unity є одним з найпростіших, але в той же час і найефективніших, ігрових рушіїв. Його можливості безмежні, а засоби розробки мають досить простий інтерфейс. Виходячи з цих параметрів, було обрано саме дане середовище розробки [5]-[6].

2.1.2 Мова програмування C #

Для вибору мови програмування було розглянуто кілька відомих мов.

C – одна з найстаріших і популярних мов програмування. Вона «легка» та швидка, тому її використовують там, де потрібна висока продуктивність. Наприклад, для створення драйверів, операційних систем або ПЗ для мікроконтролерів. При цьому C складно вивчити – багато доводиться писати з нуля. За допомогою C створюють драйвера, пишуть ядра операційних систем, а також пишуть бібліотеки для Python і інших мов.

Java - багатоплатформова мова з великою кількістю бібліотек і великим співтовариством розробників. Кросплатформеність - це можливість написати програму один раз і відразу користуватися нею на декількох операційних системах: Windows, Linux і MacOS. Завдяки бібліотекам Java може

використовуватися практично для всього: роботи з графікою, звуком, створення невеликих ігор. На ній пишуть мобільні застосунки для Android, програми для мікрохвильових печей і сервери.

Python - логічна та досить проста мова з мінімалістичним синтаксисом. У неї невеликий набір основних правил, мова легка для читання та на ній не складно писати. Розробники написали для Python безліч бібліотек, тому можна використовувати готові рішення в своїх проєктах. Головний мінус Python - її невисока швидкість. Програми на ній будуть працювати в середньому повільніше, ніж на інших мовах. Найчастіше на Python створюють сервери, обробляють дані і розробляють нейронні мережі.

C # - мова, спочатку придумана компанією Microsoft, щоб створювати застосунки для Windows. Це об'єктно-орієнтована мова - її складніше вивчати, але простіше використовувати, наприклад, менше писати однаковий код. За допомогою C # можна працювати з платформою WPF, яка допомагає створювати «гарні» віконні застосунки. Найчастіше на C # пишуть програми для Windows і створюють комп'ютерні ігри. Наприклад, на C # працює популярний рушій Unity, що і є її основною перевагою.

В якості першої ключової мови програмування в даній дипломній роботі обрано C # .

C # відноситься до мов з C-подібним синтаксисом, її синтаксис швидше близький до C ++ та Java. Успадкувавши багато від своїх попередників – C ++, Java Delphi, Java, Smalltalk – C #, спирається на практичне їх використання та виключає деякі моделі, що позиціонують себе як проблематичні при розробці програмного продукту.

Дана мова була обрана в якості основної, так як має потрібні якості для реалізації, має вбудовану підтримку узагальнень, делегатів та подій та працює з рушієм Unity [7].

2.1.3 Середовище розробки графічних об'єктів

У кожної гри є свій візуальний стиль. Він включає візуалізацію ігрового оточення і персонажів, спец ефекти і інтерфейс. Можна виділити основні типи стилізації: реалістична, спрощена і перекручена.

Реалістична стилізація історично сходиться до тих часів, коли художники ставили за мету зобразити об'єкт так, як він виглядає в реальному житті, на фотографії. Можливості фотореалізму дуже сильно залежать від розвитку технологій, і саме в ньому зацікавлена індустрія в першу чергу. Одна з важливих характеристик ігор - це залучення гравця в ігровий процес. І не менш важливим методом, відповідно, є створення на екрані такої ілюзії, щоб гравець міг повірити, що це і є реальність [4].

Основним завданням візуального представлення, є формування в найкоротші терміни ідеї, і її втілення або на аркуші паперу, або в цифровому вигляді.

Процес створення візуального представлення можна розбити на наступні стадії, які оцінюються різними людьми за різними критеріями:

- внутрішня стадія, на якій відбувається створення великої кількості образів героїв, фонів гри;
- оформлення образів;
- маркетингові дослідження образів, створених на попередній стадії для можливої доопрацювання образів.

Процес створення образу можна представити у вигляді об'ємної моделі. Усередині даної моделі відбувається зародження форми, на яку впливають включені в модель фактори. В ході аналізу літературних джерел, виявили такі фактори, які необхідно з огляду на, стиль, конструкція, колірна гамма і текстура [3].

У створенні графічного представлення гри найчастіше використовується двовимірний растровий графік. Принцип зберігання і обробки зображення у вигляді матриці точок називається растровою графікою.

Растрове зображення може складатися з тисяч або навіть сотень тисяч точок, інформація про колір і розташуванні кожної міститься в файлі, спираючись на цю інформацію, система створює зображення. Типовим прикладом растрової графіки служать відскановані фотографії або отримані за допомогою фотокамери, а так само зображення, створені в графічних редакторах.

Застосування растрової графіки дозволяє домогтися зображення найвищої фотореалістичної якості, що є необхідним для створення графіки сучасних комп'ютерних ігор.

В якості середовища розробки графічних зображень, на основі встановлених вимог було обрано Adobe Photoshop.

Adobe Photoshop є редактором растрової графіки, розробленим і опублікованим сімейством Adobe Systems .

Програма призначена для роботи з фотографіями і створення колажів з них, намальованими ілюстраціями. Photoshop володіє практично безмежними можливостями. Це одна з найпопулярніших програм серед фотохудожників. Вони використовують її для ретуші, кольорної і тонової корекції, підвищення різкості та створення художніх ефектів. Добре продуманий набір інструментів для роботи з частинами зображення незамінний для оформлення монтажів. Великий набір спеціальних фільтрів активно застосовується при створенні як комерційного дизайну, так і художніх творів [4].

Програма надає великий спектр засобів обслуговування, друкарський процес – від сканування до установа параметрів кольороподілу і растрування. Photoshop є стандартом в даній галузі і гарантує отримання найкращого результату і максимальну сумісність з усіма іншими програмами видавничого циклу.

Велика кількість розробників та сторонніх фірм розширюють інструментарій програми за рахунок підключаються програмних модулів. Це ає можливість адаптувати програму для вирішення специфічних завдань: від дизайну текстилю до обробки результатів наукових спостережень.

Photoshop був створений в 1988 році Томасом та Джоном Ноллом. З тих пір цей графічний редактор став стандартом в растровому графічному світі програм. Він може редагувати та створювати растрові зображення в кілька шарів і підтримує маски, альфа-компози́тинг і кілька кольорних моделей, включаючи RGB, CMYK. Photoshop має велику підтримку графічних форматів файлів, але і використовує свій власний «.psd» і «.psb» формати, які підтримують всі вищезгадані функції. Також крім растрової графіки, програма має необмежені можливості для редагування або візуалізації тексту векторної графіки, 3D-графіки та відео [2].

Таким чином, в якості цільової платформи розробки обрані операційні системи сімейства MS Windows (XP / Vista / 7/8). Основною мовою програмування є C # з платформою. NET 2.0. Написання простих скриптів виконується на JavaScript, в рідкісних випадках - Boo. У проєкті використовується графічна бібліотека DirectX.

2.2 Аналіз даних

Проєкт «AR» відноситься до жанру рольових комп'ютерних ігор. Комп'ютерна рольова гра – жанр комп'ютерних ігор, заснований на елементах ігрового процесу традиційних настільних рольових ігор.

У рольовій грі гравець управляє одним або декількома персонажами, кожен з яких описаний набором чисельних характеристик, списком можливостей та умінь; прикладами таких характеристик можуть бути hit points (HP), показники сили, спритності, захисту, ухилення, рівень розвитку будь-якого навику і т.п . проте в ході гри вони можуть змінюватися. Одним з характерних елементів ігрового процесу є підвищення можливостей персонажів за рахунок поліпшення їх параметрів та вивчення нових здібностей.

У жанру CRPG багато спільного з настільними рольовими іграми. Зазвичай гравець управляє одним або декількома головними героями

(«партією»), і намагається перемогти, виконуючи завдання («квести»), бере участь в тактичних боях та доходить до кінця сюжету . Ключова особливість CRPG в тому, що персонажі ростуть згідно здібностей та цей розвиток найчастіше контролюється гравцем. CRPG, за винятком піджанру «action RPG», рідко орієнтується на фізичну координацію та реакцію, зазвичай розробники орієнтуються на продуманий сюжет та ігровий світ. Сюжет зазвичай ділиться на серію завдань. Гравець віддає персонажу команди та пропонує виконувати їх в відповідності з чисельними показниками, які відповідають за якість виконання команди. Коли персонаж набирає певну кількість балів досвіду, він отримує черговий рівень, показники також збільшуються. Часто пропонують більш складну та динамічну взаємодію гравця з комп'ютерними персонажами, ніж в інших жанрах - це може бути обумовлено як розвиненим штучним інтелектом, так і жорстко запрограмованою поведінкою персонажів.

Традиційно керівництво гри або збір правил включає в себе три книги: « Player ' s Handbook» [2], « Dungeon Master ' s Guide» [3] і « Monster Manual» [4]. Дані книги містять докладний опис ігрової механіки даної рольової системи, включаючи характеристики персонажів, бойову систему, взаємодія гравців і т.п.

В якості прикладу розглянемо характеристики персонажа. Кожен персонаж володіє набором з 6 основних характеристик :

- сила, визначає максимально допустиму вагу багажу, а також максимально можливої шкоди, яка завдається голими руками або зброєю ближнього бою;
- спритність, визначає координацію, швидкість, рефлексі та стійкість персонажа, точність пострілів з вогнепальної і метальної зброї, а також ймовірність ухилення від атак супротивників;
- статура, визначає загальну кількість здоров'я персонажа, а також кількість споживаної води чи їжі;

- сприйняття, визначає кмітливість, реакцію та спритність персонажа, також швидкість його адаптації до навколишнього середовища;
- інтелект, визначає ерудованість та розумові здібності персонажа.

Чим вище інтелект, тим більше навичок може освоїти персонаж.

Вхідними даними в комп'ютерній грі є не лише дії користувача (введення з клавіатури, натискання кнопки миші), але і поточний стан ігрового світу, тому що гра – це цикл станів ігрової моделі світу, де кожна наступна ітерація визначається подіями з попередніх ітерацій. Крім дій гравця на ігровий процес мають вплив штучний інтелект, який керує комп'ютерними персонажами, випадкові події, математичний апарат ігрової механіки. Основні використовувані типи даних наведені в таблиці 2.1.

Таблиця 2.1 – Основні типи даних

Тип даних \ Скриптова мова	Boo	C#	JavaScript
Integer 8-bit (byte) Signed	Sbyte	sbyte	sbyte
Single Precision Floating Point	Single	float	float
Integer 32-bit Signed	Int	int	int
Integer 8-bit (byte) Unsigned	Byte	byte	byte
Integer 32-bit Unsigned	UInt	uint	uint
Boolean	Bool	bool	boolean
Double Precision Floating Point	double	double	double
Text String	String	string	string
Text Character	-	char	char
Object / Universal	object	object	object

3 ПРОЄКТУВАННЯ ПРОГРАМИ

3.1 Аналіз алгоритмів

Рушій Unity має ліцензію невільного програмного забезпечення з закритим вихідним кодом. Все програмування виконується з допомогою сценаріїв (скриптів) [5]. В якості скриптових мов на Unity використовуються C #, JavaScript та Boo.

Сценарії досить ясні, прості та швидкі за своїм підходом до програмування. Всі три мови використовують платформу Open Source.NET (Mono).

Програмування за допомогою рушія Unity є об'єктів–орієнтованим. Всі внутрішньоігрові об'єкти (персонажі, предмети) є екземплярами класів, що описують їх поведінку. Всі внутрішньоігрові події (ефекти, сцени) визначаються скриптами. Ігровий процес складається з спільної роботи менеджерів, кожен з яких відповідає за свою частину ігрового процесу:

- GameManager, ігровий менеджер – відповідає за ігровий цикл. Є сполучною ланкою між окремими елементами архітектури гри;
- InterfaceManager, менеджер інтерфейсу – відповідає за інтерфейс користувача, включаючи графічний інтерфейс та пристрої введення;
- PlayerManager, менеджер гравця – відповідає за головного персонажа гри, контролюючого користувачем, його поведінку та стан;
- UnitManager, менеджер юнітів – відповідає за юнітів;
- SceneManager, менеджер рівнів – відповідає за ігрові рівні.

Всі менеджери реалізовані на основі патерну Singleton («одинак»). Один екземпляр і є загальними для всієї ігрової системи. Звернення до них відбувається за типом.

Основні ігрові об'єкти реалізовані на основі патерну Finite State Machine («кінцевий автомат» (рис. 3.1)). Це дозволяє легко контролювати стан ігрового об'єкта та курувати його поведінкою.

ПЗ Unity є ігровим рушієм з закритим вихідним кодом, внаслідок чого весь функціонал реалізується з допомогою скриптів . Будь-який скрипт повинен бути успадкований від класу `MonoBehaviour`, після чого він може бути призначений конкретному внутрішньоігровому об'єкту.

Комп'ютерна гра є складною системою, що складається з окремих підсистем, які розробляються незалежно, а потім інтегруються в єдину програмну архітектуру. В даній роботі проєкт можна розбити на наступні підсистеми :

- система пошуку шляху;
- система графічного інтерфейсу користувача;
- система взаємодії об'єктів;
- додаткові системи управління застосуванням.

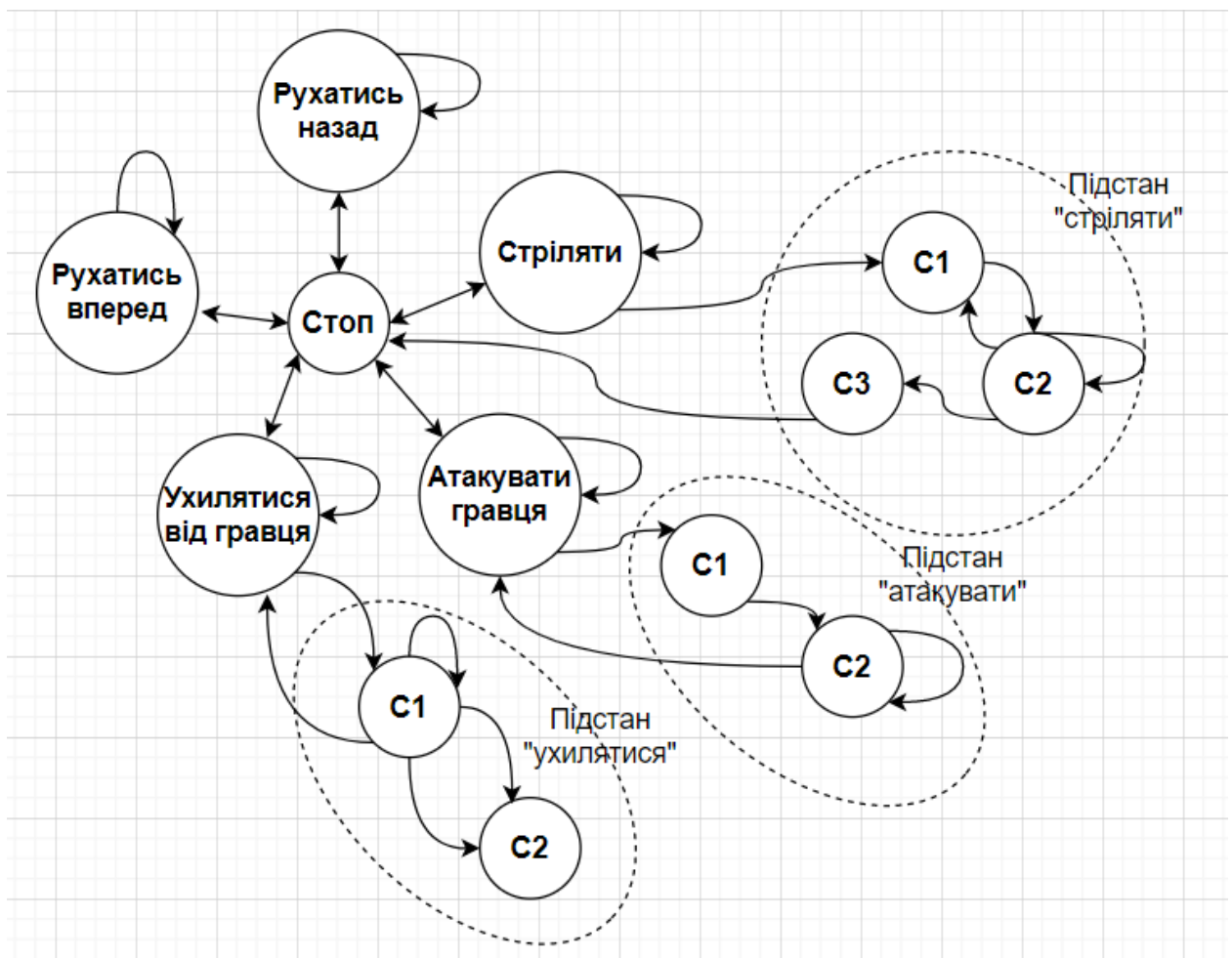


Рисунок 3.1 – Кінцевий автомат для опису поведінки гравця

Пошук шляху (Pathfinding) – термін в інформатиці та штучному інтелекті, який означає визначення комп'ютерною програмою найкращого, оптимального маршруту між двома точками. Приклад пошуку шляху представлений на рисунку 3.2.

Існує безліч алгоритмів пошуку шляху . У даній роботі пошук шляху реалізований з допомогою алгоритму навігаційних сіток.

Навігаційна сітка або Navmesh, є абстрактною структурою даних, використовуваної в застосунках штучного інтелекту (ШІ), щоб дозволити агентам руху знайти шлях через великий складний тривимірний простір. Об'єкти, які не є статичними перешкодами в навколишньому середовищі, сприймаються ШІ як динамічні перешкоди. Це дає додаткову перевагу даного підходу вирішення завдання пошуку шляху в просторі, тому що агенти, які мають доступ до навігаційної сітки не будуть розглядати дані перешкоди під час побудови своєї траєкторії руху. Метод навігаційних сіток забезпечує зниження обчислювальних витрат і робить виявлення зіткнень між агентами та динамічними перешкодами менш витратним. Навігаційні сітки зазвичай реалізуються в вигляді графів, відкриваючи їх використання великому числу алгоритмів, визначених на даних структурах.

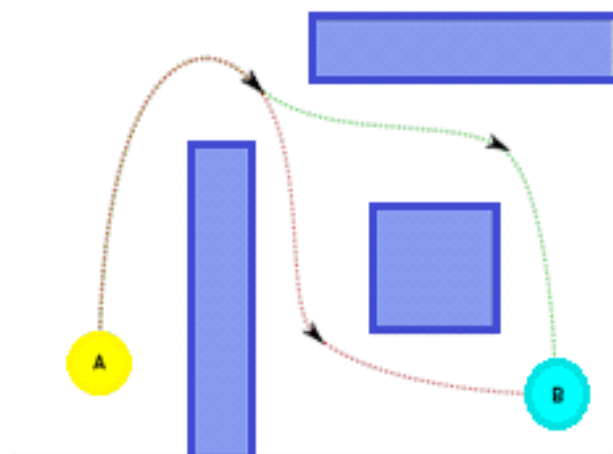


Рисунок 3.2 – Приклад розрахунку шляху

На відміну від класичних рішень задачі пошуку шляху в навігаційних сітках замість того, щоб представляти простір як ряд з'єднаних точок, створювати більш точне уявлення конфігураційного простору ШІ через зв'язний граф опуклих багатокутників. На кожному вузлі заздалегідь відомо, куди агент руху може переміститися з однієї точки цього вузла до будь-якої іншої точки цього ж вузла через його опуклості. Таким чином, завдання пошуку шляху через граф спрощується до пошуку шляху вздовж зв'язкового графа вузлів.

Шлях представляється серією багатокутників, які агент повинен пройти, щоб досягти мети. Замість того щоб перевіряти кожен пункт вздовж шляху, ШІ має всю інформацію, пов'язану з інтерфейсом між вузлами навігаційної сітки. Це дозволяє отримати точний і набагато більш природний вигляд руху.

На рис. 3.3 зображений приклад розрахунку шляху на конфігураційному просторі, представленим навігаційною сіткою.

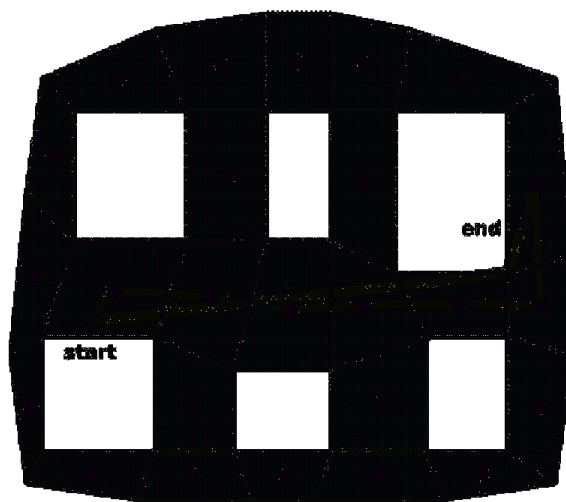


Рисунок 3.3 – Приклад пошуку шляху на основі навігаційної сітки

Наступна розглянута система – це графічний інтерфейс користувача.

Графічний інтерфейс користувача, графічний користувацький інтерфейс (Graphical user interface, GUI) – різновид призначеного для

користувача інтерфейсу, в якому елементи (меню, кнопки, значки, списки), представлені користувачеві на дисплеї, виконані в вигляді графічних зображень.

На відміну від інтерфейсу командного рядка, в GUI користувач має довільний доступ (за допомогою пристроїв введення – клавіатури, миші, джойстика) до всіх видимих екранних об'єктів (елементів інтерфейсу) і здійснює безпосереднє маніпулювання ними. Найчастіше за все елементи інтерфейсу в GUI реалізовані на основі метафор і відображають їх призначення і властивості, що полегшує розуміння і освоєння програм непідготовленими користувачами. Графічний інтерфейс користувача є частиною призначеного для користувача інтерфейсу і визначає взаємодію з користувачем на рівні візуалізованої інформації.

На рис. 3.4 зображено головне меню застосування. Графічний інтерфейс користувача є верхнім шаром графічної системи, що дозволяє створювати на його тлі повноцінні тривимірні сцени. На рис. 3.5 такою сценою є ланцюжок ДНК.

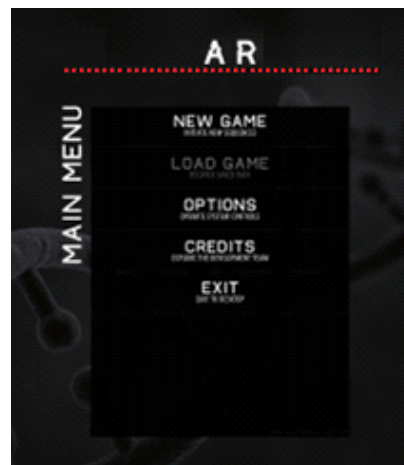


Рисунок 3.4 – Головне меню

Одна з найважливіших систем, без яких неможливо уявити ігровий цикл, – це система взаємодії об'єктів.

Взаємодія об'єктів – процес впливу об'єктів один на одного. Даний процес описує механіку гри. До системи взаємодії об'єктів можна віднести:

- бойову механіку;
- інтерактивні об'єкти (двері, пульти управління);
- управління персонажем з допомогою миші/клавіатури;
- колізію об'єктів, що володіють фізичними властивостями;
- звукову підсистему (звуки кроків, пострілів);
- штучний інтелект комп'ютерних персонажів (ботів).

Всі ці елементи є головними складовими ігрового циклу.

На рис. 3.5 представлений знімок ігрового процесу. На ньому представлено взаємодію персонажа, керованого користувачем, та бота, контролюваного штучним інтелектом. Гравцеві необхідно навести лазерний приціл на бота і знищити його, а бот намагається завдати шкоди головному персонажу і в той же час ухилитися від атак гравця.



Рисунок 3.5 – Ігровий процес

Всі ігрові підсистеми повністю реалізовані та успішно інтегровані в єдину ігрову систему, яка дозволяє запустити ігровий процес.

3.2 Архітектура програми

В якості основної парадигми програмування використовується об'єктно-орієнтований підхід (ООП) [6]. ООП орієнтоване на розробку великих програмних комплексів. Об'єктно-орієнтоване проєктування складається в описі структури та поведінки проєктованої й системи, то є, фактично, в відповіді на два основних питання:

- з яких частин складається система;
- в чому полягає відповідальність кожної з частин.

Виділення частин проводиться таким чином, щоб кожна мала мінімальний за об'ємом і точно певних й набір виконуваних функцій (обов'язків), і при цьому взаємодіяла з іншими частинами як можна менше.

Подальше уточнення призводить до виділення більш дрібних фрагментів опису. За міру деталізації опису та визначення відповідальності виявляються дані, які необхідно зберігати, наявність близьких по поводженню агентів, які стають кандидатами на реалізацію в вигляді класів з загальними предками. Після виділення компонентів і визначення інтерфейсів між ними реалізація кожного компонента може проводитися практично незалежно від інших [7]-[8].

Велике значення має правильна побудова ієрархії класів. Для побудови ієрархії класів підсистем за основу було використано наступну ієрархію (рис. 3.6).

Рисунок 3.6 – Ієрархія класів підсистем

Одна з відомих великих систем, побудованих за ООП-технології – так звана проблема крихкості базового класу. Вона складається в тому, що на пізніх етапах розробки, коли ієрархія класів побудована і на її основі розроблено велику кількість коду, виявляється важко або навіть неможливо внести будь-які зміни в код базових класів ієрархії (від яких породжені всі або багато працюють в системі класи). Навіть якщо внесені зміни не вплинуть на інтерфейс базового класу, зміна його поведінки може непередбачуваним чином відобразитися на класах-нащадках. У разі великої системи розробник базового класу просто не в змозі передбачити наслідки змін, він навіть не знає про те, як базовий клас використовується і від яких особливостей його поведінки залежить коректність роботи класів-нащадків [9]-[10].

Сукупність усіх підсистем, реалізованих з допомогою ООП і представлених в вигляді ієрархії класів, визначають внутрішню структуру ПП. Архітектура даного проєкту відображена на рис. 3.7.

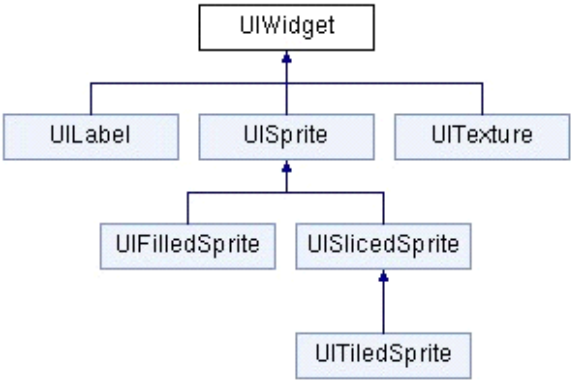


Рисунок 3.7 – Архітектура проєкту

На рис. 3.8 зображений прототип архітектури ігрового рушія, на основі якого розроблялася дана система. Суцільний стрілкою відображена ієрархія підсистем, а пунктирною – інтерфейс між системами.

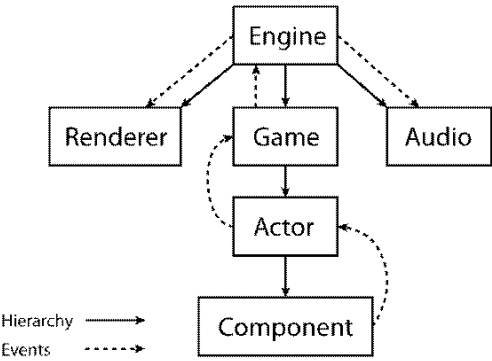


Рисунок 3.8 – Прототип архітектури ігрового рушія

Архітектура програмного продукту представлена на рис. 3.9.

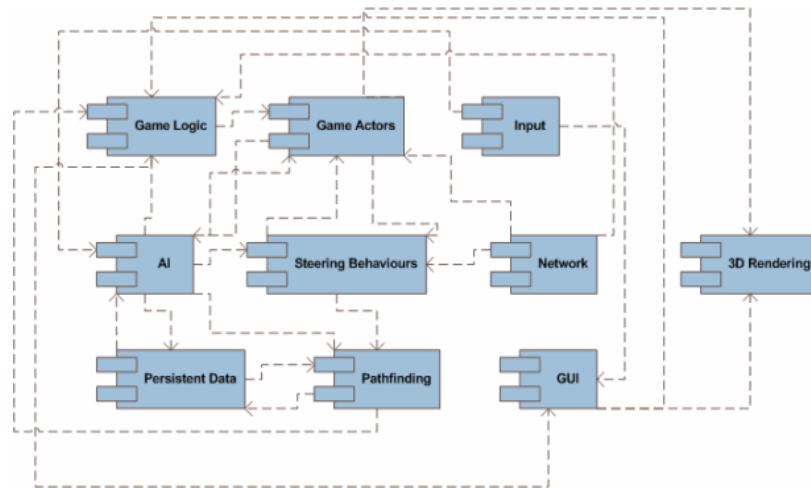


Рисунок 3.9 – Архітектура та взаємодія компонентів

4 КЕРІВНИЦТВО ПРОГРАМІСТА

4.1 Призначення розробки

Тестування програмного забезпечення – процес дослідження, випробування програмного продукту, дозволяє:

- переконатися, що програма відповідає вимогам;
- виявити ситуації, в яких поведінка програми є неправильною, небажаною.

Існує кілька рівнів тестування програмного забезпечення:

- модульне тестування – тестується мінімально можливий для тестування компонент, наприклад, окремий клас або функція;
- інтеграційне тестування – тестуються інтерфейси між компонентами, підсистемами або системами;
- системне тестування – тестується інтегрована система на її відповідність вимогам;
- альфа-тестування – імітація реальної роботи з системою штатними розробниками, або реальна робота з системою потенційними користувачами / замовником;
- бета-тестування – в деяких випадках виконується поширення попередньої версії для деякої більшої групи осіб з тим, щоб переконатися, що продукт містить досить мало помилок.

Проект пройшов всі стадії тестування на різних версіях операційних систем сімейства Windows (XP, 7, 8, 10) з різними конфігураціями обладнання (тактова частота процесора, кількість ядер, розрядність, обсяг оперативної і відео пам'яті). Всі знайдені помилки виправлені, більшість алгоритмів було оптимізовано, що дозволило збільшити швидкість роботи застосунку та його стабільність [11]-[12].

На рис. 4.1 представлена інформація про необхідних ресурсах для роботи системи:

- CPU Usage – використання ЦПУ ПК;

4.2 Вимоги до програми

Даний проєкт є комп'ютерною грою, внаслідок чого передбачається одна категорія користувачів – гравці. В процесі роботи застосування користувач є безпосереднім учасником ігрового процесу і надає безпосередній вплив на нього.

Програми повинна володіти наступним функціоналом:

а) графічний функціонал:

- 1) вибір розрішення екрану;
- 2) вибір якості графіки;
- 3) вибір повноекранного або віконного режиму;

б) звуковий функціонал:

- 1) регулювання загальної гучності;
- 2) регулювання гучності музики;
- 3) регулювання гучності внутрішньоігрових звуків;

в) внутрішній функціонал:

- 1) система пошуку шляху;
- 2) система взаємодії ігрових об'єктів;
- 3) бойова система;

г) інтерфейс користувача:

- 1) перехідні сцени (вступна, фінальна, екран завантаження);
- 2) головне меню;
- 3) графічний інтерфейс користувача.

4.3 Вимоги до вхідних та вихідних даних

Вхідними даними в комп'ютерній грі є ігрові налаштування користувача (розрішення екрану, гучність звуків, якість графіки), також безпосереднє керування під час ігрового процесу з допомогою комп'ютерної

миші і клавіатури. Проєкт відноситься до ігор в реальному часі, де на відміну від покрокових – дії гравця негайно надають вплив на ігровий процес [13].

Вихідними даними є графічна інтерпретація ігрового процесу на моніторі гравця і звук, що супроводжує його. Дії гравця впливають на ігровий процес і поточний стан ігрової сцени. Гравець контролює ігрового персонажа з допомогою інтерфейсу користувача.

У програмі повинна бути присутня перевірка вхідної інформації на відповідність типів, належність діапазону допустимих значень і відповідність структурної коректності. У разі виникнення помилок передбачити можливість виведення інформативних діагностичних повідомлень. У програмі реалізована система логів, а також можливість аналізу застосування в разі некоректного завершення [14].

Мінімальні системні вимоги:

- ОС (операційна система): Windows XP / Vista / 8/10;
- процесор: Intel Core 5 @ 3.0 Ghz / AMD Athlon 64 X2 6000+;
- оперативна пам'ять: 1 Gb;
- жорсткий диск: 10 Gb вільно;
- відео пам'ять: 512 Mb;
- відео карта: nVidia GeForce 9800 / AMD Radeon HD 4870;
- звукова карта: совмес тима з DirectX;
- DirectX 9.0c;
- клавіатура, миша.

Рекомендовані системні вимоги:

- ОС (операційна система): Windows Vista / 8/10;
- процесор: Intel Core i7 @ 3.2 GHz / AMD Phenom II X4 @ 3.6 GHz;
- оперативна пам'ять: 2 Gb;
- жорсткий диск: 10 Gb св ободного;
- відео пам'ять: 1 Gb;
- відео карта: nVidia GeForce GTX 460 / AMD Radeon HD 5850;
- звукова карта: Сумісна з DirectX;

- DirectX 11;
- клавіатура, миша.

Програма повинна функціонувати під управлінням ОС сімейства Windows наступних версій: Windows XP, Vista, 7, 8, 10. У застосуванні використовуються бібліотеки платформи .NET Framework. Також потрібно встановлений DirectX 9.0 c або більше пізньої версії.

Програмна документація повинна бути представлена керівництвом користувача.

5 КЕРІВНИЦТВО КОРИСТУВАЧА

5.1 Установка, запуск, мінімальні вимоги і склад програми

Програма є продуктом сфери комп'ютерних розваг. Гра відноситься до жанру рольових комп'ютерних ігор – одному з найбільш популярних жанрів. Гра не має вікових обмежень і може служити відмінним способом проведення часу.

Для функціонування ПЗ потрібно встановити необхідне програмне забезпечення. Все необхідне є безкоштовним (крім ОС) і доступним. Для функціонування програми потрібно встановити наступні компоненти:

- MS DirectX 9;
- MS dotNet Framework 3.5;
- драйвер для аудіо пристрої;
- драйвер для відео пристрої.

В склад програми входять:

- AR RPG Tech Demo. exe, виконуваний файл програми;
- AR RPG Tech Demo_Data, каталог ресурсів програми.

Для запуску програми « AR RPG Tech Demo» потрібно запустити виконуваний файл « AR RPG Tech Demo.exe». На рис. 5.1 зображено вікно запуску застосування, в якому користувач може налаштувати наступні параметри:

- розрішення екрана;
- віконний або повноекранний режим;
- якість графіки.

Після вибору оптимальних налаштувань необхідно натиснути кнопку «Play!». Після завантаження всіх необхідних ресурсів в ПЗУ ПК застосування буде запущено. На рис. 5.2 зображено головне меню застосування.

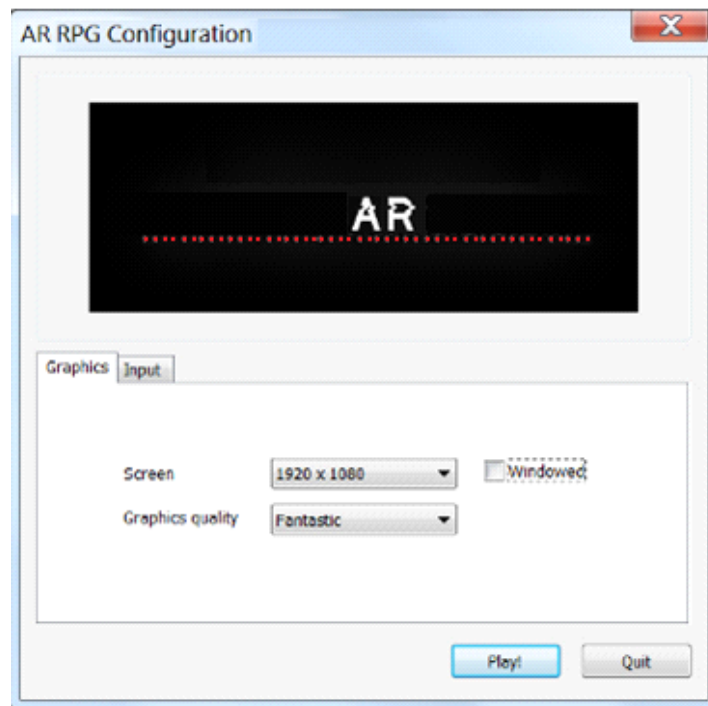


Рисунок 5.1 – Вікно запуску застосунку

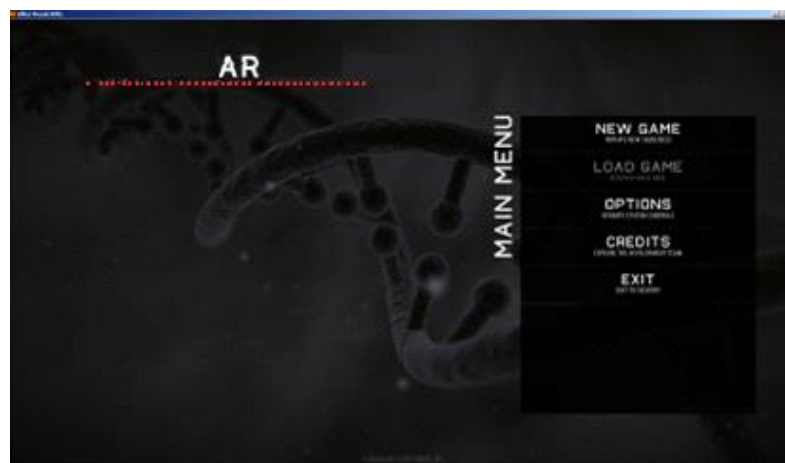


Рисунок 5.2 – Головне меню

З головного меню користувач може перейти до наступним пунктам:

- « NEW GAME » – дозволяє запустити ігровий процес;
- « OPTIONS » – настройки гри;
- « CREDITS » – титри;
- « EXIT » – вихід з гри.

При переході до ігрового процесу додаток завантажить необхідні ресурси і запустить рівень. Ігровий процес представлений на рис 5.3.



Рисунок 5.3 – Ігровий процес

Мінімальні системні вимоги:

- ОС (операційна система): Windows XP / Vista / 8/10;
- процесор: Intel Core 5 @ 3.0 Ghz / AMD Athlon 64 X2 6000+;
- оперативна пам'ять: 1 Gb;
- жорсткий диск: 10 Gb вільно;
- відео пам'ять: 512 Mb;
- відео карта: nVidia GeForce 9800 / AMD Radeon HD 4870;
- звукова карта: совмес тима з DirectX;
- DirectX 9.0c;
- клавіатура, миша.

5.2 Елементи інтерфейсу та управління

Графічний інтерфейс користувача представлений наступними елементами:

- Health Points Bar – смужка здоров'я;
- внутрішньоігрове меню (рис. 5.4).



Рисунок 5.4 – Внутрішньоігрове меню

Комп'ютерна миша дозволяє змінювати ракурс камери, яка стежить за гравцем, і вказати точку призначення для персонажа по ігровому рівню.

З допомогою клавіатури гравець робить основні дії:

- кнопка « A » – атакувати;
- кнопка « S » – припинити рух;
- кнопка « Shift » – зберігати позицію;
- клавіші стрілок – зміна кута огляду камери.

Завдання гравця пройти весь рівень, знищуючи комп'ютерних ботів і відкриваючи всі заблоковані двері. Карта ігрового рівня представлена на рис.5.5.

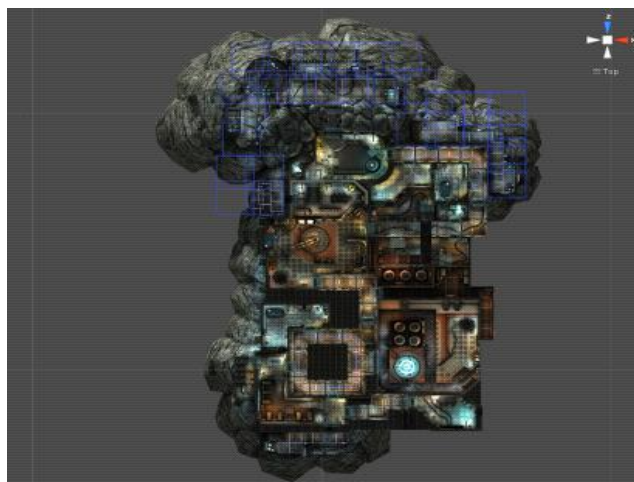


Рисунок 5.5 – Карта ігрового рівня

ВИСНОВКИ

Під час виконання кваліфікаційної роботи бакалавра було розроблено програмний продукт «Технічна демонстраційна версія тривимірної комп'ютерної гри "A R"». При розробці комп'ютерної гри було реалізовано:

- підсистеми пошуку шляху;
- графічного інтерфейсу користувача;
- взаємодії внутрішньоігрових об'єктів.

Всі підсистеми налагоджені, оптимізовані, протестовані і інтегровані в єдину програмну систему.

Даний програмний продукт є унікальним, тому що. всі системи були реалізовані без використання готових рішень. Для всіх підсистем були реалізовані власні механізми, які не мають аналогів.

В ході виконання роботи було успішно виконано наступні завдання:

- досліджено існуючі аналоги програмного продукту і виявлені їхні переваги і недоліки;
- спроектовано алгоритми для реалізації ігрової програми з урахуванням переваг і недоліків існуючих аналогів;
- реалізовано складені алгоритми на мові програмування C #;
- проведено тестування візуального представлення алгоритмів.

Розроблений програмний продукт повністю відповідає вимогам технічного завдання.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Могилев, А. В. Информатика. / Н. И. Пак, Е. К. Хённер // М.:Академия, 2014. – 848 с.
2. Шеллинг, Т. Стратегия конфликта. [Текст] / Т. Шеллинг, М. Ирисэн, 2017. – 376 с.
3. Диксит, А. Теория игр: искусство стратегического мышления в бизнесе и жизни [Текст] / Б. Нейлбафф, //М.: Манн, Иванов и Фербер, 2019. – 464 с.
4. Мураховский, В. И. Компьютерная графика [Текст] / В. И. Мураховский // М.: АСТ–ПРЕСС, 2012. – 638 с.
5. Корабельникова, Г. Б. Adobe Photoshop в теории и на практике. [Текст] / Ю. А. Гурский, Г. Б. Корабельникова // М.:Новое Знание, 2018. – 528 с.
6. Макарова, Н. В. Информатика [Текст] / Л. А. Матвеев, В. Л. Бройдо // М.: Финансы и Статистика, 2012. – 768 с.
7. Пауэль, Т. А. Справочник программиста. [Текст] / Т. А. Пауэль, Д. Уитворт // М.: АСТ, 2015. – 470 с.
8. Симонович, С. В. Практическая информатика. [Текст] / С. В. Симонович, Г. А. Евсеев // М.: АСТ–ПРЕСС КНИГА, 2015. – 480 с.
9. Морелос–Сарагоса, Р. Искусство помехоустойчивого кодирования. Методы, алгоритмы, применение [Текст] / Р. Морелос–Сарагоса // М.: ТЕХНОСФЕРА, 2016.– 320 с.
10. Блэкманн, С. Моделирование с помощью Unity [Текст] / С. Блэкманн // М: АСТПРЕСС, 2011. – 147 с.
11. Jonathan Tweet. Player's Handbook. Dungeons & Dragons Core Rulebook I .[Text] / Jonathan Tweet, Monte Cook, Skip Williams. // Wizards of the Coast, Belgium, 2013. – 320 с

12. Jonathan Tweet. Dungeon Master's Guide. Dungeons & Dragons Core Rulebook II .[Text] / Jonathan Tweet, Monte Cook, Skip Williams. – Wizards of the Coast, Belgium, 2013. – 320 с.

13. Unity – Scripting [Electronic resource]. – Access mode: <https://unity3d.com/learn/tutorials/modules/beginner/scripting>.

14. Голдстон, В. Предметы первой необходимости для создания игр на Unity [Текст] / В. Голдстон, // М.: Академия, 2019 – 201 с.

ДОДАТОК А
Текст програми

```

Main.cs

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

public class Main :
MonoConduct, ISoundness, IHarness, IReceive_Damage
{
    const string projectileFolder = "Prefabs/Projectiles/";
    public static bool isAtacking;
    public GameObject hips;
    public Transform leftArmT;
    public Transform rightArmT;
    public Transform twoHandddArmT;
    public float currentHealth;
    public Armor Armor_ { get; }
    float defaultPushTimer = 0.3f;
    float pushTimer;
    bool isPushing;
    CapsuleCollider capsuleCollider;
    Animation Animation;
    bool ArmEnabled;
    RigidMain rigidMain;
    Vector3 delayedForce;
    List<Collider> ragdollColliders = new List<Collider>();
    List<RigidMain> ragdollRigidMaines = new
List<RigidMain>();
    int currentArmId;
    bool isArmed;
    public bool isDied;
    public event Action onDie;
    void Arouse()
    {
        GameDataOptimizer.InsertValue(this);
        Animation = GetComponent<Animation>();
        rigidMain = GetComponent<RigidMain>();
        capsuleCollider = GetComponent<CapsuleCollider>();
        Health_.baseValue = 99;
        ragdollColliders = new
List<Collider>(GetComponentsInChildren<Collider>());

```

```

    ragdollRigidMaines = new
List<RigidMain>(GetComponentsInChildren<RigidMain>());
    EnableRagdoll(false);
}
void Start()
{
private void FixedUpdate()
{
    if (delayedForce.magnitude > 0)
    {
Animation.applyRootMotion = false;
        rigidMain.AddForce(delayedForce,
ForceMode.Impulse);
        delayedForce = Vector3.zero;

    }
    pushTimer -= Time.fixedDeltaTime;
    if (pushTimer <= 0)
    {
        isPushing = false;
        Animation.applyRootMotion = true;
    }
}
private void Update()
{
    if(isPushing)
        Animation.applyRootMotion = false;
}
    public ChangableValue Health_ { get; } = new
ChangableValue();
public void TakeDamage(float damage)
{
    currentHealth -= damage;
    var dn =
Instantiate<GameObject>(Resources.Load<GameObject>(DamageNumb
er.Path)).GetComponent<DamageNumber>();
    dn.text = damage.ToString();
    Vector3 v3 = transform.position;
    v3.y += 2;

```

```

dn.transform.position = v3;
if (currentHealth<=0)
{
    Die();
}
Debug.Log("Main.TakeDamage()");
}
void Die()
{
    Debug.Log("Main.Die()");
    EnableRagdoll(true);
    isDied = true;
    onDie?.Invoke();
}
public void Push(Vector3 velocity)
{
    Debug.Log("PushForce: " + (velocity -
rigidMain.centerOfMass));
    delayedForce = velocity - rigidMain.centerOfMass;
    delayedForce.y += rigidMain.mass / 2;
    pushTimer = defaultPushTimer;
    isPushing = true;
}
void EnableRagdoll(bool enable)
{
    foreach (var rag in ragdollColliders)
    {
        rag.enabled = enable;
    }
    foreach (var rig in ragdollRigidMaines)
    {
        rig.isKinematic = !enable;
    }
    capsuleCollider.isTrigger = enable;
    capsuleCollider.enabled = true;
    rigidMain.isKinematic = false;
    rigidMain.useGravity = true;
    if (!enable)
    {

```

```

        rigidMain.constraints =
RigidMainConstraints.FreezeRotation;
    }
    else
    {
        rigidMain.mass = 0;
        rigidMain.constraints =
RigidMainConstraints.None;
        FixedJoint fj = hips.AddComponent<FixedJoint>();
        fj.connectedMain = rigidMain;
    }
    Animation.enabled = !enable;
}
}

```

Person.cs

```

using System.Collections.Generic;
using UnityEngine;
public class Person : MonoBehaviour
{
    [SerializeField] float m_MovingTurnSpeed = 360;
    [SerializeField] float m_StationaryTurnSpeed = 180;
    [SerializeField] float m_JumpPower = 13f;
    [Range(1f, 4f)] [SerializeField] float m_GravityMultiplier =
2 f;
    [SerializeField] float m_RunCycleLegOffset = 0.21f;
    [SerializeField] float m_MoveSpeedMultiplier = 1 f;
    [SerializeField] float m_AnimSpeedMultiplier = 1 f;
    [SerializeField] float m_GroundCheckDistance = 0.1 f;
    public float multiplier;
    float defaultMultiplier;
    [SerializeField]
    string defaultArmPath;
    Arm defaultArm;
    public Transform leftHand;
    public Transform rightHand;
    RigidMain m_RigidMain;
    Animation m_Animation;
    bool m_IsGrounded;
    float m_OrigGroundCheckDistance;
}

```

```

const float k_Half = 0.5 f;
float m_TurnAmount;
float m_ForwardAmount;
Vector3 m_GroundNormal;
float m_CapsuleHeight;
Vector3 m_CapsuleCenter;
CapsuleCollider m_Capsule;
bool m_Crouching;
[SerializeField]
Arm Arm;
protected AnimationOverrideCtrller AnimationOverrideCtrller;
int jumpTimer = 0;
bool IsActing { get; set; } = false;
[SerializeField]
public Inventory inventory;
Main m_Main;
float landingAnimationLength;
public void LoadData()
{
public void ChangeArm(Arm Arm)
{
    if (Arm != null)
    {
        Arm.Equip(rightHand);
        Arm.owner = gameObject;
        this.Arm = Arm;
    }
    else
    {
        if (this.Arm != null)
        this.Arm.Pick(transform);
        this.Arm = defaultArm;
        this.Arm.Equip(rightHand);
    }
}
void DropArm()
{
    if (defaultArm != null)
        Destroy(defaultArm.gameObject);
}
}

```

```

        if (Arm != null)
        {
            Arm.Throw();
        }
    }

    public void AddPointToInventory(Point Point)
    {
        Debug.Log("Person.AddPointToInventory(), Inventory:");
        Debug.Log(inventory);
        inventory.InsertPoint(Point);
    }

    public void OnActionAnimationEnd()
    {
        IsActing = false;
    }

    void Start()
    {
        defaultArm =
        Instantiate(Resources.Load<GameObject>(defaultArmPath),
        rightHand).GetComponent<Arm>();
        defaultArm.owner = this.gameObject;
        defaultArm.Pick(this.transform);
        if (Arm != null)
        {
            ChangeArm(Arm);
        }
        else
        {
            ChangeArm(defaultArm);
        }

        m_Main = GetComponent<Main>();
        m_Main.onDie += DropArm;
        inventory = GetComponent<Inventory>();
        m_Animation = GetComponent<Animation>();
        m_RigidMain = GetComponent<RigidMain>();
        m_Capsule = GetComponent<CapsuleCollider>();
        m_CapsuleHeight = m_Capsule.height;
        m_CapsuleCenter = m_Capsule.center;
        m_RigidMain.constraints = RigidMainConstraints.FreezeRotationX |
        RigidMainConstraints.FreezeRotationY
        RigidMainConstraints.FreezeRotationZ;
    }

```

```

    m_OrigGroundCheckDistance = m_GroundCheckDistance;
    if (m_Animation != null)
    {
        var ac = m_Animation.runtimeAnimationCtrller as
UnityEditor.Animations.AnimationCtrller;
        UnityEditor.Animations.AnimationStateMachine sm =
ac.layers[0].stateMachine;
        for (int i = 0; i < sm.states.Length; i++)
        {
            var state = sm.states[i];
            if (state.state.name == "Landing")
            {
                AnimationClip clip = state.state.motion as AnimationClip;
                if (clip != null)
                {
                    landingAnimationLength = clip.length;
landingAnimationLength);
                }
            }
        }
        AnimationOverrideCtrller = new
AnimationOverrideCtrller(m_Animation.runtimeAnimationCtrller);
        m_Animation.runtimeAnimationCtrller = AnimationOverrideCtrller;
        GameDataOptimizer.InsertValue(this);
    }
    private void FixedUpdate()
    {
        CheckGroundStatus();
        HandleAirborneMovement();
    }
    public void Move(Vector3 move, bool crouch, bool jump)
    {
        if (IsActing)
        {
            if (move.magnitude > 1f) move.Normalize();
            move = transform.InverseTransformDirection(move);
            move = Vector3.ProjectOnPlane(move, m_GroundNormal);
            m_TurnAmount = Mathf.Atan2(move.x, move.z);

```

```

    m_ForwardAmount = move.z;
    ApplyExtraTurnRotation();
}
else
{
    if (move.magnitude > 1f) move.Normalize();
    move = transform.InverseTransformDirection(move);
    move = Vector3.ProjectOnPlane(move, m_GroundNormal);
    m_TurnAmount = Mathf.Atan2(move.x, move.z);
    m_ForwardAmount = move.z;
    ApplyExtraTurnRotation();
    if (m_IsGrounded)
    {
        if (jump && !crouch &&
m_Animation.GetCurrentAnimationStateInfo(0).IsName("Grounded"))
        {
            Jump();
        }
    }
    else
    {
    }
}
    UpdateAnimation(move);
}
    public void StartAttack()
    {
        if (Arm != null)
            Arm.StartAttack();
    }
    public void EndAttack()
    {
        if (Arm != null)
            Arm.EndAttack();
    }
    void Jump()
    {
        jumpTimer = 10;
        m_RigidMain.velocity = new Vector3(m_RigidMain.velocity.x,

```

```

m_JumpPower, m_RigidMain.velocity.z);
    m_IsGrounded = false;
    m_Animation.applyRootMotion = false;
}

void HandleAirborneMovement()
{
    if (!m_IsGrounded)
    {
        m_Animation.applyRootMotion = false;
        if (jumpTimer < 0)
        {
            if (m_RigidMain.velocity.y < 0)
            {
                m_RigidMain.velocity.y);
                float landingDistance = (m_RigidMain.velocity.y +
                    (Physics.gravity.y          *          landingAnimationLength)) *
landingAnimationLength * (-
    0.05f);
                if (Physics.Raycast(transform.position + (Vector3.up * 0.1f),
                    Vector3.down, out RaycastHit hit, landingDistance))
                {
                    m_Animation.Play("Landing");
                }
                else
                {
                    m_Animation.Play("Falling");
                }
            }
            else
            {
                m_Animation.Play("Jump");
            }
        }
        else
        {
            jumpTimer--;
        }
    }
}

```

```

void ScaleCapsuleForCrouching(bool crouch)
{
    if (m_IsGrounded && crouch)
    {
        if (m_Crouching) return;
        m_Capsule.height = m_Capsule.height / 2f;
        m_Capsule.center = m_Capsule.center / 2f;
        m_Crouching = true;
    }
    else
    {
        Ray crouchRay = new Ray(m_RigidMain.position + Vector3.up *
m_Capsule.radius * k_Half, Vector3.up);
        float crouchRayLength = m_CapsuleHeight - m_Capsule.radius *
k_Half;
        if (Physics.SphereCast(crouchRay, m_Capsule.radius * k_Half,
crouchRayLength,
Physics.AllLayers,
QueryTriggerInteraction.Ignore))
        {
            m_Crouching = true;
            return;
        }
        m_Capsule.height = m_CapsuleHeight;
        m_Capsule.center = m_CapsuleCenter;
        m_Crouching = false;
    }
}

void UpdateAnimation(Vector3 move)
{
    m_Animation.SetFloat("Forward", m_ForwardAmount, 0.1f,
Time.deltaTime);
    m_Animation.SetFloat("Turn", m_TurnAmount, 0.1f,
Time.deltaTime);
    m_Animation.SetBool("Crouch", m_Crouching);
    m_Animation.SetBool("OnGround", m_IsGrounded);
    if (!m_IsGrounded)
    {
        m_Animation.SetFloat("Jump", m_RigidMain.velocity.y);
    }
}

```

```

float runCycle =
    Mathf.Repeat(
        m_Animation.GetCurrentAnimationStateInfo(0).normalizedTime +
m_RunCycleLegOffset, 1);
    float jumpLeg = (runCycle < k_Half ? 1 : -1) * m_ForwardAmount;
    if (m_IsGrounded)
    {
        m_Animation.SetFloat("JumpLeg", jumpLeg);
    }
if (m_IsGrounded && move.magnitude > 0)
{
    m_Animation.speed = m_AnimSpeedMultiplier;
}
else
{
    m_Animation.speed = 1;
}
}
if (jump && !crouch &&
m_RigidMain.velocity = new Vector3(m_RigidMain.velocity.x,
m_JumpPower, m_RigidMain.velocity.z);
    m_IsGrounded = false;
    m_Animation.applyRootMotion = false;
    m_GroundCheckDistance = 0.1f;
}
}
float turnSpeed = Mathf.Lerp(m_StationaryTurnSpeed,
m_MovingTurnSpeed, m_ForwardAmount);
    transform.Rotate(0, m_TurnAmount * turnSpeed * Time.deltaTime,
0);
}
{

    if (m_IsGrounded && Time.deltaTime > 0)
    {
        Vector3 v = (m_Animation.deltaPosition * m_MoveSpeedMultiplier)
Time.deltaTime;
        v.y = m_RigidMain.velocity.y;
        m_RigidMain.velocity = v;
    }
}

```

```

}
}
void CheckGroundStatus()
{
    if (jumpTimer <= 0)
    {
        if UNITY_EDITOR
        Debug.DrawLine(transform.position + (Vector3.up * 0.1f),
            transform.position + (Vector3.up * 0.1f) + (Vector3.down *
            m_GroundCheckDistance));
        endif
        if (Physics.Raycast(transform.position + (Vector3.up * 0.1f),
            Vector3.down, out RaycastHit hitInfo, m_GroundCheckDistance))
        {
            m_GroundNormal = hitInfo.normal;
            m_IsGrounded = true;
            m_Animation.applyRootMotion = true;
        }
        else
        {
            m_IsGrounded = false;
            m_GroundNormal = Vector3.up;
            m_Animation.applyRootMotion = false;
        }
    }
}

public void Action(AnimationType animationType, Vector3 target)
{
    if (!IsActing)
    {
        Vector3 copy = transform.eulerAngles;
        transform.LookAt(target);
        copy.y = transform.eulerAngles.y;
        transform.eulerAngles = copy;
        if (animationType == AnimationType.EasyAttack )
        {
            var a = Arm.comboSet.NextEasyAttack();
            Arm.modifier = a.damageModifier * multiplier;
            m_Animation.SetFloat("AttackSpeed", a.speedModifier);
        }
    }
}

```

```

    AnimationOverrideCtrller[AnimationType.EasyAttack.ToString()]
= a.animation;
    m_Animation.Update(0.0f);
}else if(animationType == AnimationType.HardAttack)
{
    var a = Arm.comboSet.NextHardAttack();
    Arm.modifier = a.damageModifier * multiplier;
    m_Animation.SetFloat("AttackSpeed", a.speedModifier);
    AnimationOverrideCtrller[AnimationType.HardAttack.ToString()]
= a.animation;
    m_Animation.Update(0.0f);
}
    m_Animation.Play(animationType.ToString());
    IsActing = true;
}
}
}

```

UserCtrl.cs

```

using System;
using UnityEngine;
using UnityStandardAssets.CrossPlatformInput;
[RequireComponent(typeof(Person))]
[RequireComponent(typeof(Main))]
public class UserCtrl : MonoBehaviour
{
    private Person m_Person;
    ThirdPersonPerson on the object
    private Transform m_Cam;
    to the main camera in the scenes transform
    private Vector3 m_CamForward;
    forward direction of the camera
    private Vector3 m_Move;
    private bool m_Jump;
    public static bool isCtrlable = true;
    Vector3 prevPosition;
    [SerializeField]
    float interactionRadius;
    DialogObject selectedDialogObj;
}

```

```

private CanvasCtrller canvas;
public void PlayLeftClickAnim()
{
    m_Person.Action(AnimationType.EasyAttack,
transform.position + transform.forward);

}

public void PlayRightClickAnim()
{
    m_Person.Action(AnimationType.HardAttack,
transform.position + transform.forward);
}
private void Start()
{
    prevPosition = transform.position;
    if (Camera.main != null)
    {
        m_Cam = Camera.main.transform;
    }
    m_Person = GetComponent<Person>();
    var canvasGO =
Instantiate(Resources.Load<GameObject>("Prefabs/Canvas"));
    canvas = canvasGO.GetComponent<CanvasCtrller>();
    canvas.player = this.gameObject;
}
private void Update()
{
    UpdateSphereCast();
    if (Input.GetKeyDown(KeyCode.U))
    {
        canvas.ArmSlider.Next();
    }

    if (isCtrllable)
    {
        if (Input.GetKeyDown(KeyCode.I))
        {

```

```

canvas. StockPointPanelContainer.gameObject.SetActive(true)
;
isCtrlable = false;
}
if (Input.GetKeyDown(KeyCode.F))
{
    if (selectedDialogObj != null)

selectedDialogObj.Interact(m_Person);
}
}
else
{
    if (Input.GetKeyDown(KeyCode.I))
    {

canvas. StockPointPanelContainer.gameObject.SetActive(false
);
isCtrlable = true;
}
if (Input.GetKeyDown(KeyCode.K))
{
    canvas.ArmSlider.Expand();
}
}
}
void UpdateSphereCast()
{
    Ray ray =
Camera.main.ScreenPointToRay(Input.mousePosition);
    Debug.DrawRay(ray.origin, ray.direction);
    RaycastHit[] raycastHits =
Physics.SphereCastAll(ray, interactionRadius);
    foreach (var hit in raycastHits)
    {
        var hint =
hit.collider.gameObject.GetComponent<IShowAdvice>();
        var Dialog =
hit.collider.gameObject.GetComponent<DialogObject>();

```

```

    if (hint != null)
    {
hint.ShowHint();
    }
    if (Dialog != null)
    {
        selectedDialogObj = Dialog;
    }
    }
    private void FixedUpdate()
    {
        prevPosition = transform.position;
        if (!m_Jump)
        {
            m_Jump =
CrossPlatformInputManager.GetButtonDown("Jump");
        }
        if (isCtrllable)
        {
            if (Input.GetMouseButton(0))
            {
                PlayLeftClickAnim();
            }
            if (Input.GetMouseButton(1))
            {
                PlayRightClickAnim();
            }
            float h =
CrossPlatformInputManager.GetAxis("Horizontal");
            float v =
CrossPlatformInputManager.GetAxis("Vertical");
            bool crouch = Input.GetKey(KeyCode.C);
            if (m_Cam != null)
            {
move:
                m_CamForward = Vector3.Scale(m_Cam.forward,
new Vector3(1, 0, 1)).normalized;

```

```

    m_Move = v * m_CamForward + h * m_Cam.right;
}
else
{
m_Move = v * Vector3.forward + h *
Vector3.right;
}
if (Input.GetKey(KeyCode.LeftShift)) m_Move *=
0.5f;
endif
script
    m_Person.Move(m_Move, crouch, m_Jump);
    m_Jump = false;
}
else
{
    m_Person.Move(m_Move*0, false, m_Jump);
}
}
}

```

AICtrl.cs

```

using System;
using System.Collections.Generic;
using UnityEngine;
using UnityStandardAssets.CrossPlatformInput;
[RequireComponent(typeof(Person))]
[RequireComponent(typeof(Main))]
public class AICtrl : MonoBehaviour
{
    private Person m_Person;
    ThirdPersonPerson on the object
    private Transform m_Cam;
    to the main camera in the scenes transform
    private Vector3 m_CamForward;
    private Vector3 m_Move;
    private bool m_Jump;
    camForward and user input.

```

```

public static bool isCtrlable = true;
Vector3 prevPosition;
[SerializeField]
float interactionRadius;
DialogObject selectedDialogObj;
State state = State.Attack;
Person curentTarget;
float attackDistance = 1.3f;
private void Start()
{
m_Person = GetComponent<Person>();
}
private void FixedUpdate()
{
if ((int)(Time.time * 10) % 5 == 0)
CheckForEnemy();
switch (state)
{
case State.Attack:
if (curentTarget != null)
{
if (Vector3.Distance(transform.position,
curentTarget.transform.position) < attackDistance)
{
float r =
UnityEngine.Random.Range(0f,1f);
if (r < 0.3f)
{
m_Person.Action(AnimationType.HardAttack,
curentTarget.transform.position);
}
else if(r<0.9f)
{

m_Person.Action(AnimationType.EasyAttack,
curentTarget.transform.position);
}
}
else
{

```

```

m_Person.Move((curentTarget.transform.position -
transform.position).normalized, false, false);
    }
    }
else
{

m_Person.Move((curentTarget.transform.position -
transform.position).normalized, false, false);
    }
    }

break;
case State.Idle:
break;
case State.Run:
break;
}
}

void CheckForEnemy()
{
    Debug.Log("AICtrl.CheckForEnemy()");
    List<Person> Persons =
GameDataOptimizer.GetValue<Person>();
    if (Persons!=null&&Persons.Count > 0)
    {
        Debug.Log("Persons.Count: " +
Persons.Count);
        Person closestPerson = null;
float closestDistance =float.MaxValue;
        foreach (var c in Persons)
        {
            if (c !=
m_Person&&c.GetComponent<Main>().isDied==false)
            {
                float d =
Vector3.Distance(transform.position, c.transform.position);

```

```

        if (d < closestDistance)
        {
            closestDistance = d;
            closestPerson = c;
        }
    }

    if (closestPerson != m_Person)
    {
        curentTarget = closestPerson;
    }
}

{
    Attack,
    Run,
    Idle
}
}

```

Arm.cs:

```

using System.Collections;
using System.Collections.Generic;
using UnityEngine;
public class Arm : Point
{
    public float damage;
    public GameObject owner;
    public AttackCombo comboSet;
    public float pushForce = 30;
    public float modifier;
    bool isAtacking = false;
    List<Main> hitBodies = new List<Main>();
    protected override void Start()
    {
        base.Start();
    }
    public void StartAtack()
    {

```

```

attackTimer = 3;
isAtacking = true;
}
public void EndAttack()
{
hitBodies.Clear();
Debug.Log("Arm.EndAttack()");
isAtacking = false;
}
private void Update()
{
}
public void DealDamage(Main Main,Vector3 hitPoint)
{

    if (isAtacking&& owner.GetComponent<Main>()!= Main
&&!hitBodies.Contains(Main))
{
    hitBodies.Add(Main);
    Main.Push(((hitPoint ).normalized *
pushForce*modifier));
    Main.TakeDamage(damage * modifier);
multiplier);
    Debug.Log("Final Damage: " + damage );
}
}
private void OnTriggerEnter(Collider other)
{
Main Main = other.GetComponent<Main>();
if (Main != null&&owner!=null)
    DealDamage(Main, other.transform.position -
owner.transform.position);
}
}

```

Point.cs:

```

using System.Collections;
using System.Collections.Generic;
using UnityEngine;
public class Point : MonoBehaviour,Cooperate

```

```

{
    public Pixy Pixy;
    public string PointName;
    public string objectPath;
    public PointType PointType;
    public bool IsPickedUp { get; private set; }
    DialogObject _Dialog;
    public DialogObject Dialog {
        get
        {
            if (_Dialog==null)
            {
                _Dialog =
                Instantiate(Resources.Load<GameObject>(DialogObject.path
), transform).GetComponent<DialogObject>();
            }
            return _Dialog;
        }
        set { _Dialog = value; }
    }
    protected virtual void Start()
    {
        Dialog.text = PointName;
    }
    public void Throw()
    {
        IsPickedUp = false;
        Dialog.interactEnabled = true;
        gameObject.SetActive(true);
        transform.parent = null;
    }
    public void Pick(Transform parent)
    {
        IsPickedUp = true;
        Dialog.interactEnabled = false;
        Debug.Log("Point.Pick");
        gameObject.SetActive(false);
        transform.SetParent(parent);
    }
}

```

```

public void Hide()
{
    gameObject.SetActive(false);
}

public void Equip(Transform parent)
{
    IsPickedUp = true;
    Dialog.interactEnabled = false;
    gameObject.SetActive(true);
    transform.SetParent(parent);
    transform.position = parent.transform.position;
    transform.rotation = parent.rotation;
}

public void Destroy()
{
    Destroy(gameObject);
}

public void Interact(Person Person,
DialogObject DialogObject)
{
    if (!IsPickedUp)
    {
        Dialog = DialogObject;
        Pick(Person.transform);
        Person.AddPointToInventory(this);
    }
}

public enum PointType
{
    Arm,
    Armor,
    Others,
}
}

                                DialogObject:

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

```

```

public class DialogObject : MonoBehaviour, IShowAdvice
{
    public static readonly string path =
    "Prefabs/DialogObject";
    [HideInInspector]
    public TextMeshPro meshPro;
    public SphereCollider sphereCollider;
    public bool showHint = true;
    public string text;
    public bool interactEnabled = true;
    public void ShowHint()
    {
        if (interactEnabled)
        {
            Debug.Log("DialogObject.ShowHint()");
            meshPro.enabled = true;
            transform.rotation =
            Camera.main.transform.rotation;
            enabled = true;
            showHint = true;
        }

    }
    public void Interact(Person Person)
    {
        Debug.Log("DialogObject.Interact");
        Cooperate interact =
        GetComponentInParent<Cooperate>();
        if (interact != null)
            interact.Interact(Person, this);
    }
    void Start()
    {
        sphereCollider = GetComponent<SphereCollider>();
        meshPro = GetComponent<TextMeshPro>();
        meshPro.text = text;
    }
    void Update()
    {

```

```

    if(meshPro!=null&&!showHint)
    meshPro.enabled = false;
    showHint = false;
}
}
Листинг ArenaConroller:
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
public class ArenaCtrller : MonoBehaviour
{
    [SerializeField]
    List<GameObject> bosses;
    [SerializeField]
    GameObject player;
    int currentBoss = 0;
    float sec = 10.0f;
    bool coroutineIsRunnig = false;
    void Start()
    {
        bosses[currentBoss].SetActive(true);
    }
    void Update()
    {
        if (currentBoss<bosses.Count&&
bosses[currentBoss].GetComponent<Main>().isDied)
        {
            if(!coroutineIsRunnig)
            StartCoroutine(WaitForSec());
        }
    }
    IEnumerator WaitForSec()
    {
        coroutineIsRunnig = true;
        yield return new WaitForSeconds(sec);
        player.GetComponent<Main>().currentHealth =
400*((currentBoss+1)/(bosses.Count/2));
        player.GetComponent<Person>().multiplier *= 1.1f;
        bosses[currentBoss].SetActive(false);
    }
}

```

```

currentBoss++;
if (currentBoss < bosses.Count)
bosses[currentBoss].SetActive(true);
coroutineIsRunnig = false;
}
}

Attack

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

[CreateAssetMenu(menuName = "custom/AttackCombo", fileName
="newAttackCombo")]
public class AttackCombo : ScriptableObject
{
    public List<Attack> easyAttacks;
    public List<Attack> hardAttacks;
    int nextEasyAnimationID = -1;
    int nextHardAnimationID = -1;
    public Attack NextEasyAttack()
    {
        nextEasyAnimationID++;
        if (nextEasyAnimationID > easyAttacks.Count - 1)
            nextEasyAnimationID = 0;
        return easyAttacks[nextEasyAnimationID];
    }
    public Attack NextHardAttack()
    {
        nextHardAnimationID++;
        if (nextHardAnimationID > hardAttacks.Count - 1)
            nextHardAnimationID = 0;
        return hardAttacks[nextHardAnimationID];
    }
}

Листинг CameraCtrlrler
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class CameraCtrlrler : MonoBehaviour

```

```

{
    [SerializeField] Transform cameraY;
    [SerializeField] Transform cameraX;
    [SerializeField] float mouseSensetivity = 100f;
    [SerializeField] float yStopper = 0f;
    [SerializeField] float smoothSpeed = 0.125f;
    public Transform target;
    float xRotation = 0f;
    float yRotation = 0f;
    void Start()
    {
    }
    private void Update()
    {
        if (UserCtrl.isCtrllable)
        {
            transform.LookAt(target);
            Cursor.lockState = CursorLockMode.Locked;
            float mouseX = Input.GetAxis("Mouse X") *
mouseSensetivity * Time.deltaTime;
            float mouseY = Input.GetAxis("Mouse Y") *
mouseSensetivity * Time.deltaTime;
            xRotation -= mouseY;
            yRotation += ;
            xRotation = Mathf.Clamp(xRotation, -90f, 90f -
yStopper);
            cameraY.localRotation =
Quaternion.Euler(xRotation,
cameraY.localRotation.eulerAngles.y, 0f);
            cameraY.Rotate(Vector3.up * mouseX);
        }
        else
        {
            Cursor.lockState = CursorLockMode.None;
        }
    }
    void FixedUpdate()
    {
        cameraY.position = Vector3.Lerp(cameraY.position,

```

```

target.position, smoothSpeed);
    }
}

        StockPointPanelContainer

using System.Collections;
using System.Collections.Generic;
using UnityEngine;
public class StockPointPanelContainer :
MonoConduct, IPointSwap
{
    Inventory playerInventory;
    List<PointPanel> PointPanels = new List<PointPanel>();

    public bool PointSwap(PointPanel PointPanel)
    {
        List<Point> Points = new List<Point>();
        foreach(PointPanel ip in PointPanels)
        {
            Points.Add(ip.Point);
        }
        playerInventory.SetPoints(Points);
        return true;
    }
    void Start()
    {
        playerInventory =
GetComponentInParent<CanvasCtrller>().player.GetComponent<
Person>().inventory;
        playerInventory.GetPoints();
        Debug.Log("Inventory.Start()");
        playerInventory.onPointsChange += UpdatePoints;
        Debug.Log("playerInventory.onPointsChange:");
        Debug.Log(playerInventory);
        int capacity = playerInventory.Capacity;
        var Points = new Point[capacity];
        foreach(Point Point in Points)
        {
            PointPanel panel=
Instantiate(Resources.Load<GameObject>(PointPanel.path),

```

```
transform).GetComponent<PointPanel>();  
panel.Point = Point;  
PointPanels.Add(panel);  
}  
gameObject.SetActive(false);  
}  
void UpdatePoints(List<Point> Points)  
{  
    for(int i=0;i<Points.Count;i++)  
    {  
        PointPanels[i].Point = Points[i];  
    }  
}  
}
```

ДОДАТОК Б
Слайди презентації

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЗАПОРІЗЬКА
ПОЛІТЕХНІКА»
КАФЕДРА ПРОГРАМНИХ ЗАСОБІВ

Дипломна кваліфікаційна робота за темою
«Розробка технічної демонстраційної версії
комп'ютерної гри»

Виконала: Власова Карина Андріївна, ст. гр. КНТ-227
Керівник: Зайко Тетяна Анатоліївна, к.т.н. доцент НУ"ЗП"

1

Рисунок Б.1 – Слайд №1

Об'єкт та предмет дослідження

Об'єкт дослідження – процеси реалізації демонстраційної версії комп'ютерної гри.

Предмет дослідження – методи розробки комп'ютерних ігор.

Мета роботи – розробка технічної демонстраційної версії комп'ютерної гри.

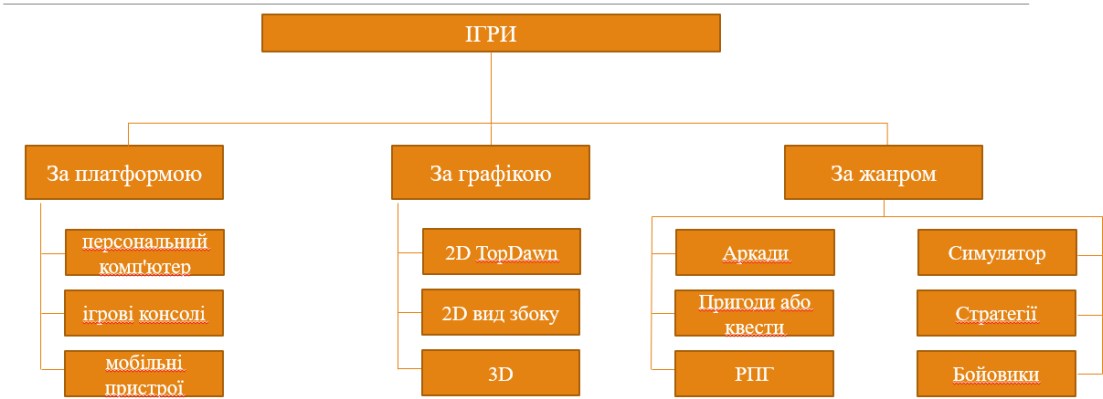
У зв'язку з поставленою метою вирішуються такі завдання:

- дослідити існуючі аналоги програмного продукту і виявити їхні переваги і недоліки;
- спроектувати алгоритми для реалізації ігрової програми з урахуванням переваг і недоліків існуючих аналогів;
- реалізувати складені алгоритми на мові програмування C #;
- провести тестування візуального представлення алгоритмів

2

Рисунок Б.2 – Слайд №2

КЛАСИФІКАЦІЯ ІГОР



3

Рисунок Б.3 – Слайд №3

ІЄРАРХІЯ КЛАСІВ

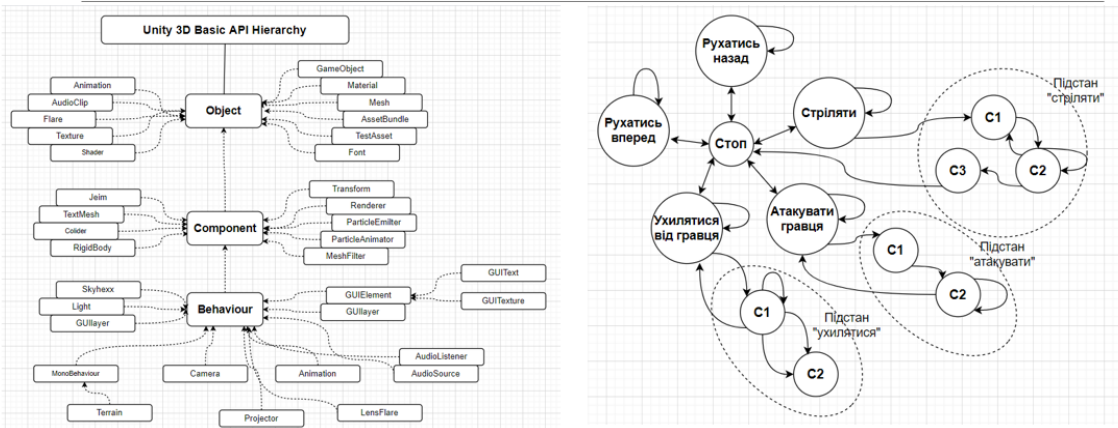


Рисунок 1 – Ієрархія класів Unity 3 d

Рисунок 2 – Кінцевий автомат для опису поведінки гравця

4

Рисунок Б.4 – Слайд №4

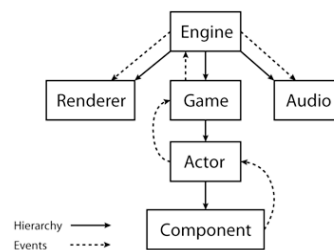
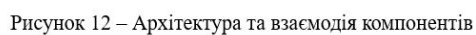


Рисунок 11 – Прототип архітектури ігрового рушія



ВИМОГИ ДО АППАРАТНОЇ ЧАСТИНИ ЗАСТОСУНКУ

Для функціонування програми потрібно встановити наступні компоненти:

- MS DirectX 9;
- MS dotNet Framework 3.5;
- драйвер для аудіо пристрої;
- драйвер для відео пристрої.

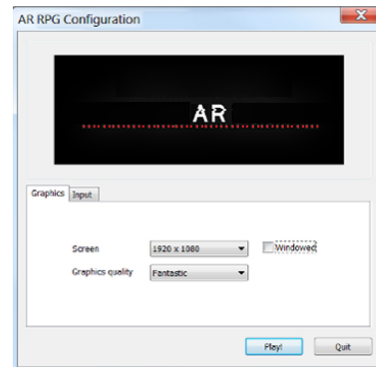


Рисунок 13 – Вікно запуску застосунку

9

Рисунок Б.9 – Слайд №9

ГРАФІЧНИЙ ІНТЕРФЕЙС



Рисунок 7 – Головне меню

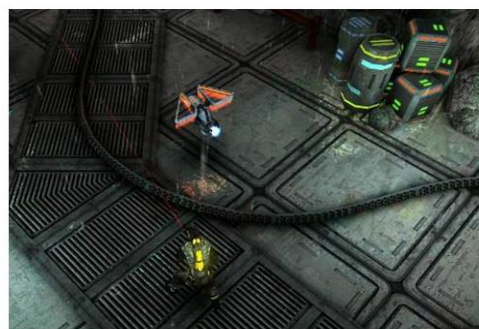


Рисунок 8 – Ігровий процес

10

Рисунок Б.10 – Слайд №10

ГРАФІЧНИЙ ІНТЕРФЕЙС

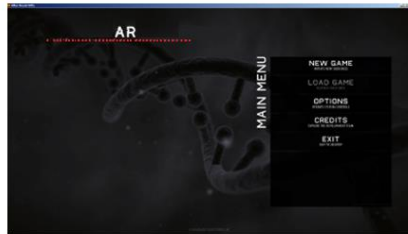


Рисунок 14– Головне меню



Рисунок 15 – Ігровий процес

11

Рисунок Б.11 – Слайд №11

ГРАФІЧНИЙ ІНТЕРФЕЙС



Рисунок 16 – Внутрішньоігрове меню

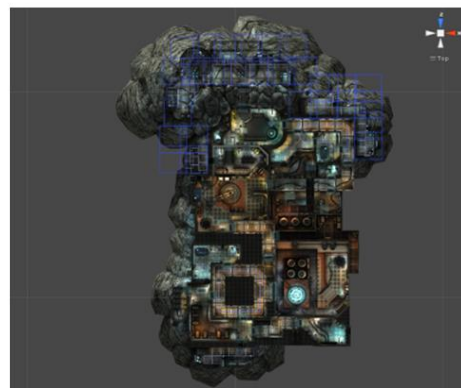


Рисунок 17 – Карта ігрового рівня

12

Рисунок Б.12 – Слайд №12

ВИСНОВОК

Під час виконання кваліфікаційної роботи бакалавра було розроблено програмний продукт «Технічна демонстраційна версія тривимірної комп'ютерної гри "A R"». При розробці комп'ютерної гри було реалізовано:

- підсистеми пошуку шляху,
- графічного інтерфейсу користувача
- взаємодії внутрішньоігрових об'єктів.

Всі підсистеми налагоджені, оптимізовані, протестовані і інтегровані в єдину програмну систему.

Даний програмний продукт є унікальним, тому що, всі системи були реалізовані без використання готових рішень. Для всіх підсистем були реалізовані власні механізми, які не мають аналогів.

В ході виконання роботи було успішно виконано наступні завдання:

- досліджено існуючі аналоги програмного продукту і виявлені їхні переваги і недоліки;
- спроектовано алгоритми для реалізації ігрової програми з урахуванням переваг і недоліків існуючих аналогів;
- реалізовано складені алгоритми на мові програмування C #;
- проведено тестування візуального представлення алгоритмів.

Розроблений програмний продукт повністю відповідає вимогам технічного завдання.

13

Рисунок Б.13 – Слайд №13

Дякую за увагу!

14

Рисунок Б.14 – Слайд №14