

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:

**«Комп'ютерна гра "Nixton's adventure". Програмна
частина»**

Виконав студент групи КН-20-2
СОІН Едуард Едуардович

Керівник роботи:
ПЕТРОСЯН Руслан Валерікович

Рецензент:
МАРЧУК Галина Вікторівна

Житомир – 2024

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

ЗАТВЕРДЖУЮ

Завідувач кафедри комп'ютерних наук

Марина ГРАФ

«25» січня 2024 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: **СОІН Едуард Едуардович**

Керівник роботи: **ПЕТРОСЯН Руслан Валерікович**

Тема роботи: **«Комп'ютерна гра "Nixton's adventure". Програмна частина»,**
затверджена наказом закладу вищої освіти від **«25» січня 2024 р., №22/с**

Термін здачі закінченої роботи: **«20» червня 2024 р.**

Вихідні дані роботи (зазначається предмет і об'єкт дослідження): використання професійного пакету для розробки програмного коду.

Консультанти кваліфікаційної роботи із зазначенням розділів, що їх стосуються:

Розділ	Консультант	Дата	
		Завдання видав	Завдання прийняв
1	Руслан ПЕТРОСЯН	20.07.2023	01.10.2023
2	Руслан ПЕТРОСЯН	02.10.2023	21.03.2024
3	Руслан ПЕТРОСЯН	05.04.2024	10.06.2024

Календарний план

№ з/п	Назва етапів кваліфікаційної роботи	Термін виконання етапів роботи	Примітка
1	Визначення проблеми. Проведення дослідження, огляду та аналізу аналогічних рішень. Складання технічного завдання.	11 лютого 2024 – 9 Березня 2024	Виконано
2	Розробка механік та поділ на класи	10 березня 2024 – 28 березня 2024	Виконано
3	Створення напів відкритого світу та написання коду	29 березня 2024 – 28 квітня 2024	Виконано
4	Тестування роботи механік та ігрового світу	29 квітня 2024 – 18 травня 2024	Виконано
5	Написання пояснювальної записки	19 травня 2024 – 1 червня 2024	
6	Попередній захист	27 червня 2024	Виконано

Здобувач вищої освіти **Едуард СОІН**
Керівник **Руслан ПЕТРОСЯН**

РЕФЕРАТ

Створення кваліфікаційної роботи охоплює декілька процесів це процес написання коду та формування комп'ютерної графіки та написання пояснювальної записки. Робота включає вступ, три розділи, висновки, перелік використаних джерел і додаток.

Текстова частина викладена на 82 сторінках.

Пояснювальна записка має 51 сторінок додатків. Список використаних джерел містить 20 найменувань і займає 2 сторінки. В роботі наведено 43 рисунків та 1 таблиця. Загальний обсяг роботи – 142 сторінок.

Ключові слова: UNITY, ГРА, ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ, JETBRAINS RIDER, PREFAB, 3D, CLASS, C#, ADVENTURE.

					КН.КР.Б – 122 – 24 – ПЗ			
Змн.	Арк.	№ докум.	Підпис	Дата	Комп'ютерна гра "Nixton's adventure". Програмна частина Пояснювальна записка	Літ.	Арк.	Аркушів
Розроб.		Е.Е. СОІН						
Керівник		Р.В. ПЕТРОСЯН					3	98
Рецензент		Г.В. МАРЧУК				Житомирська політехніка, група КН-20-2		
Н. контр.								
Зав. каф.		М.С. ГРАФ						

ABSTRACT

The creation of a qualification paper covers several processes: the process of writing code and creating computer graphics and writing an explanatory note. The work includes an introduction, three chapters, conclusions, a list of used sources and an appendix.

The text part is laid out on 82 pages.

The explanatory note has 51 pages of annexes. The list of used sources contains 20 names and occupies 2 pages. The work contains 43 figures and 1 table. The total volume of work is 142 pages.

Keywords: UNITY, GAME, SOFTWARE, JETBRAINS RIDER, PREFAB, 3D, CLASS, C#, ADVENTURE.

		Е.Е. СОИН			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Р.В. ПЕТРОСЯН				
Змн.	Арк.	№ докум.	Підпис	Дата		4

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ	6
ВСТУП	7
РОЗДІЛ 1. АНАЛІЗ ПРОБЛЕМАТИКИ, МЕТОДІВ ТА ЗАСОБІВ ВИРІШЕННЯ ЗАДАЧІ.....	9
1.1 Постановка задачі	9
1.2 Аналіз аналогів програмного продукту	11
Висновки до першого розділу	25
РОЗДІЛ 2. ПРОЕКТУВАННЯ ТА РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ	27
2.1 Візуалізація процесів розробки програмного забезпечення	27
2.2 Використання компонентів під час розробки гри	39
2.3 Основні етапи реалізації гри.....	41
Висновки до другого розділу.....	70
РОЗДІЛ 3. ОПИС РОБОТИ З ПРОГРАМНИМ ДОДАТКОМ ТА ЙОГО ТЕСТУВАННЯ	71
3.1 Порядок встановлення та налаштування системи.....	71
3.2 Інтерфейс та порядок роботи з програмним додатком.....	72
3.3 Тестування роботи програмного додатку	79
Висновки до третього розділу	82
ВИСНОВКИ.....	83
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	85
ДОДАТКИ.....	87

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

ПЗ – програмне забезпечення.

ШІ – штучний інтелект.

RB - rigidbody - це один з основних компонентів у Unity, який використовується для забезпечення фізичної поведінки об'єктів у грі.

ООП — це парадигма програмування, яка розглядає програму як набір об'єктів, що взаємодіють між собою.

		Е.Е. СОІН			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Р.В. ПЕТРОСЯН				
Змн.	Арк.	№ докум.	Підпис	Дата		6

ВСТУП

Актуальність теми. Актуальність теми ігор Adventure полягає в їхній здатності створювати неповторні світи та надихати гравців на відкриття нових граней віртуальної реальності. Ці ігри відображають собою комбінацію захоплюючих пригод, розв'язання загадок та інтригуючих сюжетів, що робить їх популярними серед різних вікових груп.

Adventure ігри часто використовуються як засіб для розвитку критичного мислення, логічних навичок та творчого мислення гравців. Вони стимулюють активну участь у вирішенні завдань та викликають необхідність стратегічного планування. Завдяки гнучкості у геймплей і можливості вибору, Adventure ігри дозволяють гравцям впливати на розвиток подій у грі і відчувати особисту відповідальність за її вигляд.

Крім того, такі ігри часто використовуються для створення навчальних програм, які дозволяють вивчати історію, культуру та інші наукові аспекти через взаємодію з віртуальним середовищем. Це робить їх не лише розважальними, а й освітніми інструментами, сприяючи збільшенню інтерактивності та поглибленню знань у гравців.

Загалом, Adventure ігри залишаються актуальними завдяки своїй здатності трансформувати гравців у активних учасників віртуального світу, де кожна дія може мати вагому значущість для подальшого розвитку історії та результатів гри.

Метою кваліфікаційної роботи є створення програмної частини для гри в жанре Adventure.

Об'єктом дослідження для кваліфікаційної роботи є створення програмної частини ігрового додатку

Предметом дослідження є використання IDE JetBrains Rider на мові програмування C#, та ігровий рушій Unity для створення ігрового всесвіту та взаємодію між скриптами.

В рамках теми кваліфікаційної роботи було опубліковано тези:
- Граф М. С., Соін Е. Е. Аналіз проблеми створення штучного інтелекту

		Е.Е. СОІН			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Р.В. ПЕТРОСЯН				
Змн.	Арк.	№ докум.	Підпис	Дата		7

Аналіз проблеми створення штучного інтелекту. Інформаційно-комп'ютерні технології: стан, досягнення та перспективи розвитку, 25-26 листопада 2021 року. Житомир : «Житомирська політехніка», 2021. С.107-108.

- Граф М. С., Соін Е. Е. ChatGPT для проведення аналізу. Інформаційно-комп'ютерні технології – 2023 30-31 березня 2023 року. Житомир : «Житомирська політехніка», 2023. С.69-70.

		Е.Е. СОІН			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Р.В. ПЕТРОСЯН				
Змн.	Арк.	№ докум.	Підпис	Дата		8

РОЗДІЛ 1. АНАЛІЗ ПРОБЛЕМАТИКИ, МЕТОДІВ ТА ЗАСОБІВ ВИРІШЕННЯ ЗАДАЧІ

1.1 Постановка задачі

Метою кваліфікаційної роботи є створення програмного частини для гри в жанрі Adventure.

Основною метою розробки гри в жанрі Adventure є створення основних механік цього жанру, а також додавання унікальних елементів, які гармонійно впишуться у контекст даного жанру ігор, забезпечення повного ігрового циклу. Важливими аспектами є:

1. Експлорація та напіввідкритий світ: гравець буде досліджувати різнобарвний і загадковий світ, вивчаючи нові локації і знаходячи таємничі артефакти та ресурси.
2. Бойова система: розробка варіантів боротьби з ворогами, які вимагають стратегії та тактичного мислення, забезпечення динамічних інтеракцій між гравцем і ворогами.
3. Система прогресу: впровадження системи прокачування гравця та розвитку його навичок, що дозволяє покращувати характеристики і здібності під час гри.
4. Динамічні події та випробування: включення різноманітних подій та випробувань, які тестують навички гравця і пропонують варіативні сценарії.

Для успішного виконання кваліфікаційної роботи необхідно розробити та реалізувати ключові ігрові механіки, створити структуру класів та зібрати проект:

Перший етап. Зробити напіввідкритий світ:

1. Зробити напіввідкритий світ та за допомогою колайдерів закрити деякі частини світу

Другий етап. Створити переміщення гравця:

1. Придумати та створити переміщення гравця також зробити слідкуючу камеру від 3 особи [\[1\]](#)
2. Додати стрибок та прискорення у гравця.

		Е.Е. СОІН			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Р.В. ПЕТРОСЯН				
Змн.	Арк.	№ докум.	Підпис	Дата		9

3. Зробити тест і перевірити чи все коректно працює.

Третій етап. Розробка системи інвентар:

1. Створити Canvas створити UI елемент Panel та зробити Panel дочірнім елементом Canvas.

2. Створити декілька ячеек які будуть дочірніми елементами Panel для інвентарю.

3. Створити об'єкти які можна буде підбирати.

4. Реалізувати можливість підбирати використовувати та вибрасувати об'єкти з інвентарю

5. Зробити перевірку.

Четвертий етап. Розробка системи бою для гравця:

1. Створити та додати зброю.

2. Додати зброю на мапу да додати можливість її підбирати.

3. Створити ворога якого можна буде бити.

4. Реалізувати можливість підбирання об'єкта в інвентар та доставит меч звіддти.

5. Зробити тест перевірку.

П'ятий етап. Розробити ворогів та розмістити їх на мапі

1. Створити ворогів додати та створити декілька варіантів можливості ворогів.

2. Розробити ворога ближнього бою та розмістити його на мапі.

3. Розробити системи бою для ворога.

4. Розробка декілька можливостей у ворога таких як погоня і патрулювання

5. Створити систему здоров'я.

6. Розробити анімації та їх переходи та додати їх ворогу

7. Зробити тест перевірку.

Шостий етап. Зробити об'єднання всіх попередніх елементів

1. Створити сцену на якій будуть проходити всі основні етапи

2. Зробити напіввідкритий всесвіт та перенести на нього ворога.

		Е.Е. СОІН			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Р.В. ПЕТРОСЯН				10
Змн.	Арк.	№ докум.	Підпис	Дата		

3. Створити гравця та перенести інвентар.
4. Створити декілька сцен та переходи між ними.
5. Зробити тест перевірку.

1.2 Аналіз аналогів програмного продукту

Для того щоб почати розробити гру по перше потрібно проаналізувати ринок та подивитися схожі ігрові додати того ж жанру в нас цей жанр Adventure. Для аналізу було продивлено декілька ігор жанру Adventure.

«Absolver» [2] – це бойова гра, яка комбінує елементи бойового мистецтва і ролевої гри. Вона зосереджується на бою один до одного (PvP) і співпраці (PvE) між гравцями в унікальному віртуальному світі. Гра пропонує глибоку систему бойових маневрів і налаштування персонажа, де гравці можуть вдосконалювати свої бойові вміння і отримувати нові рухи, щоб стати справжніми майстрами бойових мистецтв.

Хоча "Absolver" має елементи пригодницького жанру, зокрема дослідження великого віртуального світу і взаємодію з іншими гравцями, основний акцент зроблений на бойових механіках і співпраці/змаганні з іншими учасниками. Таким чином, вона більше може вважатися бойовою грою (fighting game) з елементами рольової гри, а не традиційною пригодницькою грою.

"Absolver" - це гра, де гравець контролює бойовика, який вивчає різноманітні бойові мистецтва, покращує свої навички та взаємодіє з іншими персонажами у фантастичному світі

"Absolver" виділяється серед інших ігор своїм унікальним підходом до бойової системи, яка базується на реальних бойових мистецтвах і вимагає від гравця вміння тактично планувати і виконувати удари та блоки. Також варто відзначити його унікальну механіку мультіплеєра, де гравці можуть взаємодіяти один з одним у режимі реального часу, проводячи битви або кооперативні місії (рис.1.1).

		Е.Е. СОІН			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Р.В. ПЕТРОСЯН				
Змн.	Арк.	№ докум.	Підпис	Дата		11



Рисунок 1.1 – Заставка гри «Absolver»

Переваги механік гри:

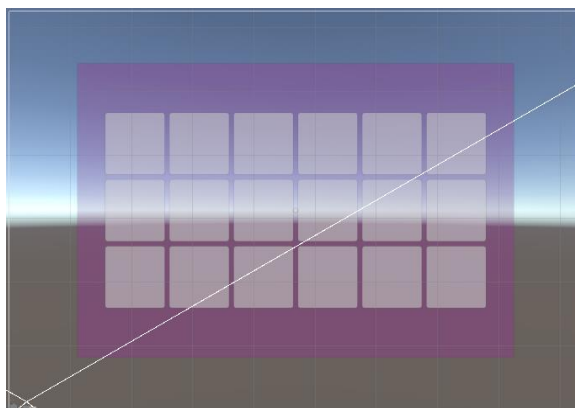
1. Унікальна бойова система: "Absolver" відрізняється від інших ігор своєю складною інтерактивною бойовою системою, яка базується на реальних бойових мистецтвах. Гравці можуть створювати та налаштовувати комбінації ударів, блоків і уників, що дозволяє створювати власний унікальний стиль бою.(Рис.1.2).
2. Мистецтво війни: гра акцентується на вивченні різних стилів бойових мистецтв і використанні їх у бою. Гравці можуть навчатися від NPC інструкторів, а також від інших гравців, що робить гру вельми пізнавальною і різноманітною в аспекті бойових технік.
3. Мультиплеєрна взаємодія: гра сприяє спільному гравецькому досвіду, де гравці можуть взаємодіяти один з одним у режимі реального часу. Відбуваються як сутички один на один, так і спільні місії чи тренування.
4. Персоналізація персонажа: гравці можуть налаштовувати зовнішній вигляд свого персонажа, а також вибирати і прокачувати різноманітні рухи і вміння, що дозволяє створити унікальний стиль гри.
5. Динамічна атмосфера: гра має вражаючу графіку і атмосферу, що додає до іммерсивності гри. Містична атмосфера світу "Absolver" і його загадковість приваблюють гравців усього світу.

		Е.Е. СОІН			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Р.В. ПЕТРОСЯН				
Змн.	Арк.	№ докум.	Підпис	Дата		12



Рисунок 1.2 – Унікальна бойова система: "Absolver"

Механіка інвентарю заслуговує увагу так як ця механіка також була реалізована у грі "Nixton's adventure". Це дуже потрібна механіка для зберігання айтемов які гравець буде знаходити під час своїх пригод у грі та якщо це якісь речі які можна буде на прикладі меч за допомогою якого гравець буде битися або якась їжа за допомогою якого гравець буде вилікувати себе (рис. 1.3)



а)



б)

Рисунок 1.3 – (а) приклад інвентарю в грі "Nixton's adventure", (б)приклад інвентарю в грі "Absolver" ,

Недоліки в механіках гри:

Хоча гра має дуже багато позитивних аспектів але є і негативні одним з цих недолік виявляється дуже короткий сюжет та не багату кількість ворогів і босів у грі (рис. 1.4).

		Е.Е. СОІН			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Р.В. ПЕТРОСЯН				
Змн.	Арк.	№ докум.	Підпис	Дата		13



Рисунок 1.4 – Ворог у грі "Absolver"

Ще одним недоліком на даний момент це невеликий онлайн и проблеми з серверами в дуже багатьох випадках у PVP режимах у гравця будуть телепорти між ворогами.

Наступною грою буде «Life is Strange» [3] – це емоційний пригодницький інтерактивний відео-сюжетний жанр. Гра складається з п'яти епізодів, де гравець контролює головного героя Максін Колфілд, яка володіє здатністю переміщати час назад. Гравці приймають рішення, які впливають на події і кінцівки, створюючи унікальний досвід для кожного гравця.

Основна ціль гри "Life is Strange" полягає в розвитку історії та взаємодії з персонажами через прийняття важливих рішень, які впливають на події в грі. Гравці виступають у ролі Макс Колфілд, молодої студентки, яка володіє здатністю переміщати час назад. Основні цілі гри включають:

1. Розслідування та вирішення таємниць: гравець розкриває таємниці, що оточують головну героїню та інших персонажів, збираючи інформацію та розв'язуючи головоломки.
2. Взаємодія з персонажами: відносини з іншими персонажами розвиваються залежно від рішень, які гравець приймає під час діалогів та взаємодії.
3. Прийняття важливих рішень: гравець змушений приймати складні моральні та етичні рішення, які впливають на хід подій і кінцевий результат гри.

		Е.Е. СОІН			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Р.В. ПЕТРОСЯН				
Змн.	Арк.	№ докум.	Підпис	Дата		14

4. Дослідження інтерактивних середовищ: гравець має можливість досліджувати деталізовані рівні та взаємодіяти з різними об'єктами, що допомагає розкрити додаткові деталі історії.

5. Емоційне занурення: гра надає емоційний досвід через сильний сюжет та персонажів, які стикаються зі складними емоційними і особистими викликами (рис.1.5).



Рисунок 1.5 – Заставка гри «Life is Strange»

Переваги механік гри:

"Life is Strange" має кілька ключових переваг у своїх механіках, які визначають його як унікальну гру:

1. Система вибору і наслідків: гравець має можливість приймати рішення, які безпосередньо впливають на події у грі та розвиток сюжету. Ці рішення часто мають далекосяжні наслідки, що стимулює повторне проходження гри для дослідження різних сценаріїв (рис. 1.6).

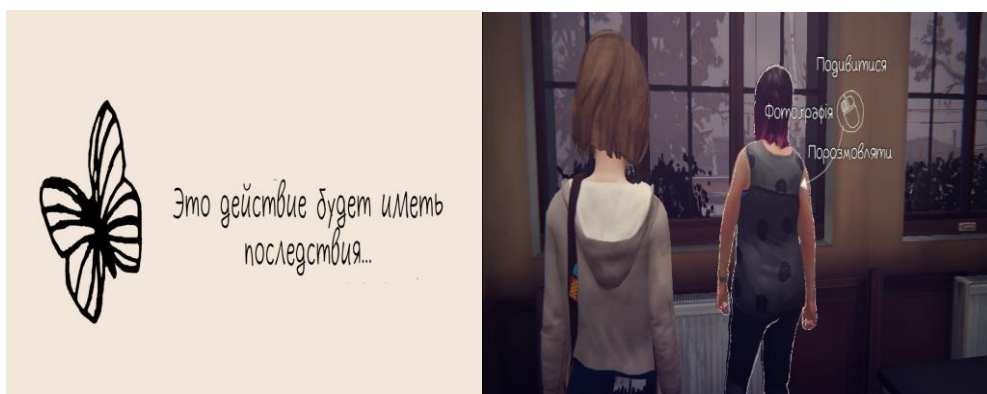
2. Здатність переміщати час назад: ця механіка дозволяє гравцеві переглядати свої рішення та вибирати альтернативні шляхи. Вона не лише розширює геймплейні можливості, а й впливає на взаємодію з персонажами та вирішення головоломок.

3. Емоційна і глибока історія: гра відома своєю запаморочливою історією, яка змушує гравців відчувати широкий спектр емоцій. Теми дружби, любові, втрати і само відкриття виграють важливу роль у приверненні уваги до ігрового світу (рис. 1.7).

4. Інтерактивне оточення: гравці мають можливість взаємодіяти з навколишнім середовищем, що допомагає їм краще зрозуміти імерсійний світ гри і знаходити додаткові деталі історії.

		Е.Е. СОІН			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Р.В. ПЕТРОСЯН				
Змн.	Арк.	№ докум.	Підпис	Дата		15

5. Художня графіка та атмосфера: унікальний стиль малюнку і атмосфера гри додають до загального настрою і емоційності досвіду гравця.



а)

б)

Рисунок 1.6 – (а)Приклад система наслідків ,(б) Приклад система вибору.



Рисунок 1.7 – Емоційна і глибока історія.

Недоліки механік гри:

"Life is Strange" отримала в цілому позитивні відгуки від критиків і гравців, але деякі недоліки були відзначені спільнотою та деякими оглядачами. Ось деякі з них:

1. Технічні проблеми: деякі гравці стикалися з технічними проблемами, такими як падіння кадрів, затримки або баги, які впливали на загальний досвід гри.

2. Діалоги і сюжет: хоча сюжет був загалом високо оцінений, деякі гравці відзначали не суттєвість або вибір діалогів, які вони здійснювали, впливали на подальший розвиток історії.(рис. 1.8).

		Е.Е. СОІН			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Р.В. ПЕТРОСЯН				
Змн.	Арк.	№ докум.	Підпис	Дата		16

3. Психологічні аспекти персонажів: часто було відзначено, що деякі персонажі могли б виходити за межі архетипічних рис, що здавалося штучним чи нереалістичним.

4. Механіка часу: хоча механіка перемотування часу була ключовим елементом гри, деякі гравці відчували, що її реалізація не завжди була послідовною або інтуїтивно зрозумілою.

5. Завершення гри: деякі гравці відзначали, що фінальна частина гри або її завершення могли б бути більш задовільними або логічно завершеними.



Рисунок 1.8 – Приклад діалогів.

«Tomb Raider (1996)» [4] – це класична відеогра в жанрі екшн-пригоди. Гра стала однією з найвпливовіших і найпопулярніших у своїй епосі, значною мірою завдяки своїй новаторській 3D-графіці та геймплею.

У "Tomb Raider" гравці беруть на себе роль Лари Крофт, археолога і шукачки пригод, яка вирушає на пошуки стародавніх артефактів по всьому світу. Лара подорожує до різних екзотичних локацій, таких як Перу, Греція, Єгипет та Атлантида. Основна мета гри — знайти частини артефакту під назвою "Сціон".

Геймплей у "Tomb Raider" поєднує в собі платформер, головоломки та елементи шутеру. Гравцям потрібно досліджувати великі рівні, повні пасток, ворогів та загадок, які потрібно вирішити, щоб просунутися далі. Лара Крофт має

		Е.Е. СОІН			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Р.В. ПЕТРОСЯН				
Змн.	Арк.	№ докум.	Підпис	Дата		17

широкий спектр рухів, включаючи біг, стрибки, плавання, лазіння і навіть виконання акробатичних трюків. Крім того, гравці можуть використовувати різноманітні види зброї, щоб захищатися від ворогів, включаючи диких тварин і людських противників (рис.1.9).

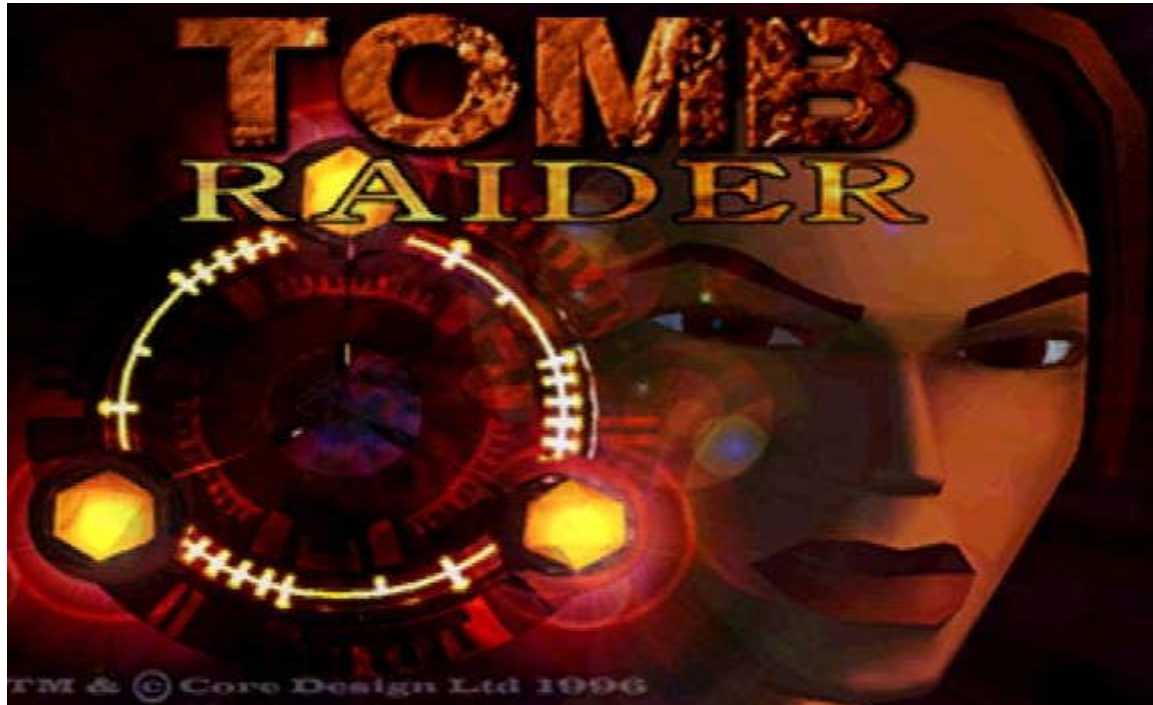


Рисунок 1.9 – Заставка гри «Tomb Raider»

Переваги механік гри:

Новаторський тривимірний геймплей :

– "Tomb Raider" стала однією з перших ігор, яка успішно реалізувала тривимірний геймплей у жанрі екшн-пригоди. Це дозволило гравцям досліджувати великі, відкриті локації з високою свободою пересування, що на той час було новинкою.

Інтуїтивне управління:

– гра пропонувала широкий спектр рухів і дій для головної героїні, Лари Крофт, таких як біг, стрибки, плавання, лазіння і виконання акробатичних трюків. Інтуїтивне управління зробило ці дії легкими для виконання, забезпечуючи захопливий ігровий досвід.

Комбінація екшну і головоломок:

– "Tomb Raider" майстерно поєднала елементи екшну з головоломками. Гравці повинні були не тільки боротися з ворогами, але й

		Е.Е. СОІН			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Р.В. ПЕТРОСЯН				
Змн.	Арк.	№ докум.	Підпис	Дата		18

вирішувати складні головоломки, що додавало різноманітності та глибини геймплею.

Різнманітність локацій:

– гра пропонувала різні екзотичні локації, такі як Перу, Греція, Єгипет та Атлантида. Кожна локація мала унікальний дизайн і атмосферу, що робило дослідження кожного рівня цікавим і непередбачуваним (рис. 1.10).

Захоплюючий сюжет:

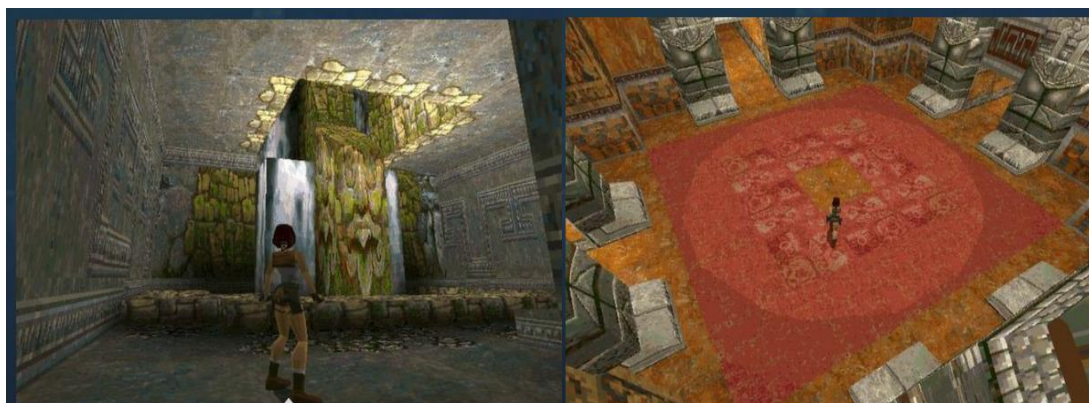
– "Tomb Raider" представила захоплюючий сюжет, який мотивував гравців продовжувати свої пригоди. Історія про пошуки стародавнього артефакту "Сціон" включала інтриги, небезпеки і загадкові події, що підсилювало загальне враження від гри.

Варіативність бою:

– гравці мали доступ до різноманітної зброї, включаючи пістолети, дробовики та гранатомети, що дозволяло використовувати різні тактики для боротьби з ворогами. Це робило бої цікавими та стратегічними (рис. 1.11).

Висока реіграбельність:

– завдяки складним головоломкам, прихованим секретам і різноманітним ворогам, "Tomb Raider" мала високу реіграбельність. Гравці могли повертатися до гри, щоб знайти всі секрети і повністю дослідити всі рівні.



(а)

(б)

Рисунок 1.10 – (а) перший різновид локацій з декількох можливих, (б) другий різновид з декількох можливих.



Рисунок 1.11 – Різновид зброї

Недоліки механік гри:

Складне управління:

– попри інноваційність, управління Ларою Крофт у грі було досить складним і незграбним. Гравцям часто було важко точно виконувати стрибки та інші акробатичні трюки, що призводило до частих помилок і розчарувань.

Обмежена камера:

– камера в грі мала фіксовані кути і часто не слідувала за гравцем належним чином. Це ускладнювало орієнтацію в просторі і робило бій і платформінг незручними, особливо в тісних приміщеннях або під час битв з ворогами.

Недостатня інтуїтивність головоломок:

– деякі головоломки в грі були занадто складними і не завжди інтуїтивно зрозумілими. Гравці часто застрягали на певних рівнях, не розуміючи, що потрібно зробити для їх проходження, що могло викликати розчарування (рис. 1.12).

Відсутність можливості збереження у будь-який момент:

		Е.Е. СОІН			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Р.В. ПЕТРОСЯН				
Змн.	Арк.	№ докум.	Підпис	Дата		20

– гра дозволяла збереження лише у спеціальних точках, що іноді було незручним. Гравці могли втратити значний прогрес, якщо не змогли дістатися до точки збереження перед загибеллю або виходом з гри.

Ліміти анімацій і рухів:

– анімації Лари Крофт були обмеженими, і її рухи іноді виглядали механічними і неприродними. Це впливало на загальне враження від геймплею і могло заважати повному зануренню в гру.

Проблеми з колізією:

– у грі часто траплялися проблеми з колізією об'єктів, коли Лара могла застрягти в текстурах або рухатися через об'єкти. Це могло призвести до несправедливих смертей або втрати прогресу.

Одноманітність боїв:

– бої з ворогами були досить одноманітними і не вимагали особливої тактики або стратегії. Гравці зазвичай використовували ті самі методи боротьби, що зменшувало загальну різноманітність геймплею.



Рисунок 1.12 – Недостатня інтуїтивність головоломок.

1.3 Обґрунтування вибору інструментальних засобів

При розробці комп'ютерної гри "Nixton's Adventure" у жанрі Adventure game, було обрано використання ігрової рушії Unity[4]. Цей вибір зумовлений низкою важливих переваг, які надає Unity порівняно з іншими

		Е.Е. СОІН			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Р.В. ПЕТРОСЯН				
Змн.	Арк.	№ докум.	Підпис	Дата		21

інструментальними засобами.

1. Кросплатформеність

Unity підтримує кросплатформену розробку, що дозволяє створювати ігри для різних операційних систем та платформ, включаючи Windows, macOS, Linux, iOS, Android, а також консолі (PlayStation, Xbox, Nintendo Switch). Це значно розширює потенційну аудиторію гри та полегшує її розповсюдження.

2. Потужний інструментарій для розробки

Unity надає широкий спектр інструментів для розробки, включаючи потужний редактор, систему компонентів, а також візуальні редактори для роботи з 3D-моделями, анімаціями, звуками та ефектами. Це дозволяє значно спростити процес створення гри та зробити його більш ефективним.

3. Велика спільнота та підтримка

Однією з ключових переваг Unity є велика і активна спільнота розробників. Це означає наявність великої кількості навчальних матеріалів, форумів, відео уроків та прикладів коду, що робить легшим вирішення будь-яких проблем та освоєння нових функцій движка. Крім того, Unity має відмінну документацію та офіційну підтримку.

4. Інтеграція з C#

Unity [5] використовує мову програмування C#, яка є однією з найбільш популярних та зручних мов для розробки ігор. C# [6] забезпечує високу продуктивність, простоту у використанні та велику кількість бібліотек і фреймворків, що дозволяє швидко та ефективно реалізовувати необхідні механіки та функціонал гри.

5. Вбудовані фізичні та анімаційні системи

Unity надає потужні засоби для роботи з фізикою та анімаціями. Вбудовані фізичні системи дозволяють легко налаштовувати поведінку об'єктів у грі, включаючи колізії, гравітацію та інші фізичні властивості. Анімаційні інструменти Unity дозволяють створювати плавні та реалістичні анімації персонажів та об'єктів, що значно покращує загальний вигляд та відчуття від гри.

6. Розширюваність та модульність

		Е.Е. СОІН			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Р.В. ПЕТРОСЯН				
Змн.	Арк.	№ докум.	Підпис	Дата		22

Unity є дуже гнучким і дозволяє розробникам створювати власні плагіни та розширення для додавання специфічного функціоналу. Це робить можливим адаптацію движка до унікальних потреб проекту та створення кастомних рішень для різних аспектів гри.

7. Можливості для прототипування

Unity забезпечує швидке створення прототипів, що є важливим етапом у процесі розробки ігор. Швидке створення і тестування ігрових механік дозволяє виявляти та виправляти помилки на ранніх стадіях, що зменшує витрати часу та ресурсів.

Порівняння Unity та Godot:

Unity і Godot – це два популярні ігрові рушії, які мають свої унікальні переваги і недоліки. Unity є більш потужним та універсальним інструментом, що підходить для розробки різноманітних ігор, від мобільних додатків до високоякісних 3D-проектів. Він підтримує понад 25 платформ, включаючи Windows, macOS, Linux, iOS, Android, консолі та WebGL, що дозволяє розробникам легко створювати багатоплатформові ігри. Unity також пропонує безкоштовну версію для розробників з невеликим доходом, а платні версії надають додаткові функції та професійну підтримку.

Godot, з іншого боку, є повністю безкоштовним і з відкритим вихідним кодом, що робить його привабливим для інді-розробників та невеликих студій. Він підтримує основні платформи, такі як Windows, macOS, Linux, iOS, Android та HTML5, але його підтримка консолей є менш розвиненою. Godot відомий своєю легкою вагою і швидкістю, особливо для 2D-ігор, але для великих 3D-проектів може знадобитися більше зусиль для досягнення оптимальної продуктивності.

Unity має інтуїтивно зрозумілий інтерфейс та багатий набір інструментів, використання C# як основної мови програмування робить його доступним для багатьох розробників. Величезна спільнота, детальна документація та безліч сторонніх ресурсів полегшують навчання та вирішення проблем. Godot використовує власну мову програмування GDScript, яка схожа на Python і легко

		Е.Е. СОІН			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Р.В. ПЕТРОСЯН				
Змн.	Арк.	№ докум.	Підпис	Дата		23

засвоюється, але загальна екосистема інструментів у нього менш розвинена.

Unity постійно оновлюється, додаючи нові функції та поліпшення, що забезпечує його стабільне вдосконалення. Велика спільнота і безліч готових рішень в Unity Asset Store значно прискорюють процес розробки. Godot також регулярно оновлюється, але темп розвитку може бути повільнішим.

Отже, Unity є ідеальним вибором для розробників, які прагнуть створити професійні та комерційні ігри завдяки його потужності, універсальності, широкій підтримці платформ, великій спільноті та багатому набору інструментів. Godot, хоча і є чудовим варіантом для інді-розробників та навчальних проєктів, все ще поступається Unity за кількістю доступних інструментів та підтримкою платформи.

Вибір JetBrains Rider як IDE середовище:

JetBrains Rider – це потужне інтегроване середовище розробки (IDE). Відоме своєю підтримкою .NET, Rider надає розробникам широкий спектр інструментів для написання, налагодження та тестування програмного коду. Це середовище об'єднує кращі можливості ReSharper і IntelliJ IDEA, пропонуючи розробникам зручні інструменти для підвищення продуктивності.

Rider підтримує мови програмування C# [7], VB.NET, ASP.NET, JavaScript, TypeScript, HTML, CSS та багато інших. Завдяки вбудованому аналізатору коду, Rider допомагає розробникам швидко знаходити та виправляти помилки, оптимізувати код та покращувати його структуру. Середовище також має потужні інструменти для ре факторингу, що дозволяють легко змінювати структуру коду без ризику порушити його роботу.

Однією з головних переваг Rider є його кросплатформеність. IDE працює на Windows, macOS та Linux, що робить його універсальним інструментом для розробників, незалежно від обраної операційної системи. Rider інтегрується з такими системами контролю версій, як Git, Mercurial та SVN, забезпечуючи зручний доступ до репозиторіїв та полегшуючи спільну роботу над проєктами.

Rider також пропонує відмінну підтримку Unity, що робить його популярним серед розробників ігор. Інструменти для роботи з Unity включають

		Е.Е. СОІН			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Р.В. ПЕТРОСЯН				
Змн.	Арк.	№ докум.	Підпис	Дата		24

можливості для налагодження, автозавершення коду, ре факторингу та аналізу продуктивності, що значно полегшує розробку ігор.

Завдяки своєму багатofункціональному інтерфейсу, високій швидкості роботи та широкій підтримці мов програмування, JetBrains Rider є ідеальним вибором для розробників, які шукають потужне та ефективне середовище розробки для своїх проєктів.

Порівняння JetBrains Rider з іншими IDE середовищами:

JetBrains Rider vs IntelliJ IDEA:

1. Фокус на .NET: Rider спеціалізується на .NET розробці, тоді як IntelliJ IDEA фокусується на Java та інших JVM-мовах. Однак обидва IDE мають багато спільних інструментів завдяки спільній основі.

2. Підтримка мов програмування: Rider підтримує широкий спектр мов, включаючи C#, VB.NET, JavaScript, TypeScript, HTML і CSS, тоді як IntelliJ IDEA найкраще підходить для розробки на Java, Kotlin, Groovy і Scala.

3. Інтеграція з іншими інструментами JetBrains: обидва IDE легко інтегруються з іншими інструментами JetBrains, такими як TeamCity, YouTrack та Upsource, забезпечуючи повний цикл розробки.

JetBrains Rider vs Visual Studio:

1. Кросплатформеність: Rider працює на Windows, macOS та Linux, тоді як Visual Studio офіційно підтримує тільки Windows, хоча є версія Visual Studio для Mac.

2. Швидкість і легкість: Rider зазвичай вважається більш швидким та легким у порівнянні з Visual Studio, що може бути важливим для великих проєктів.

3. Інтеграція з ReSharper: Rider має вбудовані можливості ReSharper, що забезпечує потужний аналіз коду та ре факторинг, у той час як для Visual Studio ReSharper є окремим плагіном, який може сповільнювати роботу IDE.

4. Ціна: обидва продукти мають платні ліцензії, але Rider зазвичай пропонує більш гнучкі умови для команд розробників.

Висновки до першого розділу

		Е.Е. СОІН			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Р.В. ПЕТРОСЯН				
Змн.	Арк.	№ докум.	Підпис	Дата		25

У першому розділі було детально розглянуто основи створення комп'ютерної гри в жанрі Adventure. Проведений аналіз існуючих аналогів дав змогу визначити ключові елементи та механіки, які мають бути враховані при розробці гри "Nixton's adventure". Було встановлено, що для успішного проектування та реалізації гри необхідно забезпечити якісну графіку, інтуїтивно зрозумілий інтерфейс та захоплюючий ігровий процес.

Також було проведено порівняння різних інструментальних засобів для розробки, зокрема Unity та Godot. Переваги Unity, такі як потужні засоби для створення тривимірної графіки, велика кількість доступних ресурсів і підтримка спільноти, роблять його найкращим вибором для реалізації нашого проекту. Додатково розглянуто використання JetBrains Rider як основного середовища розробки завдяки його швидкості, зручності та глибокій інтеграції з .NET.

Таким чином, перший розділ закладає міцну основу для подальшої розробки гри, визначаючи необхідні інструменти та технології, а також окреслюючи основні вимоги до проекту. Наступні етапи роботи включатимуть детальне проектування механік гри, структури класів та реалізацію основних елементів ігрового процесу.

		Е.Е. СОІН			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Р.В. ПЕТРОСЯН				
Змн.	Арк.	№ докум.	Підпис	Дата		26

РОЗДІЛ 2. ПРОЕКТУВАННЯ ТА РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Візуалізація процесів розробки програмного забезпечення

Діаграми[8] – це чудовий спосіб наочно показати дані, інформацію чи ідеї. Вони використовують різні форми, лінії та символи, щоб зробити інформацію зрозумілішою та легшою для сприйняття. За допомогою діаграм можна легко побачити взаємозв'язки між елементами, структуру процесу або статистичні дані. Наприклад, діаграма ігрового циклу (рис. 2.1) показує, як відбувається процес гри від самого початку запуску до моменту виходу з гри, і які дії може виконувати гравець у різні моменти гри.

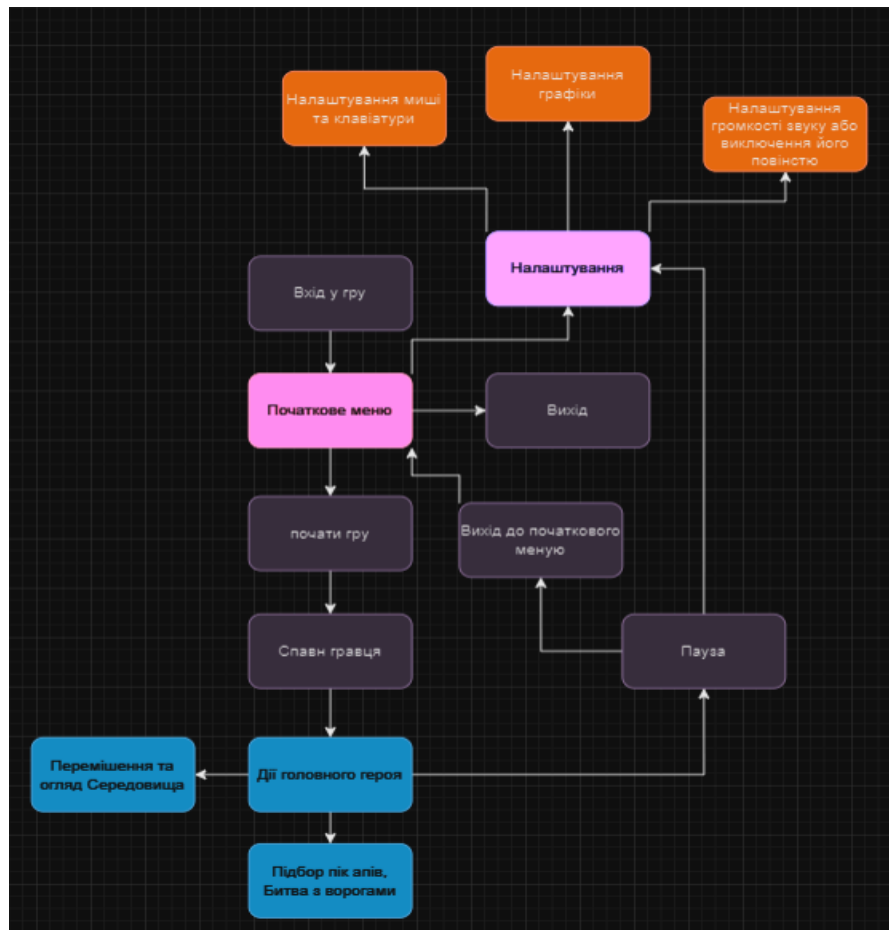


Рисунок 2.1 – Діаграма ігрового циклу

На діаграмі ігрового циклу (див. рис. 2.1) ілюструє весь процес гри від її запуску до завершення, показуючи всі можливі дії, які може виконати гравець під час гри. На початку гри гравець попадає у стартове меню де він може натиснути три кнопки які будуть відповідати за деякі дії а саме кнопка “START”

буде відповідати за початок та запуск гри, кнопка “SETTINGS” буде відповідати за налаштування у грі та кнопка “EXIT” яка буде відповідати за вихід з гри.

Коли гравець натисне кнопку “START” його перенесуть на іншу сцену так звану дім де також будуть аналогічні дії при відкритті меню але кнопка “EXIT” буде відповідати за те щоб повернути гравця до самого початкового меню де вже можна буде повністю вийти з гри. У самій початковій локації гравець може переміщуватися підбирати, використовувати фізичні об'єкти.

Наприклад гравець може підібрати меч та взаємодіяти з оточуючим середовищем наприклад пастками та деякими логічними загадками в деяких випадках гравцю потрібно буде кинути каміння на якусь платформу щоб відчинити якусь частину або знайти плити та нажати її також є можливість що потрібно буде знайти об'єкт та перетягнути його до дверей і покласти біля щоб відчинити двері які відносяться до цієї логічної задачки.

Також як тільки гравець спавниться то спавняться також і вороги у яких є можливість патрулювання та переслідувати гравця. Патрулювання працює таким чином якщо від гравця до ворога дуже велика відстань то ворог патрулює місцевість за яку він відповідає якщо до нього підходить близько гравець то він починає його переслідувати до того моменту поки гравець не відбіжить далеко від ворога якщо гравець все ж таки зміг вирватись то ворог знову починає патрулювати місцевість за яку він відповідає.

Якщо гравець не зможе перемогти, вирватись з лап ворога і вороги його переможуть або гравець потрапить у пастку з якої він не може вибратися то все потрібно буде починати знову. Гравець буде знову заспавлений на локації всі об'єкти які він зібрав потрібно буде збирати знову

Ось як буде виглядати ігровий цикл у грі. В залежності від того як гравець буде проходити рівень буде отримувати різні цікавості наприклад у початку гри гравець отримує меч, при проходженні левелів за пастками та логічним загадками гравець буде отримувати різні нагороди такі як трофеї які він зможе підібрати у свій інвентар.

Діаграма варіантів у гравця (Variant Diagram) (рис. 2.2). Вона вказує на

		Е.Е. СОІН			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Р.В. ПЕТРОСЯН				
Змн.	Арк.	№ докум.	Підпис	Дата		28

таке які дії гравець може робити під час усієї гри.

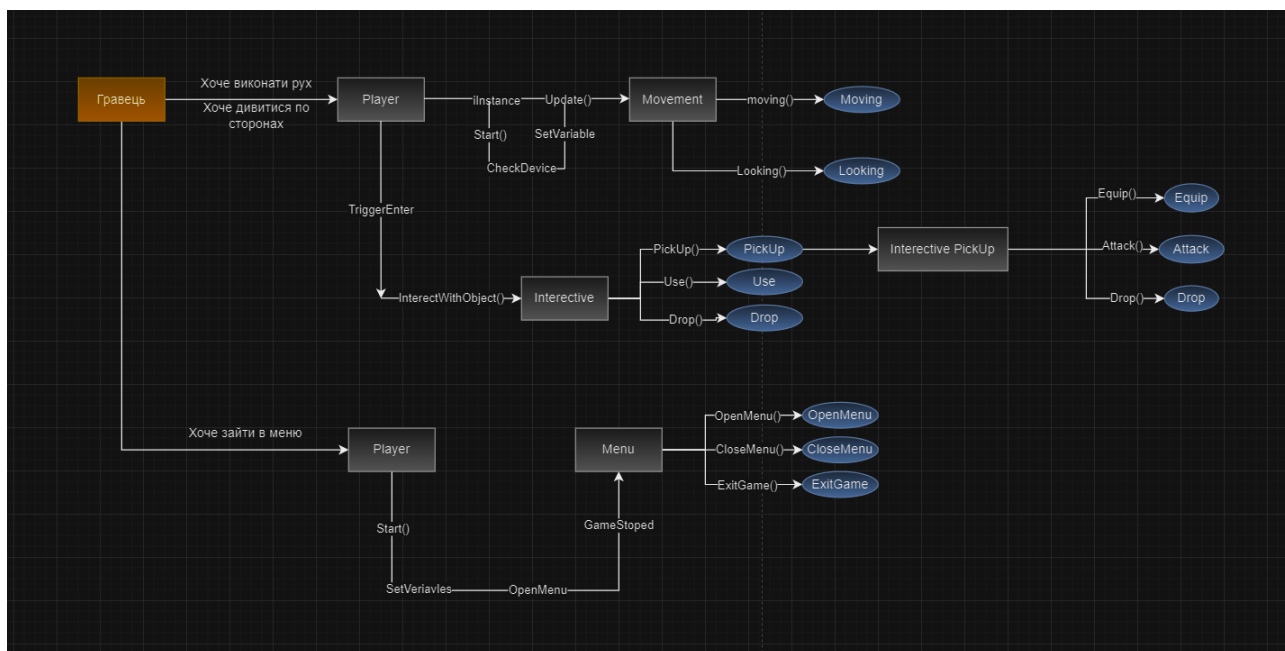


Рисунок 2.2 – Діаграма варіантів

На даній діаграмі варіантів показано всі можливості які може зробити гравець такі як оглядатися чи переміщатися підняти об'єкт якщо його було вже відібрано в інвентар то його можна або використати або викинути якщо цей об'єкт інтерактивний то також додати деякі можливості наприклад екіпірувати або використати цей об'єкт(наприклад з'їсти яблука або екепірувати меч).

Також у гравця є можливість поставити гру на паузу та вийти з гри і перейти на головну сцену. Ця функція доступна на любій сцені та стадії гри.

Головною механі гри будуть вважатися пересування, у яке входить переміщення, стрибок, прискорення, підбирання предметів, інвентар який буде зберігати предмети які буде підібрати гравець, екіпірування та використання предметів з інвентарю та вороги які будуть патрулювати місцевість та ганятись за гравцем якщо він попадає у полі зору ворога.

Діаграма активності для гри - це спосіб показати, як відбувається послідовність дій або подій у грі. Вона відображає всі дії, які може виконати гравець, і як ці дії пов'язані між собою. Така діаграма допомагає зрозуміти, як працює гра, від моменту її запуску до завершення.

Для гри було реалізовано наступні діаграми активності: підбір їжі, підбір та екіпування зброї, удар, та використання.

Для того щоб гравець зміг підібрати їжу (рис. 2.3) потрібно підійти до їжі яка лежить у якомусь місці, потрібно підійти ближче щоб рейкаст дотягнув до їжі та натиснути клавішу підбору, далі перевіряється чи рейкаст дотягується до їжі, спрацювало якщо їжа з'являється в інвентарі, якщо ні гравець не підійшов, або не натиснув клавішу підбору то їжа буде далі лежати. Також йде перевірка чи стакаються айтеми чи ні, якщо ні то створюються ячейки під цей айтем та якщо гравець знову підбере цей же айтем то в інвентарі просто буде +1 в колону з цим айтем та буде відображатися що вже є 2 такий айтема, якщо ні то під кожен новий айтем буду виділятися нова ячейка.

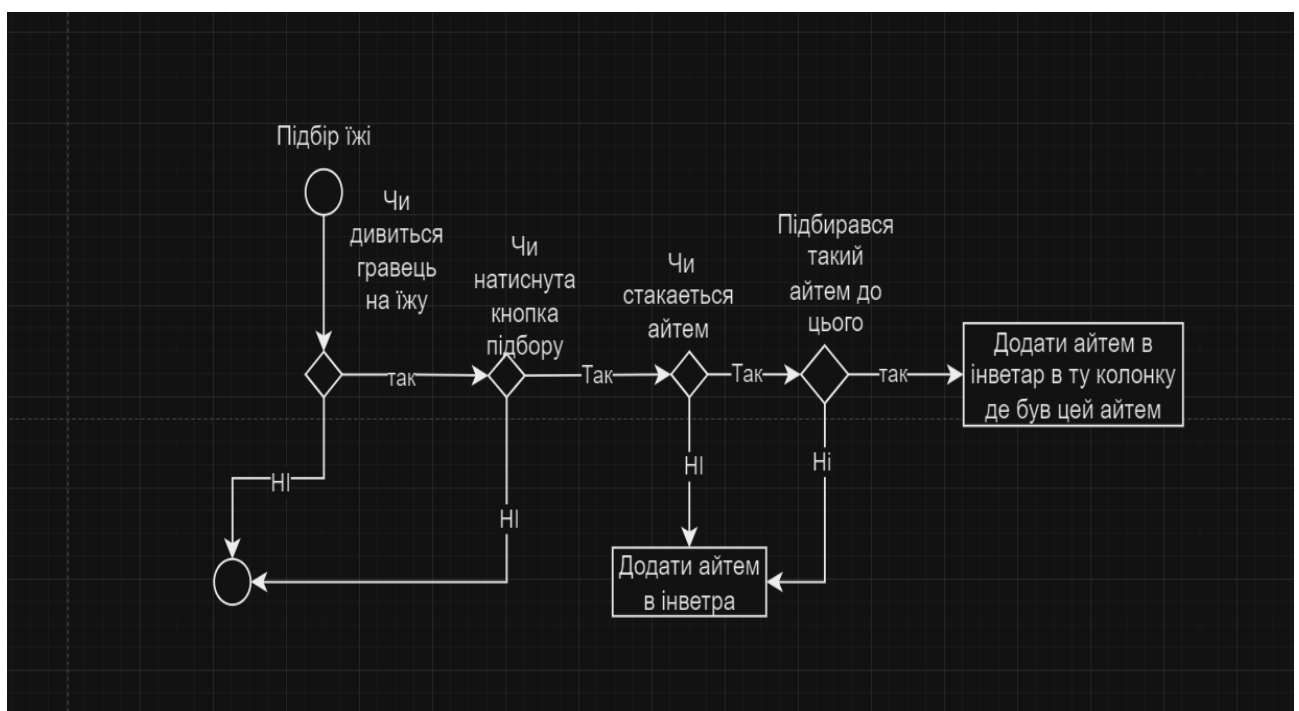


Рисунок 2.3 – Діаграма активності для підбору їжі

Щоб можна було підібрати та екіпірувати зброю (рис. 2.4) потрібно зробити декілька перевірок перша з цих це перевірка чи дивиться персонаж на цей айтем якщо так то можна продовжувати дії та зробити перевірку чи гравець натиснув клавішу яка відповідає за підбор айтему якщо так то айтем підбираємо в інвентар після цього робимо ще одну перевірку чи екіпірований цей айтем якщо так то гравець може зайти у інвентар та натиснути кнопку екіпірування якщо він уже екіпірований ми його прибираємо з рук гравця. Якщо гравець не дивиться на зброю або не натиснув клавішу екіпірування то зброя залишається на місці.

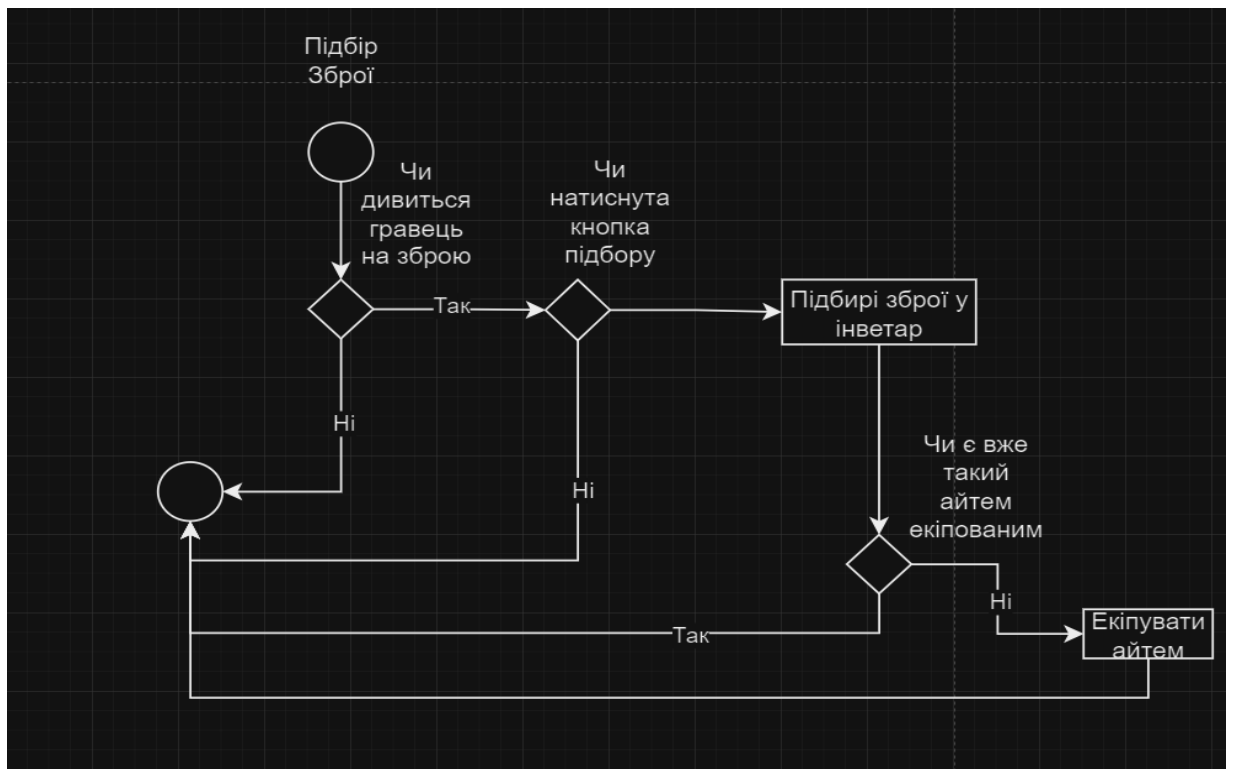


Рисунок 2.4 – Діаграма активності для підбору та екіпування зброї

Для реалізації удару (рис. 2.5) потрібно декілька перевірок чи натиснув гравець кнопку удару, якщо так тоді продовжуємо дії та перевіряємо чи екіпована зброя у гравця, якщо так то продовжується дії та робимо удар. Якщо умови не виконуються або зброя не екіпована то удар не відбувається.

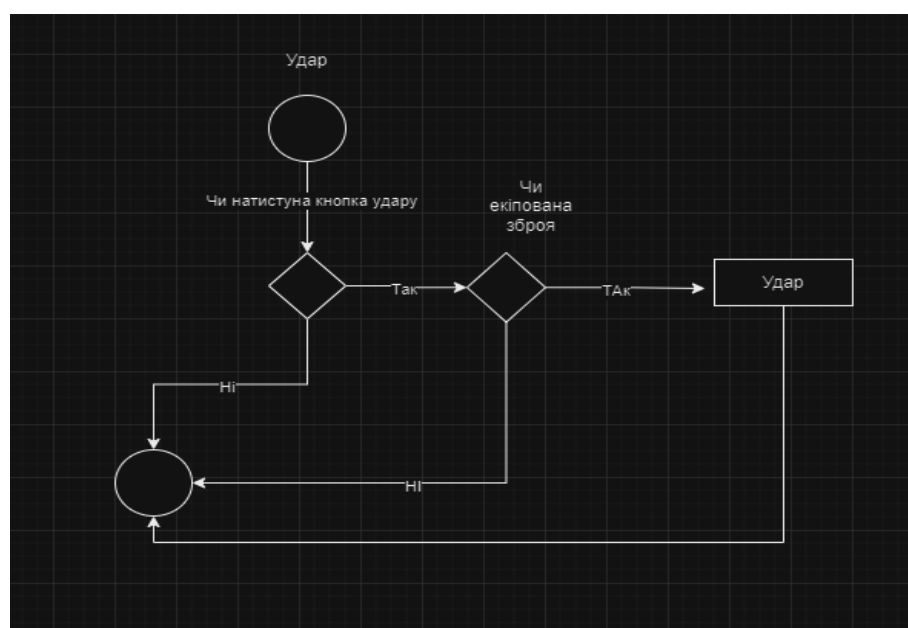


Рисунок 2.5 – Діаграма активності для удару

Для використання айтемі з інвентарю (рис. 2.6) для початку ми перевіряємо чи є у інвентарю айтеми якщо так ми продовжуем та перевіряємо чи натиснув гравець кнопку використання якщо так то ми використовуємо айтем на який гравець навів, на прикладі якщо меч то він екіпірується якщо це якийсь інший айтем який додає чи віднімає ХП то коли гравець натискає то цей інвентар пропадає з інвентарю якщо він не стаскається якщо він стаскається і знаходиться більше ніж 1 екземпляр то трапляється віднімання одного екземпляру. Якщо в інвентарі немає нічого що можна використати то нічого не буде спрацьовувати.

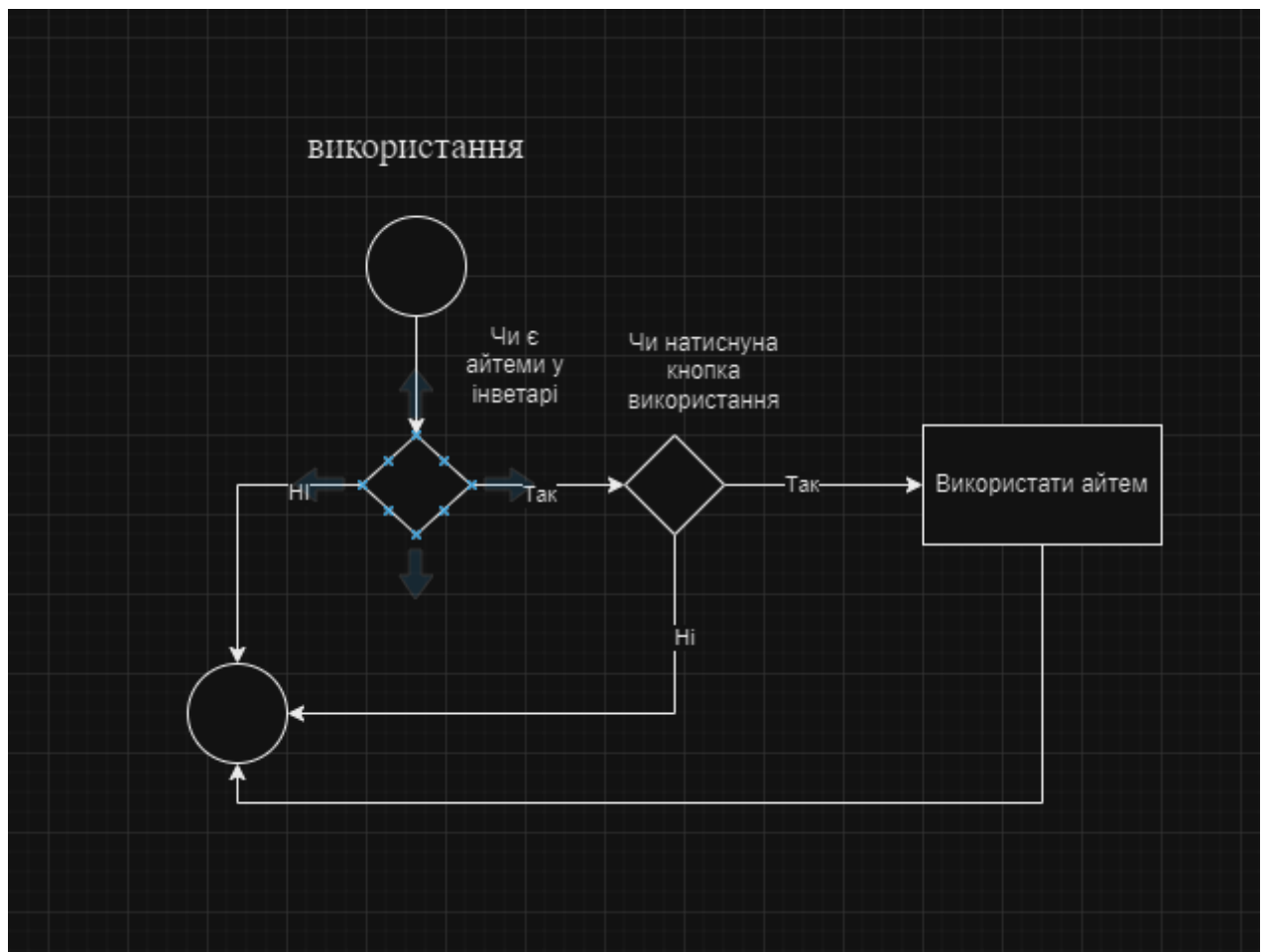


Рисунок 2.6 – Діаграма активності для використання

Також було створено діаграми класів. Діаграми класів створюються для того, щоб показати модифікації, успадкування та взаємозв'язки між різними класами.

На діаграмі (рис. 2.7) зображена діаграма класів гравця та інвентарю, яка демонструє взаємозв'язки між ключовими класами, пов'язаними з гравцем та інвентарем. Ця діаграма ілюструє, як різні класи взаємодіють між собою, щоб

забезпечити функціонування персонажа та інвентаря в грі. Вона включає класи, відповідальні за управління рухом, бойовими діями, інвентарем та іншими аспектами гравця. Кожен клас має свої унікальні атрибути і методи, які визначають його поведінку і роль у загальній структурі гри. Діаграма дозволяє легко зрозуміти, як дані і функції розподіляються між класами, що допомагає у проектуванні та реалізації ігрових механік.

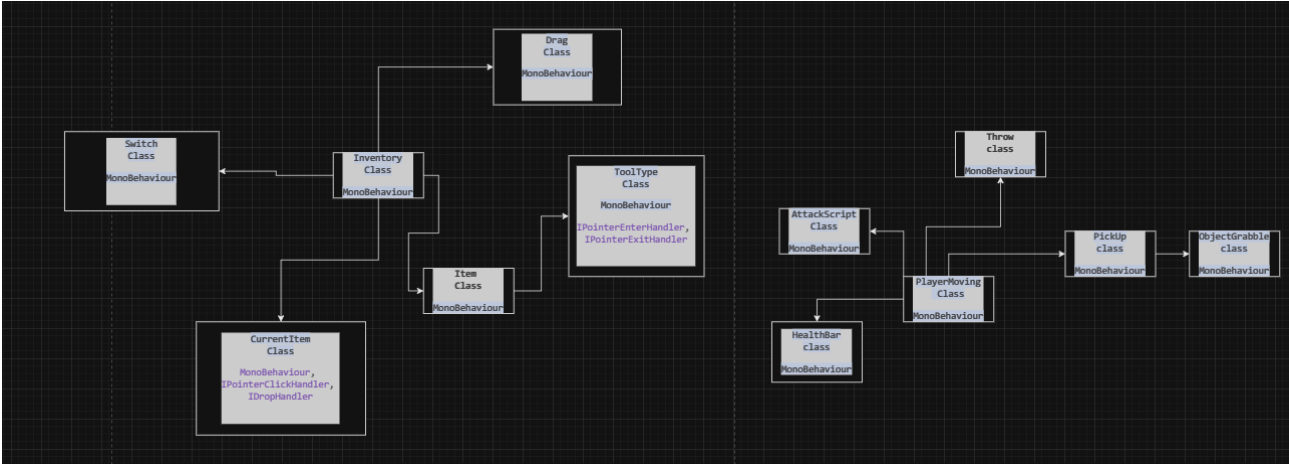


Рисунок 2.7 – Діаграма класів гравця

1. Клас `PlayerMoving` є одним із найважливіших компонентів у розробці гри, оскільки він відповідає за переміщення гравця по ігровому світу. Цей клас забезпечує основні механіки руху персонажа, включаючи ходьбу, біг, стрибки. Правильна реалізація цього класу критично важлива для створення плавного ігрового досвіду.
2. Клас `PickUp` відповідає за підбір айтемів гравцем, їх зберігання та можливість викидання на землю. Він забезпечує основні дії, пов'язані з взаємодією гравця з предметами у грі, що додає реалістичності та інтерактивності ігровому процесу.
3. Клас `ObjectGrabble` є дочірнім класом `PickUp` і відповідає за розширення функціоналу підбору предметів, додаючи можливості взаємодії з об'єктами в більш складний спосіб. Цей клас дозволяє гравцю не тільки підбирати і викидати предмети, але й маніпулювати ними у просторі, наприклад, переміщувати їх або використовувати у різних контекстах гри.
4. Клас `Throw` відповідає за реалізацію механіки кидання об'єктів, зокрема камінців, на визначені платформи для вирішення логічних задач. Цей

клас забезпечує точне та контрольоване кидання предметів, що може бути використано для активації механізмів або розв'язання головоломок у грі.

5. Клас HealthBar відповідає за управління здоров'ям (ХП) гравця, включаючи його збереження та зміну під час ігрового процесу. Цей клас відстежує поточний рівень здоров'я гравця і надає інтерфейс для зменшення або збільшення ХП залежно від подій у грі, таких як отримання шкоди або зцілення.

6. Клас Inventory є ключовим компонентом у реалізації інвентарю в грі. Він відповідає за зберігання, управління та взаємодію з предметами, які гравець може збирати та використовувати під час гри. Правильна реалізація цього класу забезпечує гравцеві зручний і ефективний спосіб керування своїми ресурсами та предметами.

7. Клас Item є основним класом, який відповідає за представлення інформації про окремий предмет в грі. Цей клас використовується в класі Inventory для зберігання, організації та управління різними предметами, які гравець може мати у своєму інвентарі.

8. Клас CurrentItem є частиною реалізації інвентарю в грі і відповідає за відстеження поточного активного предмета, який використовується або показується гравцю в даний момент. Цей клас дозволяє зручно керувати тим, який саме предмет в даний час використовується гравцем зі свого інвентаря.

9. Клас Switch відповідає за екіпірування мечу і забезпечує механізм для керування, яким предметом або знаряддям користується гравець у даний момент.

10. Клас Drag відповідає за реалізацію механіки переміщення айтемів у рюкзаку гравця. Цей клас забезпечує можливість перетягувати айтеми між різними слотами в інвентарі, що дозволяє гравцю організовувати та управляти своїм інвентарем зручним способом.

На наступному кроці ми розглянемо діаграму класів ворогів [9] (див. рис. 2.8), де відображені основні класи ворогів та їх взаємодія. Ця діаграма допомагає нам краще зрозуміти структуру програмного коду, яка включає класи, їх атрибути (поля) та методи, а також зв'язки між ними. Це важливо для належного

		Е.Е. СОІН			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Р.В. ПЕТРОСЯН				
Змн.	Арк.	№ докум.	Підпис	Дата		34

організації логіки гри та взаємодії різних елементів ігрового світу.

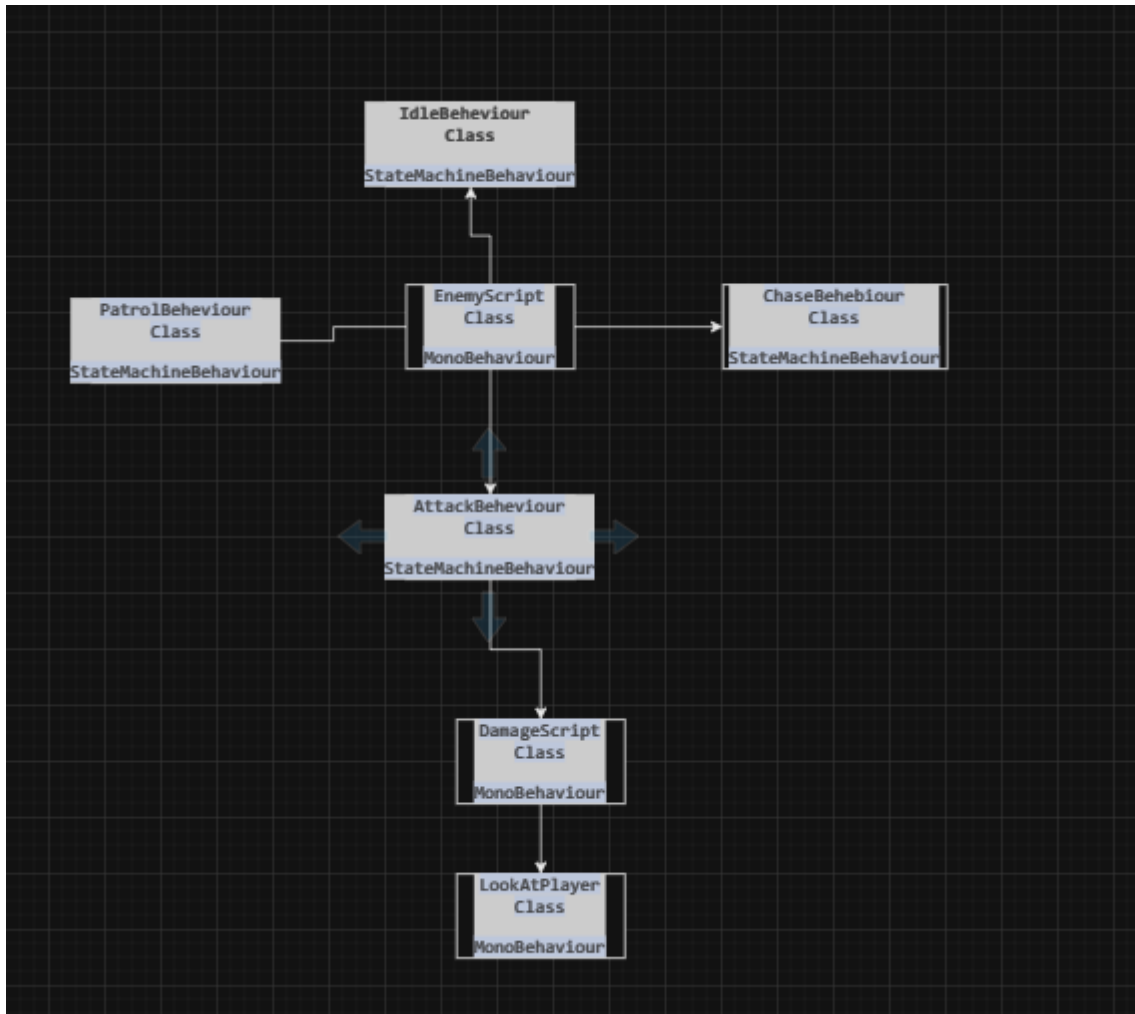


Рисунок 2.8 – Діаграма класів ворога

1. Клас **EnemyScript** є ключовим компонентом у системі, що відповідає за реалізацію ворожих персонажів у грі. Основні функції цього класу включають зберігання та управління здоров'ям (ХП) ворога, реалізацію анімаційних ефектів для рухів і атак, а також обробку отриманого урону від гравця або інших джерел. Цей клас взаємодіє з іншими компонентами гри для реалізації ігрових механік, таких як виявлення колізій, обмін повідомленнями з іншими об'єктами і взаємодія з ігровими подіями.

2. В класі **IdleBehaviour** реалізовано різні аспекти поведінки ворога в грі. Він відповідає за анімацію "айдл" (стан спокою), коли ворог не виявляє активності або не помічає гравця. Цей клас також включає в себе механізми пошуку гравця, що можуть бути використані для виявлення наявності гравця в області видимості ворога. Крім того, він має функціонал патрулювання, який

дозволяє ворогові рухатись вздовж заданого маршруту або області. Іншим важливим аспектом є можливість переходу до стану погоні за головним героєм у випадку виявлення гравця, коли ворог розпізнає небезпеку або вирішує здійснити напад.

Цей клас допомагає створювати відчуття реалістичної поведінки ворогів у грі, що змушує гравця обирати оптимальні стратегії та тактику для перемоги в різних ситуаціях.

3. В класі ChaseBehaviour описано поведінку ворогів у грі. Його основною відповідальністю є здійснення погоні за гравцем. Вороги будуть активно переслідувати гравця, якщо той потрапив у їх радіус видимості або радіус атаки. Цей клас також визначає, як далеко ворог може прослідкувати за гравцем, перш ніж він перестане його переслідувати.

Основні функціональні можливості класу включають визначення відстані, на якій ворог здатен реагувати на гравця, та спосіб, яким він взаємодіє з іншими об'єктами у грі під час погоні. Це важливо для створення відчуття напруги та небезпеки під час ігрового процесу, адже вороги будуть вести активну погоню за гравцем, якщо той знаходиться в їх зоні видимості або атаки.

4. Клас PatrolBehaviour відповідає за патрулювання. Основна його відповідальність це патрулювання між масивом точок у якийсь період часу він переходить з однієї точки до іншої.

5. Клас AttackBehaviour відповідає за поведінку ворога під час атаки на головного гравця. Він включає в себе логіку пошуку гравця в межах заданого радіуса та активацію відповідної анімації атаки. Крім того, клас визначає максимальну дистанцію, на якій ворог може наносити урон гравцеві. Під час атаки ворог може виконувати різні дії, що включають нанесення шкоди гравцеві або виконання спеціальних атак. Клас AttackBehaviour взаємодіє з іншими компонентами гри для забезпечення відповідного рівня складності та реалізму бойових сценаріїв.

6. Клас DamageScript відповідає за обробку урону, який ворог може завдати головному герою в грі. Він зазвичай містить логіку для розрахунку та

		Е.Е. СОІН			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Р.В. ПЕТРОСЯН				
Змн.	Арк.	№ докум.	Підпис	Дата		36

застосування ушкоджень до здоров'я гравця при зіткненні з ворогом. Цей клас може включати методи для визначення величини ушкоджень, взаємодію з системою здоров'я гравця та іншими елементами ігрового процесу, необхідними для обробки урону в грі.

7. В класі LookAtPlayer реалізовано логіку, яка відповідає за автоматичне слідкування ворогом за головним героєм у грі. Цей клас зазвичай включає методи для орієнтації моделі ворога в напрямку гравця, щоб забезпечити те, що ворог завжди спрямований на свою ціль. Використовуючи цей клас, ворог може автоматично обертати свою модель в напрямку гравця, що дозволяє забезпечити більш натуральну інтеракцію між ворогом і гравцем у грі.

Діаграма компонентів (рис. 2.9) демонструє підключення фізичних елементів у систему.

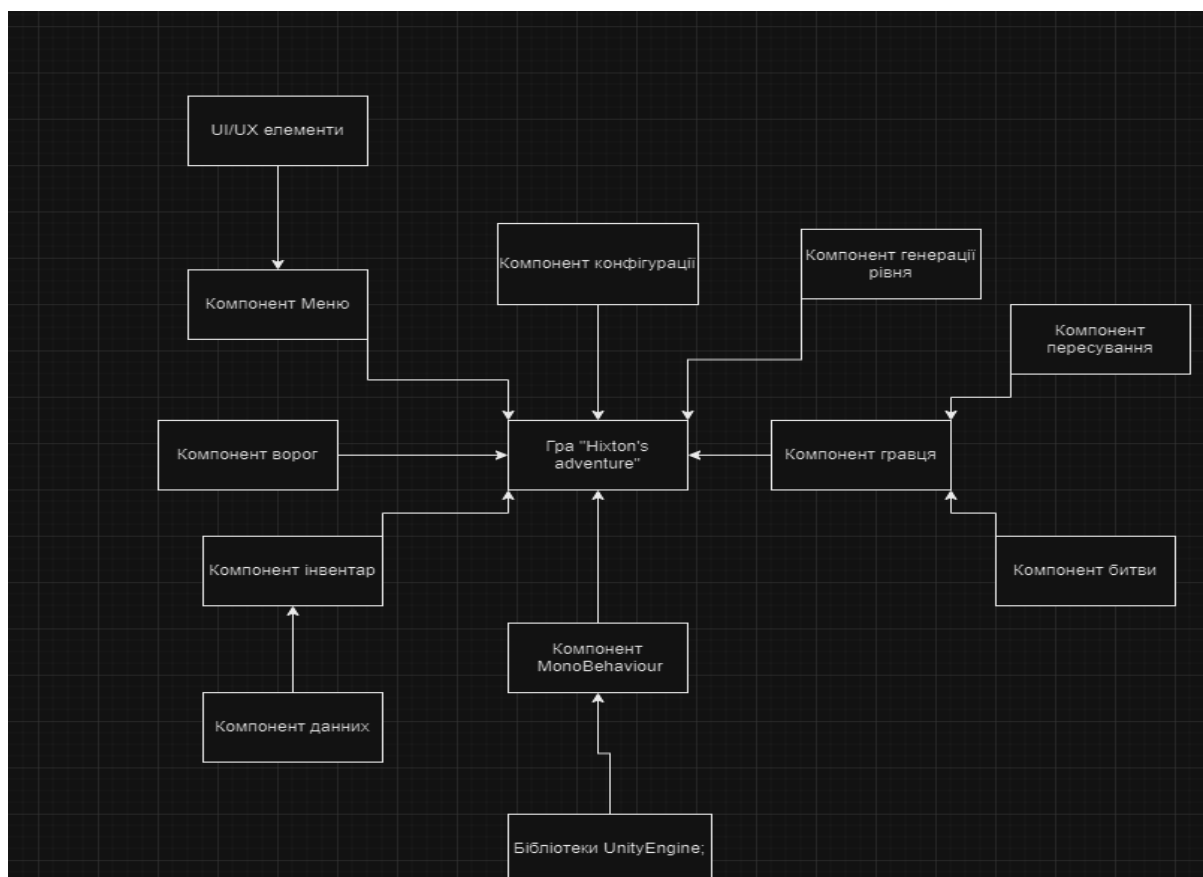


Рисунок 2.9 – Діаграма компонентів

На діаграмі компонентів представлено з яких основних компонентів складається гра а саме : компонента гравця який є одним з найважливіших та включає у себе компонент битви та компонент пересування який також є не

менш важливим, компонент генерації рівня. Так є ще декілька компонентів які також не є менш важливими, а саме компонент Меню який включає у себе UI/UX елементи, компонент інвентар який включає у себе компонент даних, компонент ворогів, компонент конфігурації, компонент ворога та компонент MonoBehaviour який включає у себе компонент бібліотеки UnityEngine.

Візуалізація розгортання - це метод представлення фізичної архітектури програмної системи або програмного продукту. Це означає, що діаграма розгортання показує, як компоненти системи розміщені на обладнанні (фізичних серверах, комп'ютерах або інших пристроях) та як вони взаємодіють між собою через мережеві з'єднання.

Діаграма розгортання зазвичай використовує такі елементи, як вузли (або середовища), компоненти програмного забезпечення (наприклад, додатки або сервіси), мережеві з'єднання (наприклад, мережеві канали або протоколи) та фізичні об'єкти (сервери, комп'ютери, зберігання даних тощо).

Плюси використання діаграм розгортання:

1. Візуалізація архітектури: Дозволяє ясно і зрозуміло відобразити, як фізично організована система або програмний продукт.
2. Планування розгортання: Допомогає розробникам і архітекторам планувати, як компоненти програми будуть розташовані на серверах та іншому обладнанні.
3. Комунікація: Забезпечує ефективну комунікацію між розробниками, архітекторами та іншими учасниками проекту щодо фізичної архітектури системи.
4. Документація: Створює цінну документацію, яка може використовуватися для розуміння і підтримки системи протягом її життєвого циклу.
5. Виявлення проблем: Допомогає виявляти потенційні проблеми з розгортанням програмної системи ще до її впровадження.

В діаграмі розгортання для Unity проекту (рис. 2.10) показані основні компоненти, такі як Unity Editor та комп'ютер, на якому буде виконуватись

		Е.Е. СОІН			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Р.В. ПЕТРОСЯН				
Змн.	Арк.	№ докум.	Підпис	Дата		38

проект.

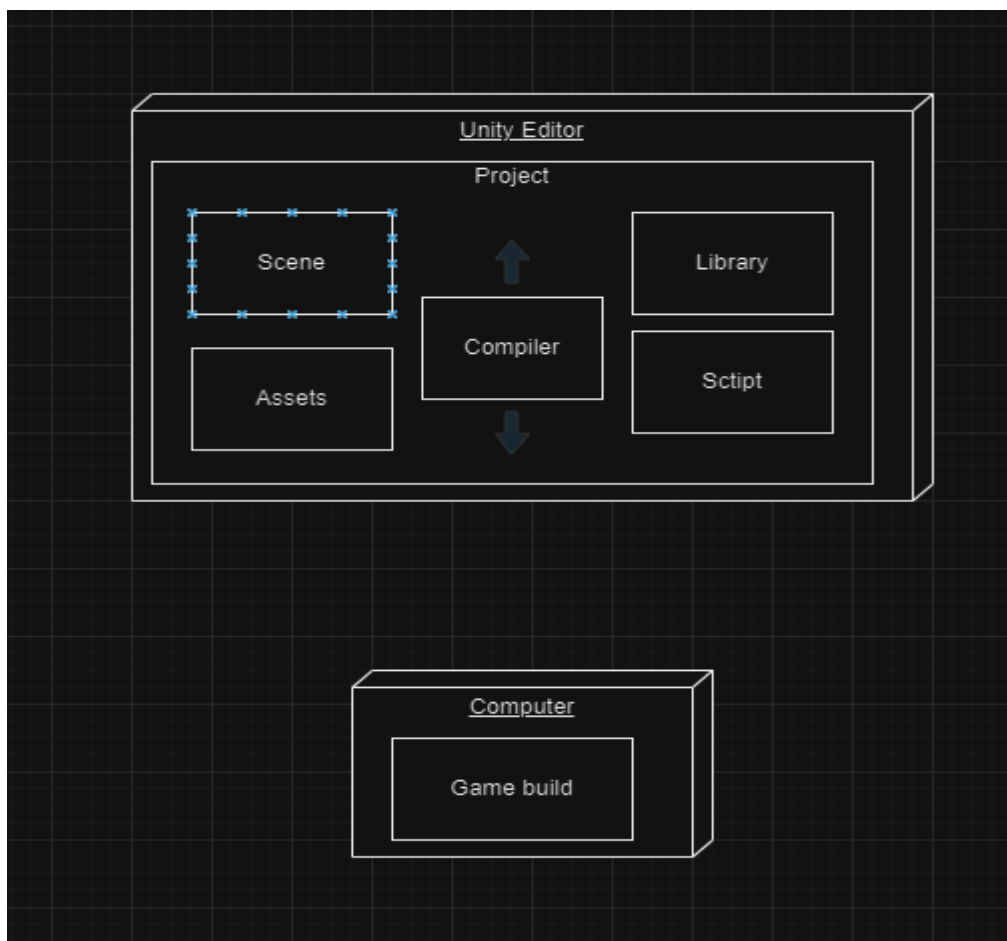


Рисунок 2.10 – Діаграма розгортання.

Відображена діаграма ілюструє, як різні складові проекту, такі як сцени, асети, компілятори та бібліотеки, проходять процес збірки в Unity Editor і перетворюються на готову гру на комп'ютері. Ця візуалізація відіграє важливу роль у конфігурації та ефективному впровадженні проекту на платформі, допомагаючи зрозуміти ключові компоненти проекту та їх взаємодію під час процесу розгортання.

Такий підхід дозволяє не лише чітко налаштувати проект для роботи на різних платформах, а й сприяє вдосконаленню усіх аспектів від розробки до завершення.

2.2 Використання компонентів під час розробки гри

Першим та одним з найважливіших компонентів був використаний Nav Mesh Agent

Nav Mesh Agent в Unity[10] - це компонент, який призначений для

керування автоматичним переміщенням ігрових об'єктів у тривимірному середовищі. Він використовується в поєднанні з NavMesh, що є навігаційною сіткою, оптимізованою для швидкого та ефективного розрахунку шляхів руху персонажів.

Основні характеристики та функції Nav Mesh Agent в Unity[11]:

1. Автоматичне переміщення

Nav Mesh Agent дозволяє ігровим об'єктам автоматично знаходити оптимальний шлях до цілі, враховуючи поточне положення, перешкоди та топографію сцени, задану в NavMesh.

2. Інтеграція з NavMesh

NavMesh представляє собою навігаційну сітку, яка визначає області, доступні для переміщення персонажів. Nav Mesh Agent використовує цю інформацію для обчислення шляхів і переміщення.

3. Контроль параметрів руху

Компонент Nav Mesh Agent забезпечує керування швидкістю переміщення, радіусом виявлення перешкод, точністю навігації та іншими параметрами, які впливають на поведінку персонажа.

4. Адаптація до динамічних змін

Nav Mesh Agent здатен адаптуватися до змін у сцені, таких як переміщення або додавання перешкод, автоматично перераховуючи шлях для обходу нових перепон.

5. Використання в штучному інтелекті

Nav Mesh Agent є основним інструментом для реалізації штучного інтелекту (AI) у іграх, дозволяючи NPC та іншим персонажам автоматично переміщатися по сцені та взаємодіяти з навколишнім світом. Nav Mesh Agent в Unity полегшує процес створення та керування складною поведінкою персонажів, забезпечуючи їм можливість навігації по сцені з мінімальним втручанням розробника.

Також ще один компонент, який був використаний, це Rb

В Unity компонент Rigidbody[12] відіграє ключову роль у додаванні

		Е.Е. СОІН			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Р.В. ПЕТРОСЯН				
Змн.	Арк.	№ докум.	Підпис	Дата		40

фізичної поведінки ігровим об'єктам. Коли ви хочете, щоб об'єкт реагував на сили, такі як гравітація або зіткнення з іншими об'єктами, саме Rigidbody допоможе вам цього досягти. Давайте розглянемо, як працює цей компонент і які можливості він надає.

2.3 Основні етапи реалізації гри

Розробка комп'ютерної гри "Nixton's Adventure" у жанрі Adventure складається з кількох важливих етапів. Спочатку необхідно чітко визначити концепцію гри, її функції та основні характеристики. Це дозволяє програмістам зрозуміти, як будуть реалізовані ігрові механіки та які компоненти потрібні для їх впровадження. Наприклад, якщо гра містить складну фізику або взаємодію об'єктів, слід визначити, які системи Unity, такі як Rigidbody або Collider, будуть використовуватися для досягнення потрібної функціональності.

Далі йде створення та інтеграція ігрових механік, для чого можна скористатися різноманітними засобами і техніками. Unity використовує мову програмування C#, що дозволяє створювати скрипти для контролю поведінки ігрових об'єктів. Процес розпочинається з організації проекту, що включає створення структурованих папок для зберігання скриптів, ігрових моделей, сцен і плагінів. Якщо потрібні додаткові плагіни чи бібліотеки, їх необхідно завантажити та додати до проекту.

Після цього скрипти додаються до відповідних ігрових об'єктів через інтерфейс Unity Editor. Це включає не тільки додавання скриптів, але й налаштування всіх необхідних компонентів, таких як Rigidbody для фізики або Animator для анімації, та конфігурацію їх параметрів. Важливим аспектом роботи з Unity є забезпечення коректної роботи всіх налаштованих компонентів. Це означає, що після створення скриптів та їх інтеграції з ігровими об'єктами необхідно провести ретельне тестування для перевірки функціональності та виявлення можливих помилок.

Налаштування скриптів в Unity Editor також є важливим етапом. Це включає налаштування властивостей скриптів, визначення зв'язків між об'єктами

		Е.Е. СОІН			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Р.В. ПЕТРОСЯН				
Змн.	Арк.	№ докум.	Підпис	Дата		41

та забезпечення коректного реагування на події в грі. Весь процес програмування на Unity є комплексним і багатоетапним, вимагає ретельного планування, налаштування та тестування для створення функціональної та стабільної гри.

Основні характеристики та функції Rigidbody:

1. Маса

– Маса об'єкта визначає, наскільки він інертний. Тяжкі об'єкти важче прискорити або зупинити, тоді як легкі об'єкти легше піддаються впливу сил. Наприклад, при зіткненні легка куля відскочить далі, ніж важкий камінь.

2. Гравітація

– При ввімкненні гравітації на об'єкт починає діяти сила тяжіння, змушуючи його падати вниз. Це дозволяє легко створювати об'єкти, які ведуть себе реалістично, наприклад, падаючі ящики або персонажів, що стрибають.

3. Сили та імпульси

– Ви можете прикладати сили до об'єкта для його прискорення або зміни напрямку руху. Наприклад, щоб запустити об'єкт у повітря, можна застосувати вибухову силу. Імпульси діють аналогічно, але застосовуються миттєво, що корисно для створення ефектів миттєвого прискорення.

4. Демпфування (Damping)

– Демпфування дозволяє зменшувати швидкість об'єкта з часом, імітуючи опір повітря або тертя. Лінійне демпфування сповільнює рух, а кутове — обертання.

5. Колізії та тригери

– Об'єкти з Rigidbody можуть стикатися один з одним, створюючи реалістичні фізичні взаємодії. Тригери дозволяють визначати події, такі як перетин об'єктом межі певної зони.

6. Типи рухів

– Dynamic: Об'єкт повністю підпорядковується фізичним силам і зіткненням.

– Kinematic: Об'єкт керується скриптами, але може взаємодіяти з динамічними об'єктами.

		Е.Е. СОІН			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Р.В. ПЕТРОСЯН				
Змн.	Арк.	№ докум.	Підпис	Дата		42

- Static: Об'єкт нерухомий і не піддається фізичним силам.

7. Обмеження (Constraints)

- Обмеження дозволяють фіксувати рух або обертання об'єкта по певних осях. Це корисно, коли потрібно, щоб об'єкт рухався тільки в одному напрямку або залишався нерухомим навколо однієї осі.

Приклади використання Rigidbody

1. Фізично коректні об'єкти

- Додавши Rigidbody до ігрового об'єкта, ви зробите його підвладним фізичним силам. Наприклад, м'ячик з Rigidbody буде відскакувати від стін і підлоги, а камінь буде падати і котитися вниз по схилу.

2. Управління персонажем

- Rigidbody можна використовувати для створення персонажів, які можуть взаємодіяти з навколишнім середовищем. Наприклад, персонаж буде падати з платформ, штовхати об'єкти та стикатися з перешкодами.

3. Симуляції та ефекти

- Rigidbody дозволяє створювати реалістичні симуляції, такі як катящіся камені, падаючі ящики і вибухи. Це додає глибину та реалізм вашому ігровому світу.

Одним з основних класів був `PlayerMoving` він був створений першим. Клас `PlayerMoving` [13] відповідає за керування пересуванням персонажа у грі, створений на Unity. Він забезпечує рух, стрибки, перевірку стану землі та керування швидкістю. Основні функції цього класу можна розділити на кілька ключових аспектів:

Опис компонентів

[Header("References")]

_orientation: Посилання на об'єкт, який визначає напрямок руху гравця.

[Header("Keybinds")]

_jumpKey: Клавіша для стрибка (за замовчуванням пробіл).

[Header("Move")]

_Speed: Швидкість руху гравця.

_realSpeed: Фактична швидкість (не використовується в коді).

_FastSpeed: Швидкість під час бігу.

[Header("Jump")]

		Е.Е. СОІН			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Р.В. ПЕТРОСЯН				
Змн.	Арк.	№ докум.	Підпис	Дата		43

_jumpForce: Сила стрибка.
 _jumpCooldown: Затримка між стрибками.
 _airMultiplier: Множник швидкості руху в повітрі.
 _canJump: Прапорець, який визначає, чи може гравець стрибати.
 [Header("Ground")]
 _distanceToGround: Відстань до землі.
 _groundGrag: Опір на землі.
 _groundLayer: Шар землі.
 _grounded: Прапорець, який визначає, чи стоїть гравець на землі.
 Основні змінні
 _horizontal: Зберігає горизонтальний ввід (A/D або стрілки).
 _vertical: Зберігає вертикальний ввід (W/S або стрілки).
 _moveDirection: Напрямок руху гравця.
 _rigidbody: Компонент Rigidbody, що додається до об'єкта гравця.

Методи

Метод Start(), який виконується один раз при запуску скрипта. Він викликає методи для отримання компонентів і початкових налаштувань.

```
private void Start()
{
    GetComponents();
    Settings();
}
```

Метод FixedUpdate(), який виконується з постійною частотою і відповідає за фізичні розрахунки. Викликає метод для руху гравця.

```
private void FixedUpdate()
{
    MovePlayer();
}
```

Метод Update(), який виконується кожен кадр. Він перевіряє стан землі, обробляє ввід користувача і контролює швидкість.

```
private void Update()
{
    CheckGround();
    SetInput();
    SpeedController();
}
```

Метод SetInput() для обробки вводу користувача. Перевіряє, чи натиснуто клавішу стрибка, і виконує стрибок, якщо це можливо.

```
private void SetInput()
```

		Е.Е. СОІН			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Р.В. ПЕТРОСЯН				
Змн.	Арк.	№ докум.	Підпис	Дата		44

```

{
    _horizontal = Input.GetAxisRaw("Horizontal");
    _vertical = Input.GetAxisRaw("Vertical");

    if (Input.GetKey(_jumpKey) && _canJump && _grounded)
    {
        _canJump = false;

        Jump();

        Invoke(nameof(ResetJump), _jumpCooldown);
    }
}

```

Метод MovePlayer() для руху гравця. Додає силу до Rigidbody залежно від того, чи стоїть гравець на землі або в повітрі.

```

private void MovePlayer()
{
    _moveDirection = _orientation.forward * _vertical + _orientation.right * _horizontal;

    if (_grounded)
    {
        if (Input.GetKey(KeyCode.LeftShift))
        {
            _rigidbody.AddForce(_moveDirection.normalized * _FastSpeed * 10f, ForceMode.Force);
        }
        else
        {
            _rigidbody.AddForce(_moveDirection.normalized * _Speed * 10f, ForceMode.Force);
        }
    }
    else
    {
        _rigidbody.AddForce(_moveDirection.normalized * _Speed * 10f * _airMultiplier, ForceMode.Force);
    }
}

```

Метод SpeedController() для контролю швидкості гравця. Обмежує максимальну швидкість руху.

```

private void SpeedController()
{
    Vector3 flatValue = new Vector3(_rigidbody.velocity.x, 0f, _rigidbody.velocity.z);

    if (flatValue.magnitude > _Speed)

```

		Е.Е. СОІН			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Р.В. ПЕТРОСЯН				
Змн.	Арк.	№ докум.	Підпис	Дата		45

```

{
    Vector3 limitedValue = flatValue.normalized * _Speed;
    _rigidbody.velocity = new Vector3(limitedValue.x, _rigidbody.velocity.y, limitedValue.z);
}
}

```

Метод **Jump()** для стрибка гравця. Додає вертикальну силу до Rigidbody.

```

private void Jump()
{
    _rigidbody.velocity = new Vector3(_rigidbody.velocity.x, 0f, _rigidbody.velocity.z);

    _rigidbody.AddForce(transform.up * _jumpForce, ForceMode.Impulse);
}

```

Метод **ResetJump()** для скидання можливості стрибка після затримки.

```

private void ResetJump()
{
    _canJump = true;
}

```

Метод **CheckGround()** для перевірки стану землі. Виконує перевірку за допомогою Raycast.

```

private void CheckGround()
{
    _grounded = Physics.Raycast(
        transform.position, Vector3.down, _distanceToGround, _groundLayer);

    if (_grounded)
        _rigidbody.drag = _groundGrag;
    else
        _rigidbody.drag = 0f;
}

```

Метод **Settings()** для початкових налаштувань.

```

private void Settings()
{
    _canJump = true;
}

```

Метод **GetComponents()** для отримання компонентів, необхідних для роботи скрипта.

```

private void GetComponents()
{
    _rigidbody = GetComponent<Rigidbody>();
    _rigidbody.freezeRotation = true;
}

```

Діаграму активності пересування головного герою надано на (рис. 2.11)

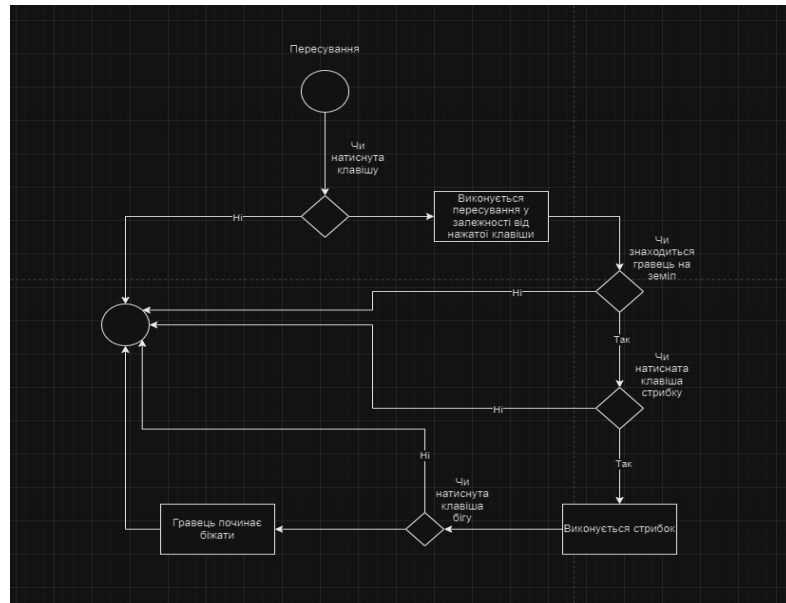


Рисунок 2.11 – Діаграма активності пересування головного герою.

Також був написаний клас Inventory який відповідає за управління інвентарем гравця в ігровому середовищі Unity. Основними функціями цього класу є управління предметами, які гравець може зібрати, їх відображення у вікні інвентаря, а також можливість взаємодії з ними через клавіші на клавіатурі.

Основні компоненти і функції класу:

1. Змінні класу:

item: Список предметів у інвентарі.[\[14\]](#)

DataBase: Посилання на базу даних ігрових предметів.

cellContainer: Контейнер для відображення предметів у вікні інвентаря.

ShowInvenotry: Клавіша для показу/приховування інвентаря.

TakeButton: Клавіша для збирання предметів.

Point: Графічний об'єкт, що використовується для відображення курсора гравця.

2. Методи ініціалізації та управління:

а. Start(): Ініціалізує початкові умови інвентаря, включаючи приховання об'єктів, налаштування змінних та налаштування слухачів подій.

```

void Start()
{
    toolTypeObj.SetActive(false);
}
  
```

```

item = new List<Item>();

cellContainer.SetActive(false);
for (int i = 0; i < cellContainer.transform.childCount; i++)
{
    cellContainer.transform.GetChild(i).GetComponent<CurrentItem>().index = i;
}

for (int i = 0; i < cellContainer.transform.childCount; i++)
{
    item.Add(new Item());
}
}

```

б. **Update():** Оновлює інвентар, перевіряє натискання клавіш, щоб зібрати предмети, і взаємодіє з ними.

```

void Update()
{
    ToggleInventory();
    if (Input.GetKeyDown(TakeButton))
    {
        Ray ray = Camera.main.ViewportPointToRay(new Vector3(0.5f, 0.5f, 0.5f));
        RaycastHit hit;
        if (Physics.Raycast(ray, out hit, 10f))
        {
            if (hit.collider.GetComponent<Item>())
            {
                AddItem(hit.collider.GetComponent<Item>());
                Item currentItem = hit.collider.GetComponent<Item>();
                Message(currentItem);
            }
        }
    }
}

```

3. Додаткові функції:

а. **Message():** Відображає повідомлення про зібраний предмет.

```

void Message(Item currentItem)
{
    GameObject msgObj = Instantiate(Message);
    msgObj.transform.SetParent(MessageManager.transform);

    Image msgImage = msgObj.transform.GetChild(0).GetComponent<Image>();
    msgImage.sprite = currentItem.Icon;
}

```

		Е.Е. СОІН			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Р.В. ПЕТРОСЯН				
Змн.	Арк.	№ докум.	Підпис	Дата		48

```
TMP_Text msgText = msgObj.transform.GetChild(1).GetComponent<TMP_Text>();
msgText.text = currentItem.nameItem;
```

```
}
```

b. **AddItem():** Додає зібраний предмет до інвентаря, враховуючи його тип (стековий або нестековий).

```
void AddItem(Item currentItem)
{
    if (currentItem.isStackable == true)
    {
        AddStackableItem(currentItem);
    }
    else
    {
        AddUnstakableItem(currentItem);
    }
}
```

c. **ToggleInventory():** Показує або приховує вікно інвентаря за допомогою відповідної клавіші.

```
void ToggleInventory()
{
    if (Input.GetKeyDown>ShowInvenotry))
    {
        if (cellContainer.activeSelf)
        {
            cellContainer.SetActive(false);
            Time.timeScale = 1;
            Cursor.visible = false;
            Point.SetActive(true);
        }
        else
        {
            cellContainer.SetActive(true);
            Time.timeScale = 0;
            Cursor.visible = true;
            Cursor.lockState = CursorLockMode.None;
            Point.SetActive(false);
        }
    }
}
```

d. **DisplayItems():** Оновлює візуальне відображення предметів у вікні

		Е.Е. СОИН			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Р.В. ПЕТРОСЯН				
Змн.	Арк.	№ докум.	Підпис	Дата		49

інвентаря, з урахуванням їх кількості та зображення.

```
public void DisplayItems()
{
    for (int i = 0; i < item.Count; i++)
    {
        Transform cell = cellContainer.transform.GetChild((i));
        Transform icon = cell.GetChild(0);
        Transform count = icon.GetChild(0);
        TMP_Text txt = count.GetComponent<TMP_Text>();
        Image img = icon.GetComponent<Image>();
        if (item[i].id != 0)
        {
            img.enabled = true;
            img.sprite = item[i].Icon;
            if (item[i].countItem > 1)
            {
                txt.text = item[i].countItem.ToString();
            }
            else
            {
                txt.text = null;
            }
        }
        else
        {
            img.enabled = false;
            img.sprite = null;
            txt.text = null; }
    }
}
```

4. Методи роботи з предметами:

а. AddUnstakableItem() та AddStackableItem(): Додають нестекові та стекові предмети в інвентар.

```
void AddUnstakableItem(Item currentItem)
{
    for (int i = 0; i < item.Count; i++)
    {
        if (item[i].id == 0)
        {
            item[i] = currentItem;
            item[i].countItem = 1;
            DisplayItems();
            Destroy(currentItem.gameObject);
            break;
        }
    }
}
```

		Е.Е. СОИН			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Р.В. ПЕТРОСЯН				
Змн.	Арк.	№ докум.	Підпис	Дата		50

```

    }
}

```

Клас Inventory використовується для створення та управління функціями інвентаря у грі, забезпечуючи збір, відображення і взаємодію з ігровими предметами через інтерфейс користувача. Діаграма активності інвентарю вказана на (Рис.2.12)

Також щоб створити сам інвентар нам потрібно зробити префаб (Prefab) ячеек для того щоб їх потім можна було знову визвати.

Префаб (Prefab) у Unity — це один з основних інструментів для розробників, який дозволяє створювати, зберігати та багаторазово використовувати складені об'єкти. Замість того, щоб кожного разу заново створювати ячейки інвентарю, можна зберегти його як префаб і потім вставляти у свій проект скільки завгодно разів.

Коли ви створюєте префаб, ви по суті зберігаєте об'єкт разом з усіма його компонентами та налаштуваннями. Це може бути що завгодно: від простого куба до складного персонажа з анімаціями, фізикою та скриптами. Префаб зберігається у вигляді окремого файлу в проекті, і ви можете просто перетягнути його на сцену, щоб створити новий екземпляр цього об'єкта.

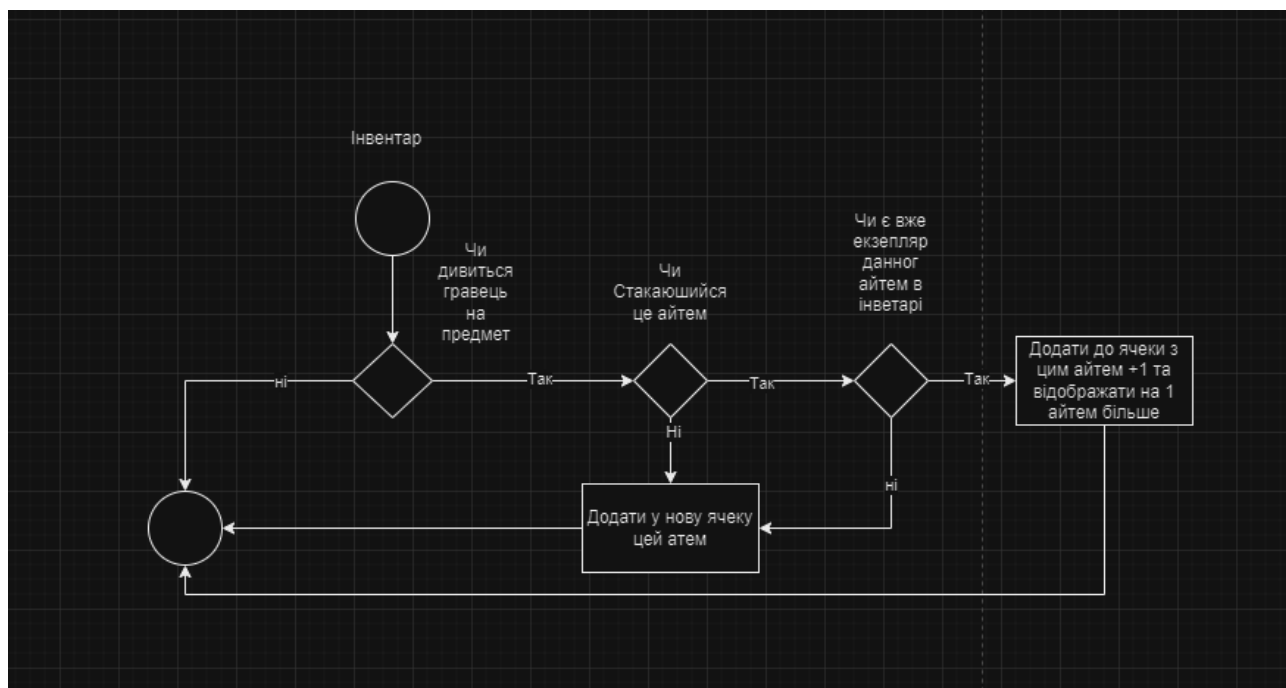


Рисунок 2.12 – Діаграма активності для інвентарю.

До класу Inventory також відноситься клас CurrentItem який відповідає за

		Е.Е. СОІН			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Р.В. ПЕТРОСЯН				
Змн.	Арк.	№ докум.	Підпис	Дата		51

взаємодію з елементами інвентаря в грі. Основна мета цього класу - обробка кліків на предмети інвентаря та керування предметами під час перетягування (Remove and Drop) в інтерфейсі користувача:

Поля

```
[HideInInspector]
```

```
public int index;
```

```
private GameObject inventoryObj;
```

```
private Inventory inventory;
```

- **index** - індекс поточного предмета в інвентарі.
- **inventoryObj** - об'єкт, який містить компонент інвентаря.
- **inventory** - компонент Inventory, який містить інформацію про всі предмети інвентаря.

```
void Start()
```

```
{
```

```
    inventoryObj = GameObject.FindGameObjectWithTag("InventoryManager");
```

```
    inventory = inventoryObj.GetComponent<Inventory>();
```

```
}
```

- **Start()** метод знаходить об'єкт інвентаря в сцені за тегом InventoryManager і отримує компонент Inventory.

Метод класу OnPointerClick():

```
public void OnPointerClick(PointerEventData eventData)
```

```
{
```

```
    if (eventData.button == PointerEventData.InputButton.Left)
```

```
    {
```

```
        if (inventory.item[index].customEvent != null)
```

```
        {
```

```
            inventory.item[index].customEvent.Invoke();
```

```
        }
```

```
        if (inventory.item[index].isRemoveable)
```

```
        {
```

```
            Remove();
```

```
        }
```

```
    }
```

```
    if (eventData.button == PointerEventData.InputButton.Right)
```

```
    {
```

		Е.Е. СОІН			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Р.В. ПЕТРОСЯН				
Змн.	Арк.	№ докум.	Підпис	Дата		52

```

        if (inventory.item[index].isDropped)
        {
            Drop();
            Remove();
        }
    }
}

```

– OnPointerClick метод обробляє кліки на предмети інвентаря.

– Ліва кнопка миші: якщо предмет має подію customEvent, ця подія викликається. Якщо предмет можна видалити (isRemoveable), викликається метод Remove().

– Права кнопка миші: якщо предмет може бути викинутий (isDropped), викликається метод Drop() і потім Remove().

Метод класу Remove():

```

public void Remove()
{
    if (inventory.item[index].id != 0)
    {
        if (inventory.item[index].countItem > 1)
        {
            inventory.item[index].countItem--;
        }
        else
        {
            inventory.item[index] = new Item();
        }

        inventory.DisplayItems();
    }
}

```

– Remove метод видаляє один екземпляр предмета з інвентаря або повністю видаляє предмет, якщо кількість екземплярів цього предмета дорівнює 1. Потім оновлює відображення інвентаря за допомогою inventory.DisplayItems().

Метод класу Drop():

```

public void Drop()
{
    if (inventory.item[index].id != 0)
    {
        for (int i = 0; i < inventory.DataBase.transform.childCount; i++)

```

		Е.Е. СОІН			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Р.В. ПЕТРОСЯН				53
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        {
            Item item = inventory.DataBase.transform.GetChild(i).GetComponent<Item>();
            if (item)
            {
                if (inventory.item[index].id == item.id)
                {
                    GameObject droppedObj = Instantiate(item.gameObject);
                    droppedObj.transform.position = Camera.main.transform.position +
Camera.main.transform.forward * 6;
                }
            }
        }
    }
}

```

– Drop метод викидає предмет з інвентаря в світ гри. Він знаходить відповідний об'єкт у базі даних предметів і створює його копію в світі перед камерою.

Метод класу OnDrop():

```

public void OnDrop(PointerEventData eventData)
{
    GameObject draggedObj = Drag.draggedObj;
    if (draggedObj == null)
    {
        return;
    }

    CurrentItem currentdraggeItem = draggedObj.GetComponent<CurrentItem>();

    if (currentdraggeItem != null)
    {
        Item currentItem = inventory.item[GetComponent<CurrentItem>().index];
        inventory.item[GetComponent<CurrentItem>().index] = inventory.item[currentdraggeItem.index];
        inventory.item[currentdraggeItem.index] = currentItem;
        inventory.DisplayItems();
    }
}

```

– OnDrop метод обробляє перетягування предметів в інвентарі. Він обмінює предмети місцями між слотом, куди був перетягнутий предмет, і початковим слотом перетягнутого предмета. Після обміну викликається метод inventory.DisplayItems(), щоб оновити відображення інвентаря.

		Е.Е. СОІН			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Р.В. ПЕТРОСЯН				
Змн.	Арк.	№ докум.	Підпис	Дата		54

Клас Drag також відноситься до класу Inventory , він відповідає за обробку подій перетягування (drag and drop) в інвентарі гри. Клас реалізує інтерфейси IBeginDragHandler, IDragHandler і IEndDragHandler з UnityEngine.EventSystems, що дозволяють визначити поведінку під час початку, процесу та завершення перетягування.

Поля

```
public static GameObject draggedObj;
```

```
private Inventory inventory;
```

– draggedObj - статичне поле, що зберігає посилання на об'єкт, який наразі перетягується.

– inventory - посилання на компонент Inventory, що використовується для доступу до інвентаря у грі та буде використаний в дальнішому.

Метод класу Start():

```
void Start()
{
    inventory = GameObject.FindGameObjectWithTag("InventoryManager").GetComponent<Inventory>();
}
```

Метод Start() знаходить об'єкт з тегом InventoryManager і отримує з нього компонент Inventory, щоб взаємодіяти з інвентарем.

Метод класу OnBeginDrag():

```
public void OnBeginDrag(PointerEventData eventData)
{
    draggedObj = gameObject;
    inventory.DragPrefab.SetActive(true);
    inventory.DragPrefab.GetComponent<CanvasGroup>().blocksRaycasts = false;

    if (draggedObj.GetComponent<CurrentItem>() != null)
    {
        int index = draggedObj.GetComponent<CurrentItem>().index;
        inventory.DragPrefab.GetComponent<Image>().sprite = inventory.item[index].Icon;
        if (inventory.item[index].countItem > 1)
        {
            inventory.DragPrefab.transform.GetChild(0).GetComponent<TMP_Text>().text =
            inventory.item[index].countItem.ToString();
        }
        else
        {

```

		Е.Е. СОІН			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Р.В. ПЕТРОСЯН				
Змн.	Арк.	№ докум.	Підпис	Дата		55

```

        inventory.DragPrefab.transform.GetChild(0).GetComponent<TMP_Text>().text = null;
    }

    if (inventory.DragPrefab.GetComponent<Image>().sprite == null)
    {
        draggedObj = null;
        inventory.DragPrefab.SetActive(false);
    }
}
}

```

– **OnBeginDrag** метод викликається, коли починається перетягування об'єкта.

- Зберігає перетягуваний об'єкт у статичному полі `draggedObj`.
- Активує `DragPrefab`, що представляє іконку перетягнутого об'єкта.
- Встановлює зображення та кількість предмета в `DragPrefab` на основі даних з інвентаря.

- Вимикає перетягування, якщо об'єкт не має спрайта.

Метод класу **OnDrag()**:

```

public void OnDrag(PointerEventData eventData)
{
    inventory.DragPrefab.transform.position = Input.mousePosition;
}

```

- **OnDrag** метод викликається під час перетягування об'єкта.
- Встановлює позицію `DragPrefab` відповідно до позиції миші.

Метод класу **OnEndDrag(PointerEventData eventData)**:

```

public void OnEndDrag(PointerEventData eventData)
{
    if (!EventSystem.current.IsPointerOverGameObject())
    {
        draggedObj.GetComponent<CurrentItem>().Drop();
        draggedObj.GetComponent<CurrentItem>().Remove();
    }
    draggedObj = null;
    inventory.DragPrefab.SetActive(false);
    inventory.DragPrefab.GetComponent<CanvasGroup>().blocksRaycasts = true;
}

```

- **OnEndDrag** метод викликається, коли перетягування об'єкта завершується.

		Е.Е. СОІН			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Р.В. ПЕТРОСЯН				
Змн.	Арк.	№ докум.	Підпис	Дата		56

- Якщо об'єкт не над інтерфейсом користувача (GUI), викликає методи Drop() та Remove() у компонента CurrentItem, що дозволяє викинути предмет зі слота і видалити його з інвентаря.
- Скидає draggedObj на null.
- Вимикає DragPrefab і дозволяє знову використовувати промені взаємодії (blocksRaycasts = true).

Цей клас забезпечує повну функціональність перетягування предметів в інвентарі. Від початку перетягування, через процес перетягування та до завершення перетягування, він дозволяє користувачам взаємодіяти з предметами в інвентарі, переміщуючи їх між слотами або викидаючи їх зі свого інвентаря. Діаграму активності наведено на (рис. 2.13).

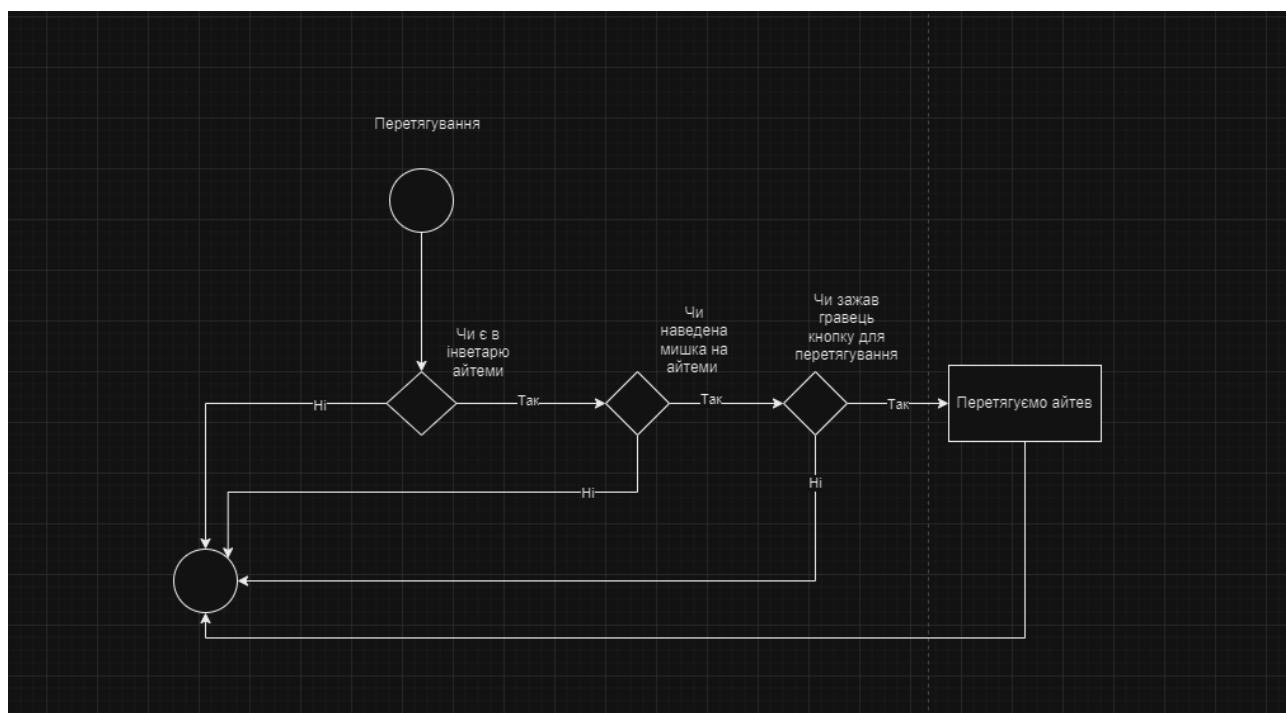


Рисунок 2.13 – Діаграма активності для Перетягування айтемів в інвентарі.

Також клас Item визначає структуру для зберігання даних про предмети в грі. Клас використовується для створення предметів, що будуть зберігатися в інвентарі гравця. Кожен предмет має кілька властивостей, які описують його характеристики та поведінку.

```

public string nameItem;
public int id;
public int countItem;
public bool isStackable;

```

[Multiline(5)]

public string descriptionItem;

public bool isRemoveable;

public bool isDropped;

public Sprite Icon;

public UnityEvent customEvent;

– nameItem - рядок, що зберігає назву предмета. Це ім'я буде відображатися в інтерфейсі користувача.

– id - унікальний ідентифікатор предмета, який використовується для відрізнєння його від інших предметів.

– countItem - ціле число, що вказує на кількість цього предмета. Використовується для предметів, які можна складати (стекувати).

– isStackable - булевий прапорець, що визначає, чи може предмет складатися у стеки (тобто чи можна зберігати декілька таких предметів в одному слоті інвентаря).

– descriptionItem - рядок з описом предмета. Поле з атрибутом [Multiline(5)] дозволяє вводити багаторядковий текст в інспекторі Unity.

– isRemoveable - булевий прапорець, що вказує, чи може предмет бути видалений з інвентаря.

– isDropped - булевий прапорець, що вказує, чи може предмет бути викинутий з інвентаря.

– Icon - поле типу Sprite, що зберігає іконку предмета. Іконка використовується для відображення предмета в інтерфейсі інвентаря.

– customEvent - поле типу UnityEvent, яке дозволяє визначити кастомну подію для предмета. Це може бути будь-яка дія, яку можна викликати, коли взаємодіють з предметом.

Також був реалізований клас ToolType у цьому коді відповідає за відображення підказки (tooltip) з інформацією про предмети в інвентарі, коли гравець наводить на них курсор миші(наприкладі виводить назву та опис предмету). Це допомагає гравцю отримати детальну інформацію про предмет без необхідності використовувати його чи відкривати додаткові вікна.

Поля класу:

		Е.Е. СОІН			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Р.В. ПЕТРОСЯН				
Змн.	Арк.	№ докум.	Підпис	Дата		58

```
private Inventory inventory
```

Метод класу OnDisable():

```
private void OnDisable()
{
    inventory.toolTypeObj.SetActive(false);
}
```

Цей метод викликається, коли об'єкт деактивується. Він приховує об'єкт підказки.

Метод класу Start() викликається один раз при створенні об'єкту:

```
void Start()
{
    inventory = GameObject.FindGameObjectWithTag("InventoryManager").GetComponent<Inventory>();
}
```

У методі Start відбувається пошук об'єкта з тегом "InventoryManager" та отримання компонента Inventory для подальшої роботи з ним.

Метод класу Update() викликається один раз за кожен кадр:

```
void Update()
{
    inventory.toolTypeObj.transform.position = new Vector2(Input.mousePosition.x + inventory.offsetX,
Input.mousePosition.y + inventory.offsetY);
}
```

Метод Update постійно оновлює позицію об'єкта підказки, щоб він слідував за курсором миші з заданим зміщенням (offsetX та offsetY).

Метод класу OnPointerEnter(PointerEventData eventData):

```
public void OnPointerEnter(PointerEventData eventData)
{
    CurrentItem currentItem = GetComponent<CurrentItem>();
    Item item = inventory.item[currentItem.index];

    if (item.id != 0)
    {
        inventory.toolTypeObj.SetActive(true);
        inventory.icon.sprite = item.Icon;
        inventory.itemName.text = item.nameItem;
        inventory.itemDescription.text = item.descriptionItem;
    }
}
```

Цей метод викликається, коли курсор миші входить в область об'єкта, до якого прикріплений цей скрипт. Він отримує предмет інвентаря на поточному

		Е.Е. СОІН			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Р.В. ПЕТРОСЯН				59
Змн.	Арк.	№ докум.	Підпис	Дата		

індексі та відображає його інформацію у підказці, якщо предмет існує(його id не дорівнює нулю).

Метод класу OnPointerExit(PointerEventData eventData):

```
public void OnPointerExit(PointerEventData eventData)
{
    inventory.toolTypeObj.SetActive(false);
}
```

Цей метод викликається, коли курсор миші залишає область об'єкта. Він приховує об'єкт підказки.

Клас ToolType призначений для забезпечення функціональності відображення підказок для предметів в інвентарі. Коли гравець наводить курсор на предмет, підказка з'являється біля курсору, показуючи зображення предмета, його назву та опис. Коли курсор залишає область предмета, підказка зникає. Це покращує взаємодію користувача з інвентарем, надаючи йому швидкий доступ до інформації про предмети.

Клас

PatrolBehaviour[[15](#)]

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

public class PatrolBehaviour : StateMachineBehaviour
{
    private float timer;
    private List<Transform> points = new List<Transform>();
    private NavMeshAgent agent;
    private Transform player;
    private float chaseRange = 10;
    override public void OnStateEnter(Animator animator, AnimatorStateInfo stateInfo, int layerIndex)
    { timer = 0;
      Transform pointsObj = GameObject.FindGameObjectWithTag("Points").transform;
      foreach (Transform t in pointsObj)
      { points.Add(t); }
      agent = animator.GetComponent<NavMeshAgent>();
      agent.SetDestination(points[0].position);
      player = GameObject.FindGameObjectWithTag("Player").transform; }
```

```

override public void OnStateUpdate(Animator animator, AnimatorStateInfo stateInfo, int layerIndex)
{ if (agent.remainingDistance <= agent.stoppingDistance)
  { agent.SetDestination(points[Random.Range(0, points.Count)].position); }
  timer += Time.deltaTime;
  if (timer > 10)
  { animator.SetBool("IsPatrolling", false); }

  float distance = Vector3.Distance(animator.transform.position, player.position);
  if (distance < chaseRange)
  { animator.SetBool("IsChasing", true); } }

override public void OnStateExit(Animator animator, AnimatorStateInfo stateInfo, int layerIndex)
{ agent.SetDestination(agent.transform.position); }
}

```

Цей клас реалізує поведінку патрулювання для ворога. Він використовує механізм станів (State Machine) та NavMeshAgent для руху ворога по визначених точках.

– OnStateEnter: Викликається, коли ворог входить у стан патрулювання. Ініціалізує таймер, завантажує точки патрулювання з об'єкта з тегом "Points", встановлює першу точку призначення для агента та отримує трансформ гравця.

– OnStateUpdate: Викликається на кожному кадрі оновлення в цьому стані. Якщо агент досягає поточної точки призначення, вибирається нова випадкова точка. Також таймер збільшується з кожним кадром. Якщо таймер перевищує 10 секунд, стан патрулювання завершується. Якщо гравець підходить ближче до ворога, ніж на 10 одиниць, ворог переходить у стан переслідування.

– OnStateExit: Викликається, коли ворог виходить зі стану патрулювання. Зупиняє агента.

Цей клас дозволяє ворогу переміщатися між кількома точками на карті, поки не закінчиться таймер або поки гравець не потрапить у зону виявлення.

Клас LookAtPlayer

```

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

```

```

public class LookAtPlayer : MonoBehaviour

```

		Е.Е. СОІН			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Р.В. ПЕТРОСЯН				
Змн.	Арк.	№ докум.	Підпис	Дата		61

```

{
    public Transform camera;
    // Update is called once per frame
    void LateUpdate()
    {
        transform.LookAt(camera);
    }
}

```

Цей клас відповідає за те, щоб об'єкт постійно повертався до гравця. В основному, він використовується для ворогів або об'єктів, які повинні завжди дивитися на камеру гравця.

– **LateUpdate:** Викликається після Update і обробляє всі необхідні обчислення для повороту об'єкта до камери гравця.

Цей клас забезпечує, що об'єкт завжди повернутий до гравця, створюючи ефект спостереження або уваги.

Клас IdleBehaviour

```

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

public class IdleBehaviour : StateMachineBehaviour
{
    private float timer;

    private Transform player;
    private float chaseRange = 10;
    override public void OnStateEnter(Animator animator, AnimatorStateInfo stateInfo, int layerIndex)
    {
        timer = 0;
        player = GameObject.FindGameObjectWithTag("Player").transform;
    }

    override public void OnStateUpdate(Animator animator, AnimatorStateInfo stateInfo, int layerIndex)
    {
        timer += Time.deltaTime;
        if (timer > 5)
        {
            animator.SetBool("IsPatrolling", true);
        }

        float distance = Vector3.Distance(animator.transform.position, player.position);
    }
}

```

		Е.Е. СОІН			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Р.В. ПЕТРОСЯН				62
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        if (distance < chaseRange)
        {
            animator.SetBool("IsChasing", true);
        }
    }
}

```

```

override public void OnStateExit(Animator animator, AnimatorStateInfo stateInfo, int layerIndex)
{

}

}

```

Цей клас реалізує стан очікування для ворога. Він дозволяє ворогу стояти на місці, перш ніж перейти до патрулювання або переслідування гравця.

- OnStateEnter: Ініціалізує таймер та знаходить трансформ гравця.
- OnStateUpdate: Збільшує таймер на кожному кадрі. Якщо таймер перевищує 5 секунд, ворог переходить у стан патрулювання. Якщо гравець підходить ближче до ворога, ніж на 10 одиниць, ворог починає переслідування.
- OnStateExit: Викликається, коли ворог виходить зі стану очікування.

Цей клас дозволяє ворогу перебувати в стані очікування на місці, перш ніж розпочати патрулювання або переслідування гравця.

Клас EnemyScript

```

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

public class EnemyScript : MonoBehaviour
{
    private int HP = 100;

    public Animator animator;

    public Slider healthBar;

    public GameObject Sword;
    // Update is called once per frame
    void Update()
    {

```

		Е.Е. СОІН			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Р.В. ПЕТРОСЯН				
Змн.	Арк.	№ докум.	Підпис	Дата		63

```

        healthBar.value = HP;

    }

    public void TakeDamage(int damageAmount)
    {
        HP -= damageAmount;
        if (HP <= 0)
        {
            animator.SetTrigger("death");
            GetComponent<Collider>().enabled = false;
            healthBar.gameObject.SetActive(false);
            Sword.GetComponent<Collider>().enabled = false;
        }
        else
        {
            animator.SetTrigger("damage");
        }
    }
}

```

Цей клас керує здоров'ям ворога та відповідає за відображення стану здоров'я та поведінку при пошкодженні або смерті.

- Update: Оновлює значення здоров'я на шкалі здоров'я кожен кадр.
- TakeDamage: Зменшує здоров'я ворога при отриманні шкоди. Якщо здоров'я знижується до нуля або нижче, викликає анімацію смерті, відключає коллайдери та шкалу здоров'я. Якщо здоров'я все ще вище нуля, викликає анімацію пошкодження.

Цей клас забезпечує управління здоров'ям ворога, включаючи відображення стану здоров'я та обробку анімацій при пошкодженні та смерті.

Клас DamageScript

```

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

public class DamageScript : MonoBehaviour
{

```

		Е.Е. СОІН			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Р.В. ПЕТРОСЯН				
Змн.	Арк.	№ докум.	Підпис	Дата		64

```

public int damageCount = 10;

private void OnTriggerEnter(Collider other)
{
    StartCoroutine(FindObjectOfType<HealthBar>().Damage(damageCount));
}
}

```

Цей клас відповідає за нанесення шкоди об'єктам, які входять в його тригерну зону.

– **OnTriggerEnter:** Викликається, коли інший об'єкт входить у тригерну зону цього об'єкта. Викликає корутину **Damage** з **HealthBar**, зменшуючи здоров'я на **damageCount**.

Цей клас дозволяє наносити шкоду іншим об'єктам при зіткненні з цим об'єктом.

Клас ChaseBehaviour

```

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

public class ChaseBehaviour : StateMachineBehaviour
{
    private NavMeshAgent agent;
    private Transform player;
    float attack_range = 2;
    private float Chase_Range = 10;
    override public void OnStateEnter(Animator animator, AnimatorStateInfo stateInfo, int layerIndex)
    {
        agent = animator.GetComponent<NavMeshAgent>();
        agent.speed = 4;

        player = GameObject.FindObjectOfType<PlayerMoving>().transform;
    }

    override public void OnStateUpdate(Animator animator, AnimatorStateInfo stateInfo, int layerIndex)
    {
        agent.SetDestination(player.position);
        float distance = Vector3.Distance(animator.transform.position, player.position);
        if (distance < attack_range)
        {
            animator.SetBool("IsAttack", true);
        }
    }
}

```

		Е.Е. СОІН			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Р.В. ПЕТРОСЯН				
Змн.	Арк.	№ докум.	Підпис	Дата		65

```

    }

    if (distance > 10)
    {
        animator.SetBool("IsChasing", false);
    }
}

override public void OnStateExit(Animator animator, AnimatorStateInfo stateInfo, int layerIndex)
{
    agent.SetDestination(agent.transform.position);
    agent.speed = 2;
}

}

```

Цей клас реалізує поведінку переслідування для ворога.

- **OnStateEnter:** Встановлює початкові параметри при вході в стан переслідування, включаючи збільшення швидкості агента та визначення трансформу гравця.

- **OnStateUpdate:** Оновлює позицію агента для переслідування гравця. Якщо гравець підходить ближче, ніж на 2 одиниці, ворог переходить у стан атаки. Якщо гравець віддаляється більш ніж на 10 одиниць, ворог припиняє переслідування.

- **OnStateExit:** Відновлює початкову швидкість агента та зупиняє його переслідування.

Цей клас дозволяє ворогу переслідувати гравця, змінюючи швидкість та стан в залежності від відстані до гравця.

Клас AttackBeheviour

```

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

public class AttackBeheviour : StateMachineBehaviour
{
    private Transform player;

    override public void OnStateEnter(Animator animator, AnimatorStateInfo stateInfo, int layerIndex)
    {
        player = GameObject.FindObjectOfType<PlayerMoving>().transform;
    }
}

```

		Е.Е. СОІН			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Р.В. ПЕТРОСЯН				
Змн.	Арк.	№ докум.	Підпис	Дата		66

```

    }

    override public void OnStateUpdate(Animator animator, AnimatorStateInfo stateInfo, int layerIndex)
    {
        animator.transform.LookAt(player);
        float distance = Vector3.Distance(animator.transform.position, player.position);
        if (distance > 3)
        {
            animator.SetBool("IsAttack", false);
        }
    }

    override public void OnStateExit(Animator animator, AnimatorStateInfo stateInfo, int layerIndex)
    {

    }

}

```

Цей клас реалізує поведінку атаки для ворога.

- OnStateEnter: Отримує трансформ гравця при вході в стан атаки.
- OnStateUpdate: Ворог постійно повертається до гравця, поки знаходиться в стані атаки. Якщо гравець віддаляється більш ніж на 3 одиниці, ворог припиняє атаку.
- OnStateExit: Викликається, коли ворог виходить зі стану атаки.

Цей клас дозволяє ворогу атакувати гравця, підтримуючи при цьому його орієнтацію на гравця та припиняючи атаку, якщо гравець виходить за межі досяжності.

Клас ActivateStalactite

Цей клас відповідає за активацію сталактитів у грі. Сталактити в цьому контексті є об'єктами, які починають падати, коли гравець наближається до певної області. Ось як працює цей клас:

Змінні класу:

_stalactite: Масив сталактитів, які потрібно активувати.

Метод Awake():

В цьому методі отримуються всі об'єкти сталактитів у сцені і зберігаються у масиві _stalactite.

		Е.Е. СОІН			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Р.В. ПЕТРОСЯН				
Змн.	Арк.	№ докум.	Підпис	Дата		67

Метод OnTriggerEnter(Collider other):

Викликається, коли інший об'єкт з колайдером потрапляє у тригер цього об'єкта.

Перевіряє, чи має об'єкт, який потрапив у тригер, компонент OnTriggerEnter.

Якщо умова виконується, активується кожен сталактит у масиві _stalactite, викликаючи їх метод activeStalactite().

Клас FallingTrap

Цей клас представляє пастку, яка активується, коли гравець натрапляє на неї і призводить до її знищення.

Метод OnCollisionEnter(Collision collision):

Викликається при зіткненні з іншим колайдером.

Перевіряє, чи об'єкт, з яким сталася колізія, має компонент PlayerMoving.

Якщо умова виконується, викликається Destroy(gameObject), що призводить до знищення самої пастки.

Клас ShootArrow

Цей клас відповідає за стрільбу стрілами з кількох точок на сцені.

Змінні класу:

ArrowSpawnPoint, ArrowSpawnPoint1, ..., ArrowSpawnPoint4: Точки, з яких починається рух стріл.

ArrowPrefab: Префаб стріли, який буде використовуватися для створення стріл.

arrow, arrow1, ..., arrow4: Створені екземпляри префабу стріли.

arrowForce: Сила, з якою рухається стріла.

EndPosition, EndPosition1, ..., EndPosition4: Кінцеві позиції, до яких мають дійти стріли.

CanShootArrow: Логічна змінна, що вказує на можливість стрільби.

Метод Start():

В цьому методі створюються екземпляри префабу стріли на кожній з точок початку ArrowSpawnPoint, ..., ArrowSpawnPoint4.

Метод Update():

		Е.Е. СОІН			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Р.В. ПЕТРОСЯН				
Змн.	Арк.	№ докум.	Підпис	Дата		68

Якщо CanShootArrow true, викликається метод shootArrow().

Метод shootArrow():

Перевіряє, чи стріла досягла своєї кінцевої позиції EndPosition.

Якщо так, то стріла деактивується і повертається на початкову позицію.

Інакше стріла рухається в напрямку EndPosition з заданою швидкістю arrowForce.

Клас Stalactite

Цей клас представляє сталактит, який спочатку не рухається (гравітація вимкнена), а після активації стає падати.

Змінні класу:

rb: Компонент Rigidbody, який управляє фізикою об'єкта.

Метод Start():

В цьому методі отримується компонент Rigidbody об'єкта і вимикається гравітація (rb.useGravity = false).

Метод activeStalactite():

Вмикає гравітацію для сталактиту, що призводить до його початку падіння.

Клас TrapWithArrow

Цей клас активує стрільбу стріл з класу ShootArrow, коли гравець натрапляє на пастку.

Змінні класу:

_shootArrow: Посилання на об'єкт класу ShootArrow, який відповідає за стрільбу стріл.

Метод OnCollisionEnter(Collision collision):

Викликається при зіткненні з іншим колайдером.

Перевіряє, чи об'єкт, з яким сталася колізія, має компонент PlayerMoving.

Якщо умова виконується і CanShootArrow в ShootArrow є false, то встановлює CanShootArrow в true.

Клас TrapWithSpike

Цей клас активує пастку з шипами при зіткненні гравця з нею.

Змінні класу:

		Е.Е. СОІН			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Р.В. ПЕТРОСЯН				
Змн.	Арк.	№ докум.	Підпис	Дата		69

Spike: Префаб об'єкта шипів.

spikeActivate: Логічна змінна, що вказує, чи активована пастка з шипами.

Метод OnCollisionEnter(Collision collision):

Викликається при зіткненні з іншим колайдером.

Перевіряє, чи об'єкт, з яким сталася колізія, має компонент PlayerMoving.

Якщо spikeActivate є false, то створюється екземпляр Spike, що відповідає пастці.

Якщо spikeActivate є true, то не відбувається ніяких подій.

Ці класи служать для реалізації різноманітних пасток у грі, які реагують на дії гравця або інших об'єктів у сцені. Вони демонструють основні принципи обробки колізій, активації руху об'єктів, стрільби та інші основи програмування поведінки об'єктів у 3D іграх.

Висновки до другого розділу

Було продемонстровано візуалізацію процесів розробки програмного продукту, де було показано різні типи діаграм, що вказують на етапи та елементи циклу розробки програми. Використані діаграми включають діаграми варіантів використання, діаграми активності, діаграми класів, а також діаграми компонентів і розгортання. Кожна з цих діаграм була докладно описана та пояснена, що дозволило чітко представити процес розробки та взаємозв'язки між різними елементами програмного продукту.

У процесі розробки програмного продукту в Unity було використано кілька ключових компонентів для досягнення функціональності та відтворення очікуваного користувацького досвіду. Використання їх у проекті дозволило досягти бажаних результатів у розробці програмного продукту для Unity.

Було описано основні етапи реалізації гри, включаючи створення гравця та відповідних класів, таких як Реалізація механік гравця та інвентарю, Реалізація ворогів та Створення пасток та простих логічних задач. В роботі були докладно розглянуті всі етапи розробки та ключові скрипти, які були ілюстровані діаграмами.

		Е.Е. СОІН			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Р.В. ПЕТРОСЯН				
Змн.	Арк.	№ докум.	Підпис	Дата		70

РОЗДІЛ 3. ОПИС РОБОТИ З ПРОГРАМНИМ ДОДАТКОМ ТА ЙОГО ТЕСТУВАННЯ

3.1 Порядок встановлення та налаштування системи

Nixton's Adventure — це захоплююча гра жанру Adventure, створена за допомогою потужного ігрового рушія Unity3D. Гра переміщує гравців у чарівний світ, повний загадок, пасток та неймовірних пригод. Завдяки багатій графіці, захоплюючому геймплею та різноманітним завданням, Nixton's Adventure стане незабутньою подорожжю для кожного гравця.

Версія гри буде доступна для операційних систем Windows.

1. Системні вимоги

Перед встановленням гри переконайтеся, що ваш комп'ютер відповідає наступним мінімальним вимогам:

- Операційна система: Windows 7 або новіша версія
- Процесор: Intel Core i3 або еквівалентний
- Оперативна пам'ять: 4 GB RAM
- Відеокарта: DirectX 11 сумісна відеокарта з 1GB відео пам'яті
- Місце на диску: 10 GB вільного місця
- DirectX: Версія 11

2. Завантаження гри

- Перейдіть на веб-сайт, де розміщена гра «Nixton's Adventure».
- Знайдіть кнопку для завантаження гри та натисніть на неї.
- Виберіть версію для Windows і збережіть інсталяційний файл на свій комп'ютер.

3. Встановлення гри

- Відкрийте папку, куди ви завантажили інсталяційний файл.
- Двічі клацніть на інсталяційний файл, щоб запустити процес встановлення.
- Дотримуйтесь інструкцій майстра встановлення:
- Прийміть ліцензійну угоду.

		Е.Е. СОІН			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Р.В. ПЕТРОСЯН				
Змн.	Арк.	№ докум.	Підпис	Дата		71

- Виберіть папку для встановлення гри (за замовчуванням буде запропоновано стандартну папку).
- Натисніть «Далі» і дочекайтеся завершення процесу встановлення.
- Після завершення встановлення натисніть «Готово», щоб закрити майстра встановлення.

3.2 Інтерфейс та порядок роботи з програмним додатком

При запуску гри на користувача чекає простеньке головне меню.

В меню гравець може, розпочати гру натиснувши на кнопку «Play» в якій він перейде на основну сцену або вийти з гри натиснувши кнопку «Exit» (рис. 3.1).

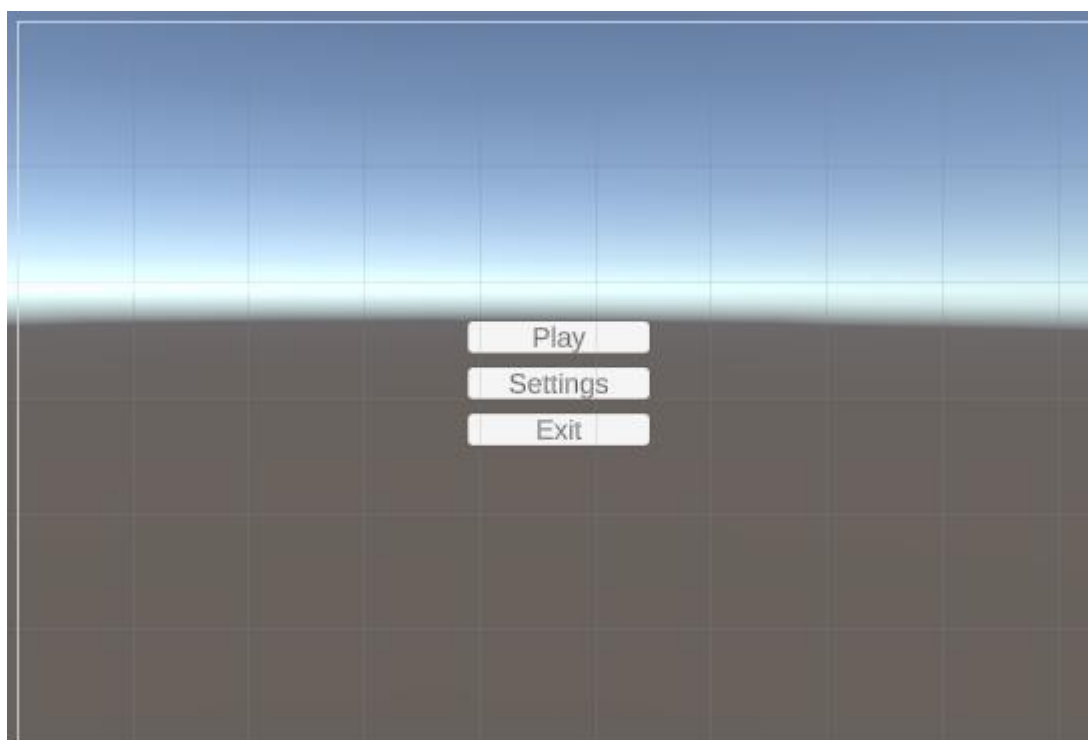


Рисунок 3.1 – Головне меню гри

Після натискання кнопки «Play»

Після того, як гравець натискає кнопку «Play», починається справжня пригода у світі Nixon's Adventure. Відразу після натискання цієї кнопки, гравець потрапляє на хаб-локацію створену за допомогою ProBuilder[16]. Хаб-локація є однією з ключових у грі, оскільки саме з неї гравець розпочинатиме свою подорож після кожного завершеного забігу.

		Е.Е. СОІН			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Р.В. ПЕТРОСЯН				
Змн.	Арк.	№ докум.	Підпис	Дата		72

Хаб-локація: Ваша база перед пригодами

Після завантаження хаб-локації гравець опиняється в затишному будинку Гікстона, який слугує базою та місцем відпочинку між пригодами. Вже на цьому етапі гравець може керувати персонажем і досліджувати будинок. Інтерфейс гри змінюється, відкриваючи нові елементи та можливості.

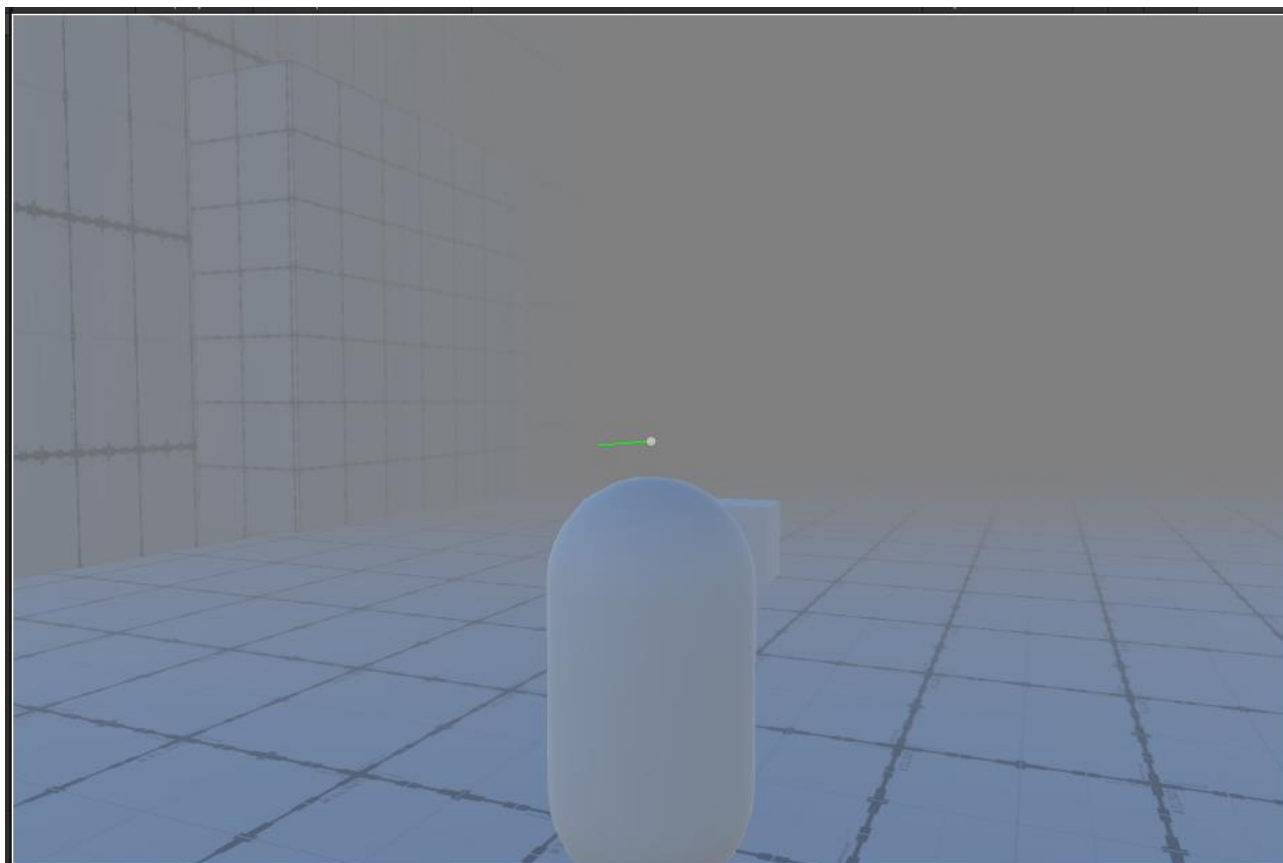


Рисунок 3.2 – Хаб локації(дім гравця)

Якщо гравець піде далі по до, він натрапить на двері, де можна буде перейти до ігрової локації (рис. 3.3).

		Е.Е. СОІН			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Р.В. ПЕТРОСЯН				
Змн.	Арк.	№ докум.	Підпис	Дата		73

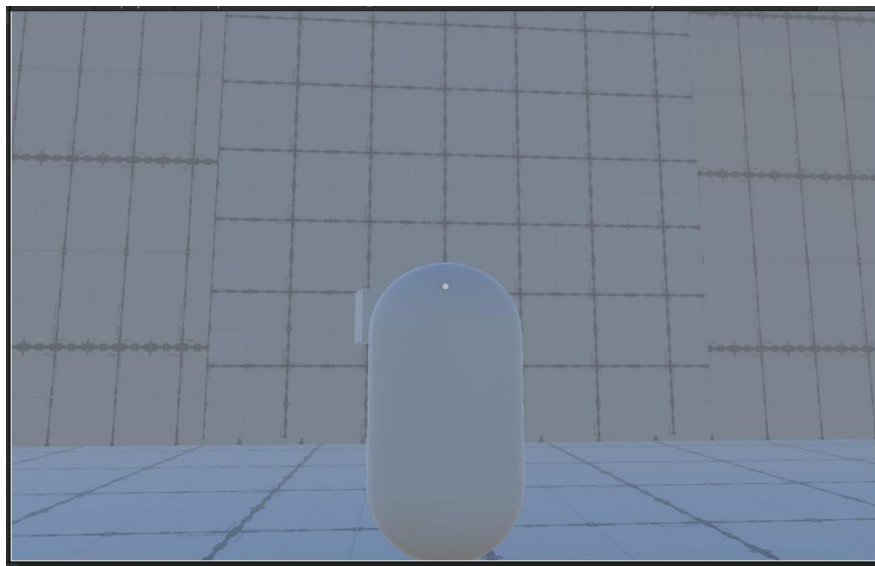


Рисунок 3.7 – Перехід з дому на вулицю

В бойовій сцені, гравець з'являється недалеко від дому (рис. 3.8).

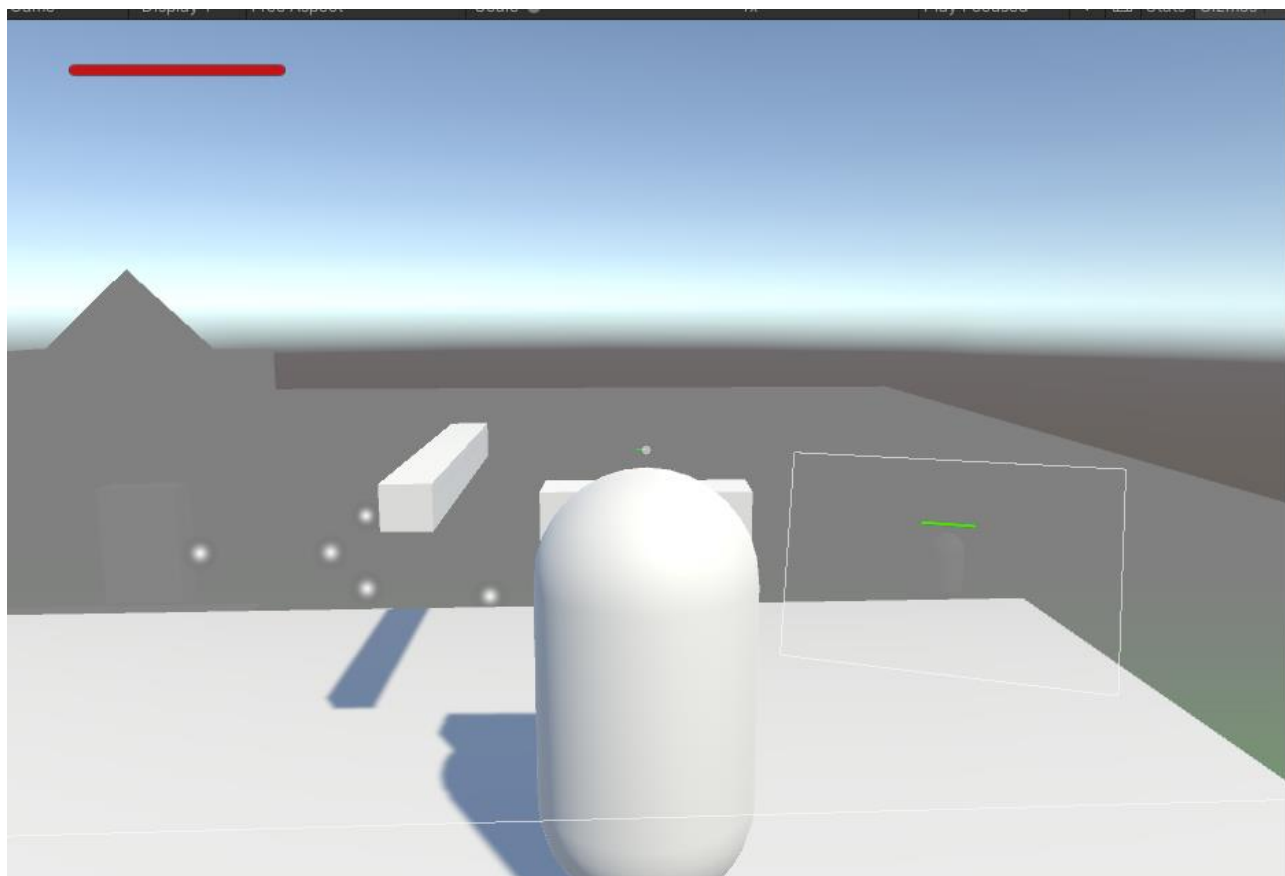


Рисунок 3.3 – Зброя

Зліва зверху на екрані гравець може побачити смугу здоров'я персонажа [17]. Цей елемент інтерфейсу є надзвичайно важливим, оскільки він показує поточний стан здоров'я персонажа. Смуга здоров'я зменшується, коли персонаж отримує ушкодження від ворогів або пасток, і збільшується при використанні

лікувальних предметів або інших способів відновлення здоров'я. Візуальне відображення здоров'я дозволяє гравцеві швидко орієнтуватися в ситуації та приймати необхідні рішення для виживання в грі. рис (3.3)



Рисунок 3.3 – Смуга здоров'я

Якщо гравець підійде до зброї подивиться на неї та натисне клавішу підбору то предмет з'явиться в інвентарі (рис. 3.4).

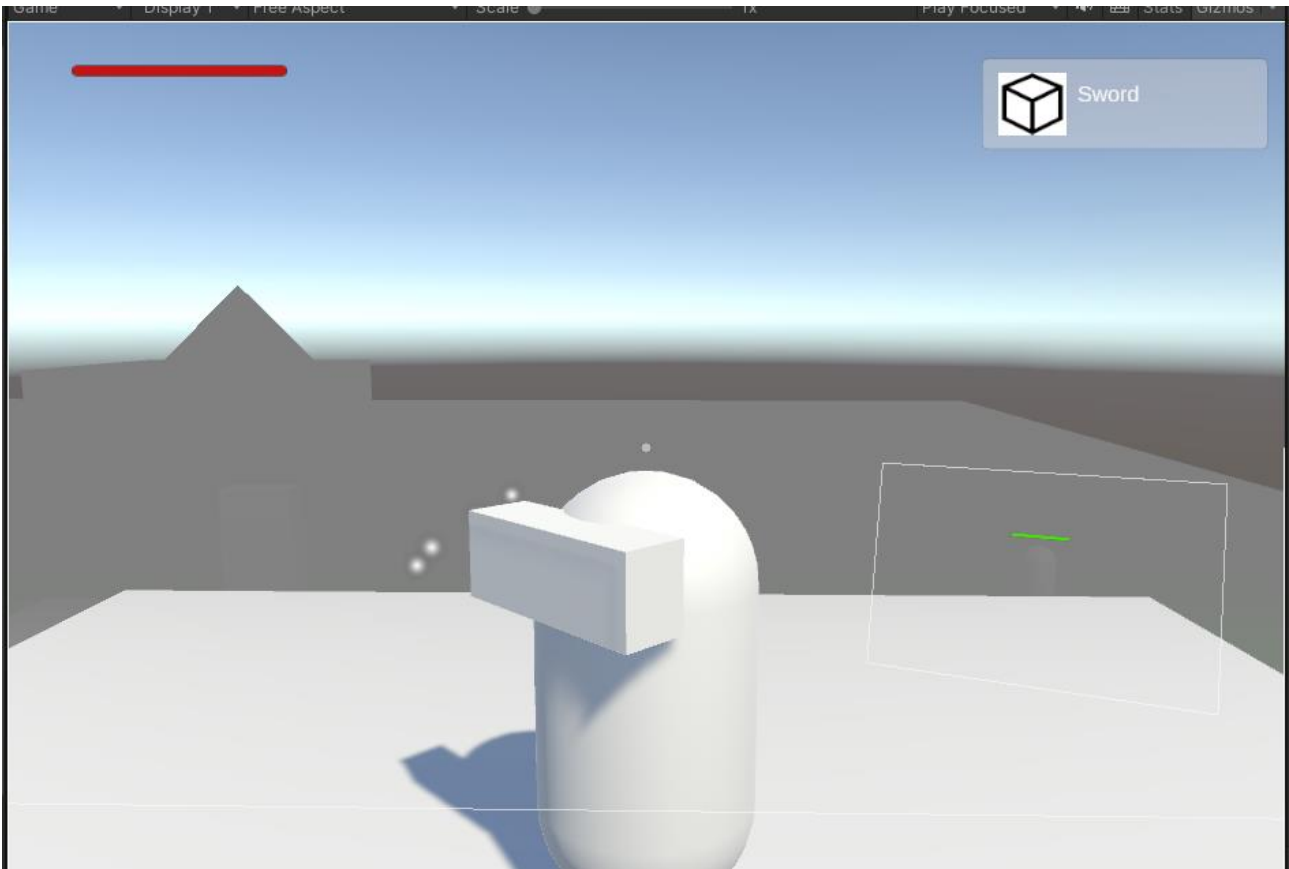


Рисунок 3.4 – Підбір зброї у інвентар

На (рис. 3.5) представлено скріншот з тим як піднімається зброя та що виводяться її назва та опис

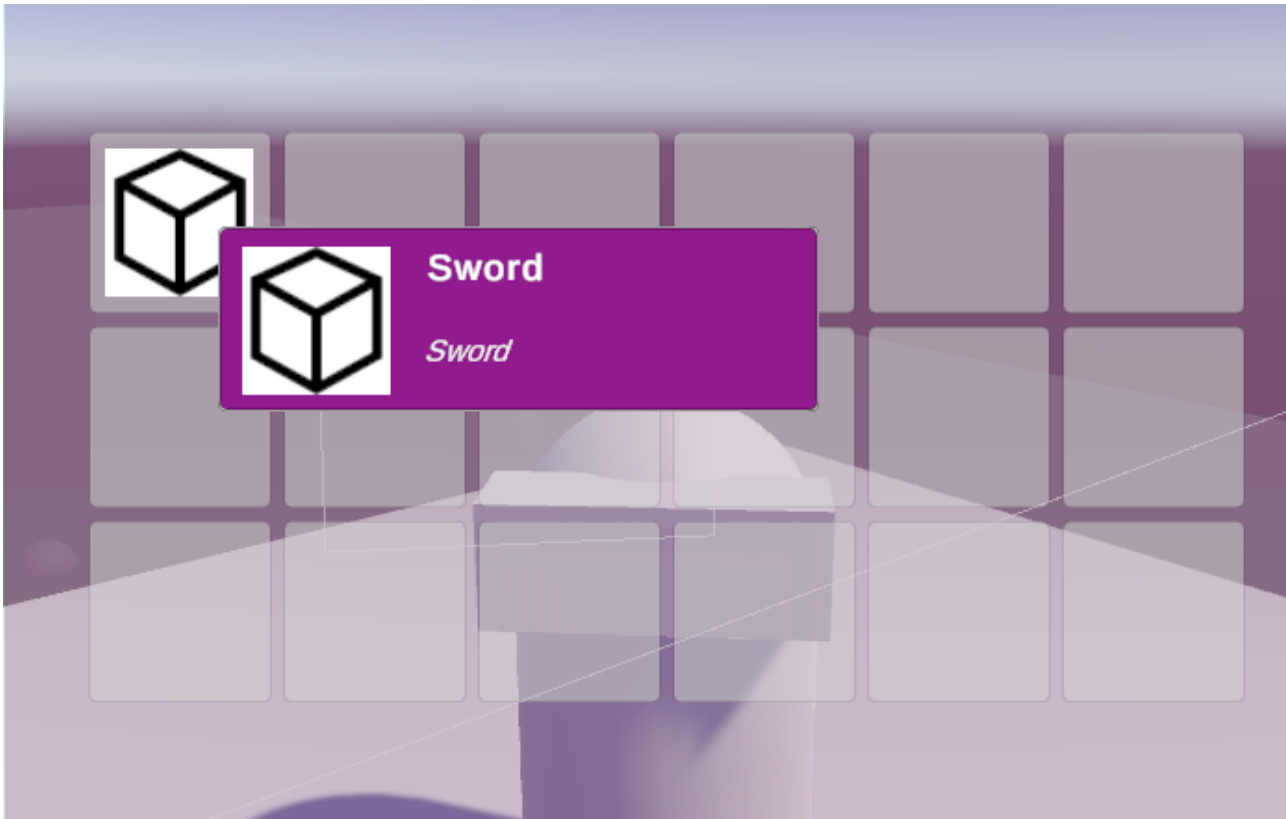


Рисунок 3.5 – Інвентар зі зброєю

Далі гравець може екіпірувати цей айтем і він з'явиться у руках головного героя (рис. 3.6).

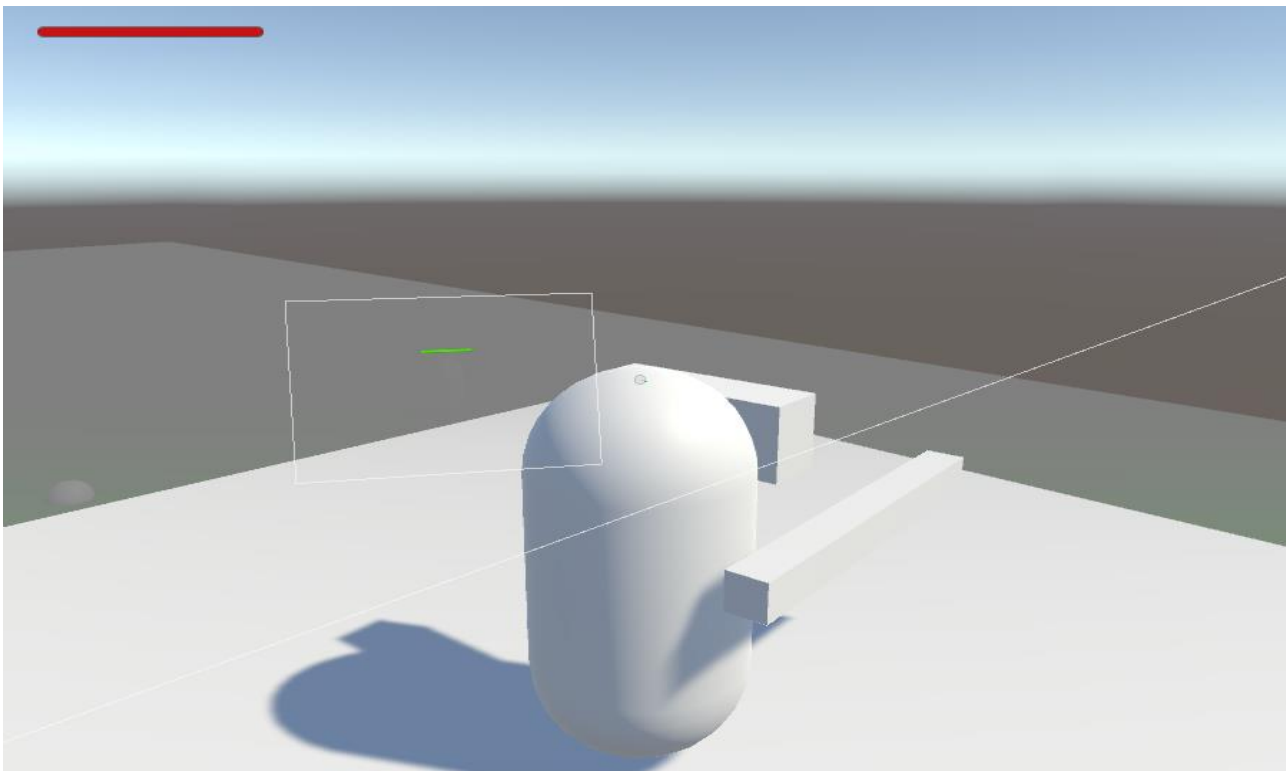


Рисунок 3.6 – Екіпірування зброї

У деяких точка на мапі будуть знаходитися вороги[18], у яких є можливість

		Е.Е. СОІН			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Р.В. ПЕТРОСЯН				76
Змн.	Арк.	№ докум.	Підпис	Дата		

патрулювання між точками та погонєю за гравцем[19](рис. 3.7).

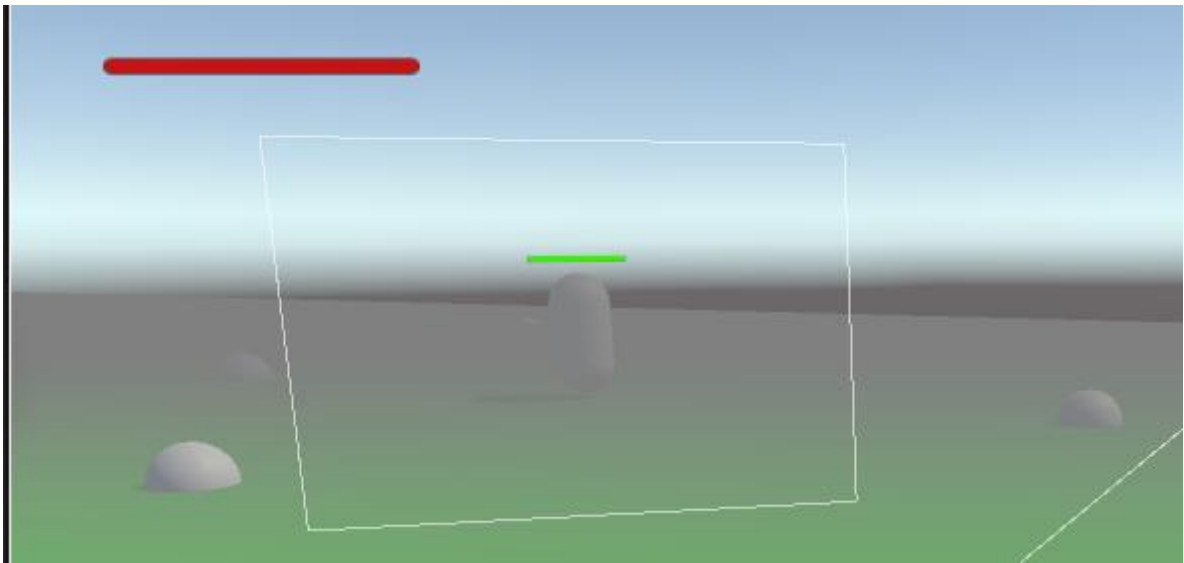


Рисунок 3.7 – Ворог та патрулювання

На (рис. 3.8) продемонстрована невеличка логічна задача в якій гравцю потрібно потрапити камінням на платформу і будуть відчинені двері

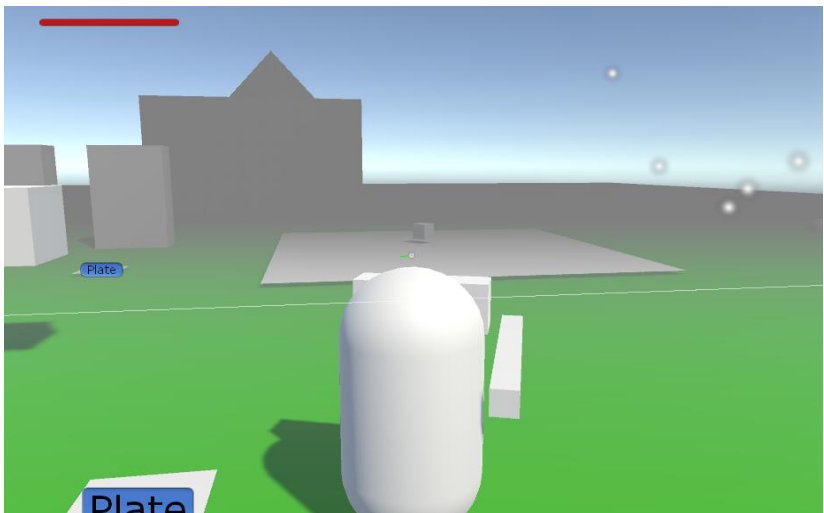
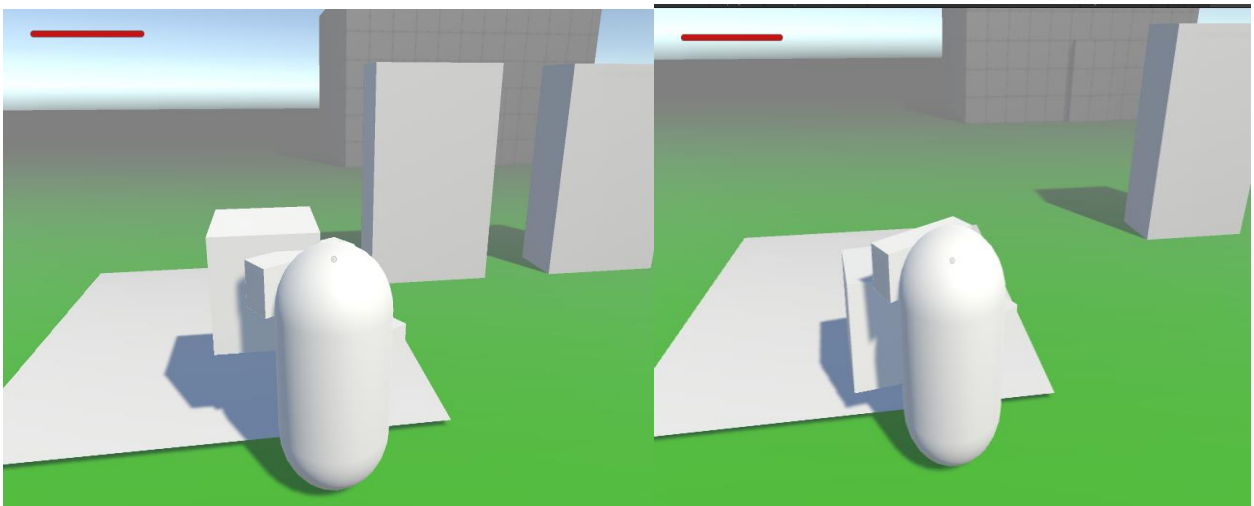


Рисунок 3.8 – Невеличка логічна задача



а)

б)

Рисунок 3.9 – Невеличка логічна задачка де гравцю потрібно з однієї точки перенести об'єкт до іншої щоб відчинити двері

(а) підняття об'єкту та закрита двері (б) положив об'єкт на платформу та відкрив двері

Якщо гравець дивиться на якийсь пікап та натискає кнопку підняття то він піднімає об'єкт в інветар та там вже може його використати у кожного об'єкту є свій опис та назва'являться справа зверху коли гравець його підіймає а опис з'являється коли гравець наводить мишу на цей айтем у інветарі (рис. 3.10).



Рисунок 3.10 – Підняття пікапів

На (рис. 3.11) продемонстрована пастка з стрілами, якщо гравець проходить та нажимає на плиту то спрацьовує механізм та випускає декілька стріл зі стіни

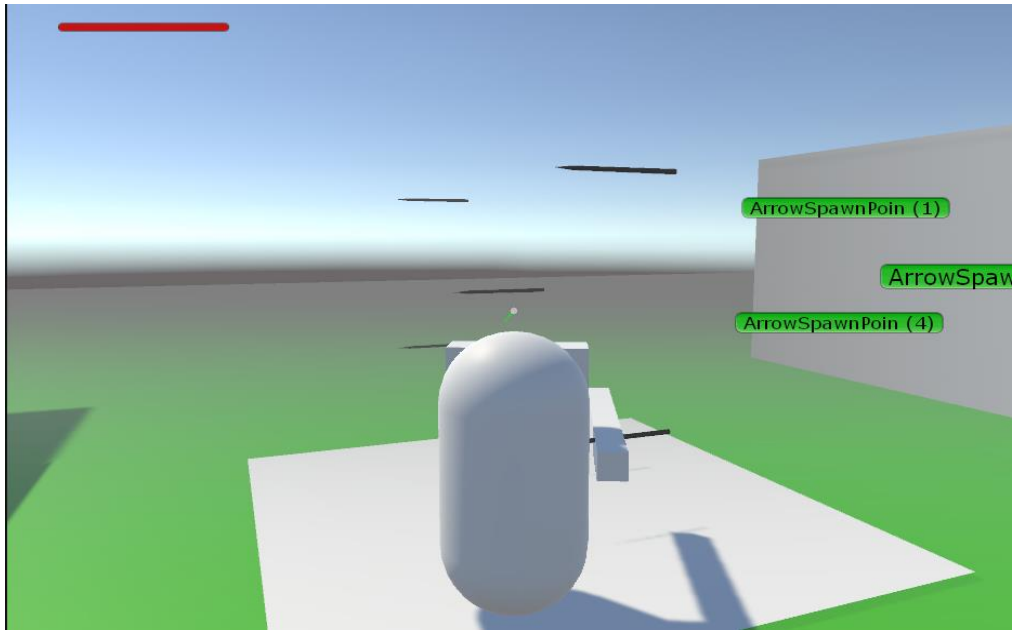


Рисунок 3.11 – Пастка зі стрілами

Також при отриманні урону був реалізований невеликий UI елемент який викликає почервоніння при отриманні урону(рис. 3.12)

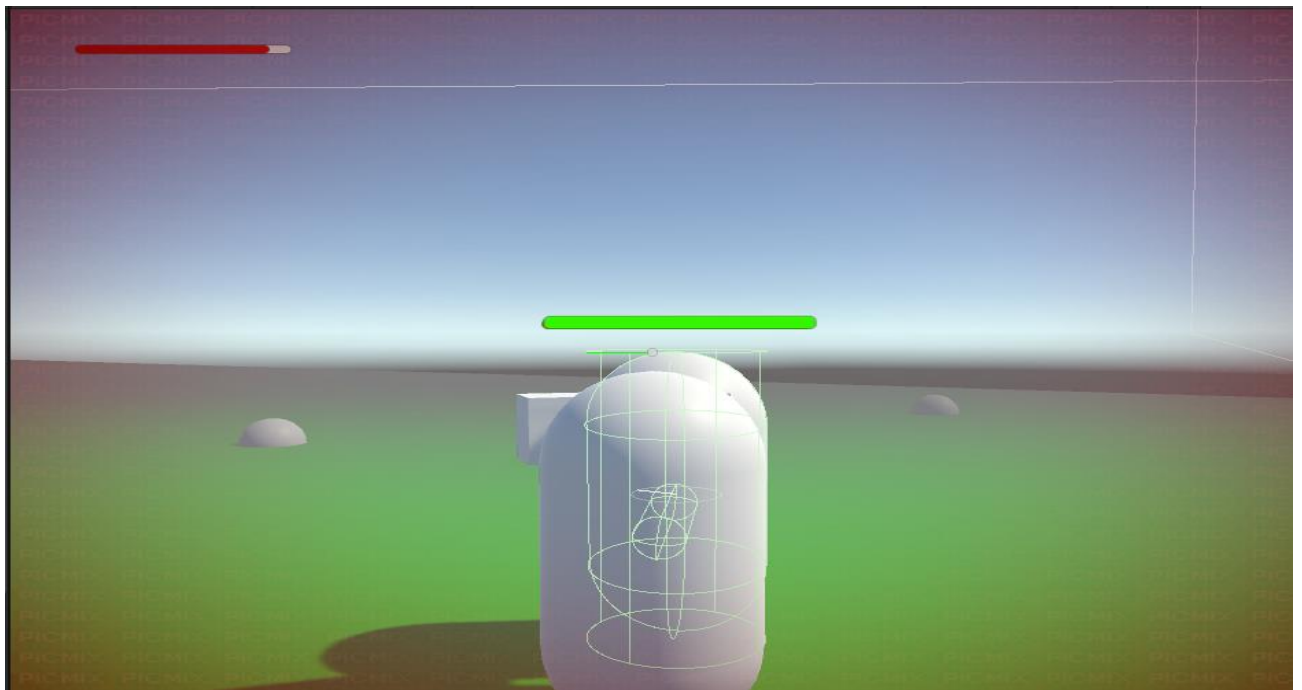


Рисунок 3.12 – Почервоніння екрану при отриманні урону

3.3 Тестування роботи програмного додатку

Проведено тестування усього програмного продукту [20], за для виявлення багів та помилок. Усі пункти тестування вказані в таблиці – таблиця 3.1

		Е.Е. СОІН			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Р.В. ПЕТРОСЯН				
Змн.	Арк.	№ докум.	Підпис	Дата		79

Таблиця тестування

Назва тестування	Вхідні дані	Очікувані дані	Отримані дані
Вхід в гру	Натискання два рази на гру	Відкриття гри	Реалізовано
Натискання кнопки "Play"	Клік по кнопці "Play"	Початок гри	Реалізовано
Натискання кнопки "Quit Game"	Клік по кнопці "Quit Game"	Вихід з гри	Реалізовано
Перенесення в хаб локацію	Виконання команди "Go to home"	Перехід у дім	Реалізовано
Пересування гравця	Виконання методу "MovePlayer"	Рух гравця	Реалізовано
Пересування мишкою	Рух мишки	Реакція гри на рух мишкою	Реалізовано
Перехід на ігрову локацію	Виконання методу "MoveIntoOut"	Перехід на вулицю	Реалізовано
Генерація рівня	Завантаження одного рівня	Відображення рівня	Реалізовано
Підбір зброї	Підбір предмету	Додавання зброї до інвентаря	Реалізовано

Продовження Таблиця 3.1

Екіпірування зброї	Виконання методу "SwitchSword"	Екіпірування у руку гравця	Реалізовано
Удар	Натискання клавіші "Fire1"	виконання удару	Реалізовано
Розміщення ворогів	розміщення ворогів на мапу	Поява ворогів на мапі	Реалізовано
Коректна робота ворогів	Взаємодія ворогів та гравця	Робота ШІ	Реалізовано
Смерть ворога	Вбивство ворогів	Стан смерті у ворога	Реалізовано
Віднімання здоров'я у гравця	Атака ворога	Зменшення здоров'я та відображення	Реалізовано
Перезапуск мапи при смерті	Смерть гравця	Перезапуск мапи	Реалізовано
Підіймання айтемів	Підбір предмету	Додавання айтему до інвентаря	Реалізовано
Використання айтемі	Виконання евенту	Додавання або віднімання здоров'я в залежності від предмету	Реалізовано
Логічна задача	Перетискання айтем з однієї точки в іншу	Відкриття дверей	Реалізовано
Пастка	Натискання гравцем на плиту	Виліт стріл зі стіни	Реалізовано

У процесі розробки гри "Nixton's Adventure" були виявлені кілька помилок. Наприклад, на початкових етапах гра не завжди коректно реагувала на натискання кнопки "Play", або вороги могли проходити крізь стіни. Також були проблеми з правильним відображенням рівня здоров'я персонажа та з його

		Е.Е. СОІН			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Р.В. ПЕТРОСЯН				
Змн.	Арк.	№ докум.	Підпис	Дата		81

пересуванням.

Всі ці помилки були детально проаналізовані і виправлені. Завдяки регулярному тестуванню і налагодженню, команда розробників змогла досягти стабільної і коректної роботи гри. Це дозволило забезпечити плавний і захоплюючий ігровий процес для користувачів.

Висновки до третього розділу

У цьому розділі було докладно розглянуто порядок встановлення та налаштування системи для гри "Nixton's Adventure". Також було описано інтерфейс гри та порядок роботи з програмним додатком, включаючи різні функції та дії, які може виконувати гравець, від початку до завершення гри. Були проаналізовані дії, які гравець має виконати для переходу на інші локації, а також вимоги до системи для забезпечення оптимальної роботи гри.

Було проведено тестування роботи програмного додатку, результати якого були представлені у вигляді таблиці. В процесі тестування було виявлено кілька помилок, таких як некоректна реакція на натискання кнопки "Play", проходження ворогів крізь стіни, проблеми з відображенням рівня здоров'я персонажа та його пересуванням. Усі ці помилки були детально проаналізовані та виправлені. Завдяки регулярному тестуванню та налагодженню, команда розробників досягла стабільної та коректної роботи гри, що дозволило забезпечити плавний та захоплюючий ігровий процес для користувачів.

		Е.Е. СОІН			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Р.В. ПЕТРОСЯН				
Змн.	Арк.	№ докум.	Підпис	Дата		82

ВИСНОВКИ

У даній кваліфікаційній роботі було успішно розроблено комп'ютерну гру "Nixton's Adventure" за допомогою ігрового рушія Unity3D. Процес розробки включав кілька ключових етапів: від аналізу проблематики та постановки задачі до проектування, розробки програмного забезпечення та його тестування.

Аналіз та постановка задачі: На початковому етапі було детально проаналізовано аналогічні ігри та визначено основні вимоги до проекту. Було встановлено, що для успішної реалізації гри необхідно забезпечити високу якість графіки, зручний інтерфейс та захоплюючий ігровий процес. Було обрано Unity3D як основний інструмент розробки завдяки його потужним можливостям у створенні тривимірної графіки та великій кількості доступних ресурсів.

Проектування та розробка: У процесі проектування було створено різноманітні діаграми, що відображають етапи та елементи розробки програми. Використані діаграми включають діаграми варіантів використання, активності, класів, компонентів і розгортання. Було детально розглянуто компоненти Unity, використані в проекті.

Реалізація: Основні етапи реалізації гри включали створення механік гравця, генерацію рівнів, реалізацію штучного інтелекту ворогів та розробку пасток. Усі етапи та ключові скрипти були детально описані та проілюстровані діаграмами, що допомогло чітко представити процес розробки та взаємозв'язки між різними елементами програмного продукту.

Тестування та виправлення помилок: Під час тестування були виявлені та виправлені кілька помилок, включаючи некоректну реакцію гри на натискання кнопки "Play", неправильне відображення рівня здоров'я персонажа та проблеми з пересуванням ворогів. Регулярне тестування та налагодження дозволили досягти стабільної та коректної роботи гри, забезпечивши плавний та захоплюючий ігровий процес для користувачів.

Висновки до розділів: У кожному розділі було детально описано результати роботи, включаючи мінімальні та рекомендовані вимоги до системи, функціонал гри, результати тестування та виправлення помилок. Ці висновки

		Е.Е. СОІН			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Р.В. ПЕТРОСЯН				
Змн.	Арк.	№ докум.	Підпис	Дата		83

демонструють досягнення поставлених цілей та успішну реалізацію проекту.

Отже, розробка гри "Nixton's Adventure" стала прикладом успішного виконання складного проекту з використанням сучасних інструментів та технологій, що підтверджує доцільність вибору Unity3D та інших засобів для реалізації комп'ютерних ігор.

		Е.Е. СОІН			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Р.В. ПЕТРОСЯН				
Змн.	Арк.	№ докум.	Підпис	Дата		84

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Youtube - THIRD PERSON MOVEMENT [Електронний ресурс]: <https://www.youtube.com/watch?v=UCwwn2q4Vys>
2. Absolver [Електронний ресурс] – Режим доступу до ресурсу: <https://store.steampowered.com/app/473690/Absolver/>
3. Life is Strange [Електронний ресурс]. – Режим доступу до ресурсу: [https://store.steampowered.com/app/319630/Life is Strange Episode 1/](https://store.steampowered.com/app/319630/Life_is_Strange_Episode_1/)
4. Tomb Raider I (1996) [Електронний ресурс]. – Режим доступу до ресурсу: [https://store.steampowered.com/app/224960/Tomb Raider I 1996/](https://store.steampowered.com/app/224960/Tomb_Raider_I_1996/)
5. Unity - Unity Documentation [Електронний ресурс]: <https://docs.unity.com/>
6. C# Guide - w3schools [Електронний ресурс]: <https://www.w3schools.com/cs/index.php>
7. C# Guide - Programming concepts (C#) [Електронний ресурс]: <https://learn.microsoft.com/uk-ua/dotnet/csharp/programming-guide/concepts/>
8. draw.io - Work with layers in draw.io [Електронний ресурс]: <https://www.drawio.com/doc/layers>
9. Logic of Enemy - Enemy AI Flow charts [Електронний ресурс]: <https://thevideogamediararies.wordpress.com/2017/05/12/enemy-ai-flow-charts/>
10. Unity - Scripting API: NavMesh [Електронний ресурс]: <https://docs.unity3d.com/ScriptReference/AI.NavMesh.html>
11. YouTube - New AI Navigation in Unity [Електронний ресурс]: <https://www.youtube.com/watch?v=u2EQtrdgfNs&t=44s>
12. Unity - Scripting API: Rigidbody [Електронний ресурс]: <https://docs.unity3d.com/ScriptReference/Rigidbody.html>
13. Unity – Character Controller [Електронний ресурс]: <https://docs.unity3d.com/ru/2018.4/Manual/class-CharacterController.html>

		Е.Е. СОІН			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Р.В. ПЕТРОСЯН				85
Змн.	Арк.	№ докум.	Підпис	Дата		

14. YouTube - C# Lists and Dictionaries in Unity! - Intermediate Scripting Tutorial [Электронный ресурс]:
https://www.youtube.com/watch?v=0WdWiF_Si4I
15. Unity - Enemy AI in Unity [Электронный ресурс]:
<https://gamedevbeginner.com/enemy-ai-in-unity/>
16. Unity - How to design a simple level using Probuilder in Unity [Электронный ресурс]: <https://blog.logrocket.com/design-level-probuilder-unity/>
17. Unity – Manual: User interface (UI) [Электронный ресурс]:
<https://docs.unity3d.com/Manual/UIToolkits.html>
18. Unity - Unity User Manual Framework for Unity3D [Электронный ресурс]: <https://docs.unity3d.com/Manual/index.html>
19. Unity - Animator - Scripting API [Электронный ресурс]:
<https://docs.unity3d.com/ScriptReference/Animator.html>
20. pinglestudio - What is the game testing process? [Электронный ресурс]: <https://pinglestudio.com/blog/game-testing/what-is-the-game-testing-process>

ДОДАТКИ

		Е.Е. СОІН			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Р.В. ПЕТРОСЯН				87
Змн.	Арк.	№ докум.	Підпис	Дата		

Файл PlayerMoving відповідає за переміщення врага:

```
using System;
```

```
using Unity.VisualScripting;
```

```
using UnityEditor;
```

```
using UnityEngine;
```

```
public class PlayerMoving : MonoBehaviour
```

```
{
```

```
    [Header("References")]
```

```
    [SerializeField] private Transform _orientation;
```

```
    [Header("Keybinds")]
```

```
    [SerializeField] public KeyCode _jumpKey = KeyCode.Space;
```

```
    [Header("Move")]
```

```
    [SerializeField] private float _Speed;
```

```
    private float _realSpeed;
```

```
    [SerializeField] private float _FastSpeed;
```

```
    [Header("Jump")]
```

```
    [SerializeField] private float _jumpForce;
```

```
    [SerializeField] private float _jumpCooldown;
```

		Е.Е. СОІН			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Р.В. ПЕТРОСЯН				
Змн.	Арк.	№ докум.	Підпис	Дата		88

```
[SerializeField] private float _airMultiplier;

[SerializeField] private bool _canJump;

[Header("Ground")]

[SerializeField] private float _distanceToGround;

[SerializeField] private float _groundGrag;

[SerializeField] private LayerMask _groundLayer;

[SerializeField] private bool _grounded;
```

```
private float _horizontal;

private float _vertical;

private Vector3 _moveDirection;

private Rigidbody _rigidbody;
```

```
private void Start()

{

    GetComponents();

    Settings();

}
```

```
private void FixedUpdate()

{
```

```

        MovePlayer();

    }

private void Update()

{

    CheckGround();


    SetInput();

    SpeedController();

}

private void SetInput()

{

    _horizontal = Input.GetAxisRaw("Horizontal");

    _vertical = Input.GetAxisRaw("Vertical");


    if (Input.GetKey(_jumpKey) && _canJump && _grounded)

    {

        _canJump = false;


        Jump();

```

		Е.Е. КОІН			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Р.В. ПЕТРОСЯН				90
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        Invoke(nameof(ResetJump), _jumpCooldown);

    }

}

private void MovePlayer()

{

    _moveDirection = _orientation.forward * _vertical + _orientation.right * _horizontal;

    if (_grounded)

    {

        if (Input.GetKey(KeyCode.LeftShift))

        {

            _rigidbody.AddForce(_moveDirection.normalized * _FastSpeed * 10f, ForceMode.Force);

        }

        else

        {

            _rigidbody.AddForce(_moveDirection.normalized * _Speed * 10f, ForceMode.Force);

        }

    }

    else

    {

        _rigidbody.AddForce(_moveDirection.normalized * _Speed * 10f * _airMultiplier, ForceMode.Force);

```

		Е.Е. КОИН			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Р.В. ПЕТРОСЯН				
Змн.	Арк.	№ докум.	Підпис	Дата		91

```

    }

}

private void SpeedController()

{

    Vector3 flatValue = new Vector3(_rigidbody.velocity.x, 0f, _rigidbody.velocity.z);

    if (flatValue.magnitude > _Speed)

    {

        Vector3 limitedValue = flatValue.normalized * _Speed;

        _rigidbody.velocity = new Vector3(limitedValue.x, _rigidbody.velocity.y, limitedValue.z);

    }

}

private void Jump()

{

    _rigidbody.velocity = new Vector3(_rigidbody.velocity.x, 0f, _rigidbody.velocity.z);

    _rigidbody.AddForce(transform.up * _jumpForce, ForceMode.Impulse);

}

private void ResetJump()

```

		Е.Е. КОИ			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Р.В. ПЕТРОСЯН				92
Змн.	Арк.	№ докум.	Підпис	Дата		

```

{

    _canJump = true;

}

private void CheckGround()

{

    _grounded = Physics.Raycast(

        transform.position, Vector3.down, _distanceToGround, _groundLayer);

    if (_grounded)

        _rigidbody.drag = _groundGrag;

    else

        _rigidbody.drag = 0f;

}

private void Settings()

{

    _canJump = true;

}

private void GetComponents()

{

```

		Е.Е. КОИ			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Р.В. ПЕТРОСЯН				93
Змн.	Арк.	№ докум.	Підпис	Дата		

```
_rigidbody = GetComponent<Rigidbody>();
```

```
_rigidbody.freezeRotation = true;
```

```
}
```

```
}
```

		Е.Е. СОИИ			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Р.В. ПЕТРОСЯН				94
Змн.	Арк.	№ докум.	Підпис	Дата		

Файл PickUp відповідає за підняття об'єктів:

```
using System.Collections;
```

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

```
using UnityEngine;
```

```
public class PickUp : MonoBehaviour
```

```
{
```

```
    [SerializeField] private Transform playerTransform;
```

```
    [SerializeField] private Transform objectGrabPoinTransform;
```

```
    private ObjectGrabble objectGrabble;
```

```
    [SerializeField] private LayerMask _layerMask;
```

```
    private float pickUpDistance = 100f;
```

```
    // Update is called once per frame
```

```
    void Update()
```

```
    {
```

```
        Vector3 rayOrigin = new Vector3(0.5f, 0.5f, 0f);
```

```
        Ray ray = Camera.main.ViewportPointToRay(rayOrigin);
```

```
        Debug.DrawRay(ray.origin, ray.direction * pickUpDistance, Color.green);
```

```
        // Debug.DrawRay(playerCameraTransform.transform.position,playerCameraTransform.transform.position , Color.red);
```

```
        if (Input.GetKeyDown(KeyCode.E))
```

		Е.Е. СОІН			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Р.В. ПЕТРОСЯН				95
Змн.	Арк.	№ докум.	Підпис	Дата		

```

{

    RaycastHit hit;

    if (objectGrabble == null)

    {

        if (Physics.Raycast(ray, out hit, pickUpDistance, _layerMask))

        {

            if (hit.transform.TryGetComponent(out objectGrabble))

            {

                objectGrabble.Grab(objectGrabPoinTransform);

                Debug.Log(objectGrabble);

            }

        }

    }

    else

    {

        objectGrabble.Drop();

        objectGrabble = null;

    }

}

}

```

		Е.Е. СОИН			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Р.В. ПЕТРОСЯН				
Змн.	Арк.	№ докум.	Підпис	Дата		96

Файл ObjectGrabble займається управлінням процесом підняття і відпускання об'єктів:

```
using System;
```

```
using System.Collections;
```

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

```
using UnityEngine;
```

```
public class ObjectGrabble : MonoBehaviour
```

```
{
```

```
    private Rigidbody objectRigidbody;
```

```
    private Collider objectCollider;
```

```
    private Transform objectGrabPointTransform;
```

```
    private float lerpSpeed = 10f;
```

```
    private LayerMask _layerMask;
```

```
    private void Awake()
```

```
{
```

```
        objectRigidbody = GetComponent<Rigidbody>();
```

```
        objectCollider = GetComponent<Collider>();
```

```
}
```

```
    public void Grab(Transform objectGrabPoinTransform)
```

```
{
```

		Е.Е. СОІН			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Р.В. ПЕТРОСЯН				
Змн.	Арк.	№ докум.	Підпис	Дата		97

```

this.objectGrabPointTransform = objectGrabPointTransform;

objectRigidbody.useGravity = false;

objectCollider.enabled = false;

}

public void Drop()

{

this.objectGrabPointTransform = null;

objectRigidbody.useGravity = true;

objectCollider.enabled = true;

}

private void FixedUpdate()

{

if (objectGrabPointTransform != null)

{

Vector3 newPosition = Vector3.Lerp(transform.position, objectGrabPointTransform.position, Time.deltaTime *
lerpSpeed);

objectRigidbody.MovePosition(newPosition);

}

}

}

```

		Е.Е. СОИ			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Р.В. ПЕТРОСЯН				98
Змн.	Арк.	№ докум.	Підпис	Дата		

Файл HealthBar відповідає за ХП героя:

```
using System;
```

```
using System.Collections;
```

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

```
using System.Security;
```

```
using UnityEngine;
```

```
using UnityEngine;
```

```
using UnityEngine.SceneManagement;
```

```
using UnityEngine.UI;
```

```
public class HealthBar : MonoBehaviour
```

```
{
```

```
    public GameObject RedOverlay;
```

```
    public void AddHealth(int value)
```

```
{
```

```
        GetComponent<Slider>().value += value;
```

```
}
```

```
    private void Update()
```

```
{
```

```
        Death();
```

		Е.Е. СОІН			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Р.В. ПЕТРОСЯН				99
Змн.	Арк.	№ докум.	Підпис	Дата		

```
}
```

```
void Death()
```

```
{
```

```
    if (GetComponent<Slider>().value <= 0)
```

```
    {
```

```
        SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex);
```

```
        Time.timeScale = 1;
```

```
    }
```

```
}
```

```
public IEnumerator Damage(int damageCount)
```

```
{
```

```
    GetComponent<Slider>().value -= damageCount;
```

```
    RedOverlay.SetActive(true);
```

```
    yield return new WaitForSeconds(.5f);
```

```
    RedOverlay.SetActive(false);
```

```
}
```

```
}
```

		Е.Е. КОИН			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Р.В. ПЕТРОСЯН				100
Змн.	Арк.	№ докум.	Підпис	Дата		

Файл AttackScript відповідає за атаку:

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

public class AttackScript : MonoBehaviour

{

    private Animator animator;
    // Start is called before the first frame update
    void Start()
    {
        animator = GetComponent<Animator>();
    }
    // Update is called once per frame
    void Update()

    {

        if (Input.GetButtonDown("Fire1"))

        {

            animator.SetBool("Attack", true);

        }

        else if (Input.GetButtonUp("Fire1"))

        {

            animator.SetBool("Attack", false);

        }

    }

}
```

		Е.Е. СОІН			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Р.В. ПЕТРОСЯН				101
Змн.	Арк.	№ докум.	Підпис	Дата		

Файл Throw відповідає за метання каміння:

```
using System;

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.Serialization;


public class Throw : MonoBehaviour

{

    public Transform RockSpawnPoint;


    public GameObject RockPrefab;


    public Transform cameraFold;


    private Rigidbody rbRock;


    private GameObject rock;

    private bool CanThrow;

    private Vector3 viewDirection;


    public float throwForce = 5f;
```

```
private void Start()

{

    CanThrow = true;

    rock = Instantiate(RockPrefab, RockSpawnPoint.position, RockSpawnPoint.rotation);

    rbRock = rock.GetComponent<Rigidbody>();

    rock.SetActive(false);

}


// Update is called once per frame

void Update()

{

    viewDirection = transform.position - new Vector3(

        cameraFold.position.x, transform.position.y, cameraFold.position.z);

    if (Input.GetKeyDown(KeyCode.F) && CanThrow)

    {

        CanThrow = false;

        throwRock();

        Invoke(nameof(ResetThrow), 3f);

    }

}
```

```

private void throwRock()

{

    rock.SetActive(true);

    rock.transform.position = RockSpawnPoint.position;

    rbRock.velocity = new Vector3(0,0,0);

    rbRock.AddForce(viewDirection.normalized * throwForce, ForceMode.Impulse);

}


private void ResetThrow()

{

    CanThrow = true;

}

}

```

		Е.Е. СОІН			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Р.В. ПЕТРОСЯН				104
Змн.	Арк.	№ докум.	Підпис	Дата		

Файл ThirdPersonCamera відповідає за камеру у грі

```
using System;
```

```
using System.Collections;
```

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

```
using System.Security.Cryptography.X509Certificates;
```

```
using UnityEngine;
```

```
public class ThirdPersonCamera : MonoBehaviour
```

```
{
```

```
    [Header("References")]
```

```
    [SerializeField] private Transform _player;
```

```
    [SerializeField] private Transform _playerBody;
```

```
    [SerializeField] private Transform _orientation;
```

```
    private Rigidbody _rbRigidbody;
```

```
    [Header("Settings")]
```

```
    [SerializeField] private float _rotationSpeed;
```

```
    private Transform _currentTransform;
```

```
    private float _turnSmoothVelocity;
```

		Е.Е. КОІН			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Р.В. ПЕТРОСЯН				105
Змн.	Арк.	№ докум.	Підпис	Дата		

private void Start()

{

GetComponents();

CursorSettings();

}

private void Update()

{

RotationProcess();

}

private void RotationProcess()

{

CalculateOrientation();

RotatePlayerBody();

}

private void CalculateOrientation()

{

Vector3 viewDirection = _player.position - new Vector3(

_currentTransform.position.x, _player.position.y, _currentTransform.position.z);

```

        _orientation.forward = viewDirection.normalized;

    }

    private void RotatePlayerBody()

    {

        float horizontal = Input.GetAxisRaw("Horizontal");

        float vertical = Input.GetAxisRaw("Vertical");

        Vector3 inputDirection = _orientation.forward * vertical + _orientation.right * horizontal;

        if (inputDirection != Vector3.zero)

        {

            _playerBody.forward = Vector3.Slerp(_playerBody.forward, inputDirection.normalized,

                Time.deltaTime * _rotationSpeed);

        }

    }

    private void CursorSettings()

    {

        Cursor.lockState = CursorLockMode.Locked;

        Cursor.visible = false;

    }

```

```
private void GetComponents()

{

    _currentTransform = GetComponent<Transform>();

    _rbRigidbody = _player.GetComponent<Rigidbody>();

}

}
```

		Е.Е. СОИН			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Р.В. ПЕТРОСЯН				108
Змн.	Арк.	№ докум.	Підпис	Дата		

Файл AttackBeheviour відповідає за стан атаки у ворого

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

public class AttackBeheviour : StateMachineBehaviour

{
    private Transform player;

    override public void OnStateEnter(Animator animator, AnimatorStateInfo stateInfo, int layerIndex)

    {
        player = GameObject.FindObjectOfType<PlayerMoving>().transform;
    }

    override public void OnStateUpdate(Animator animator, AnimatorStateInfo stateInfo, int layerIndex)
    {
        animator.transform.LookAt(player);

        float distance = Vector3.Distance(animator.transform.position, player.position);

        if (distance > 3)

        {

            animator.SetBool("IsAttack", false);
        }
    }

    override public void OnStateExit(Animator animator, AnimatorStateInfo stateInfo, int layerIndex)

    {

    }

}
```

		Е.Е. СОІН			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Р.В. ПЕТРОСЯН				109
Змн.	Арк.	№ докум.	Підпис	Дата		

Файл ChaseBehaviour відповідає за стан переслідування у ворога за гравцем

```
using System.Collections;
```

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

```
using UnityEngine;
```

```
using UnityEngine.AI;
```

```
public class ChaseBehaviour : StateMachineBehaviour
```

```
{
```

```
    private NavMeshAgent agent;
```

```
    private Transform player;
```

```
    float attack_range = 2;
```

```
    private float Chase_Range = 10;
```

```
    override public void OnStateEnter(Animator animator, AnimatorStateInfo stateInfo, int layerIndex)
```

```
{
```

```
        agent = animator.GetComponent<NavMeshAgent>();
```

```
        agent.speed = 4;
```

```
        player = GameObject.FindObjectOfType<PlayerMoving>().transform;
```

```
}
```

```
    override public void OnStateUpdate(Animator animator, AnimatorStateInfo stateInfo, int layerIndex)
```

```
{
```

		Е.Е. КОІН			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Р.В. ПЕТРОСЯН				110
Змн.	Арк.	№ докум.	Підпис	Дата		

```
agent.SetDestination(player.position);
```

```
float distance = Vector3.Distance(animator.transform.position, player.position);
```

```
if (distance < attack_range)
```

```
{
```

```
    animator.SetBool("IsAttack", true);
```

```
}
```

```
if (distance > 10)
```

```
{
```

```
    animator.SetBool("IsChasing", false);
```

```
}
```

```
}
```

```
override public void OnStateExit(Animator animator, AnimatorStateInfo stateInfo, int layerIndex)
```

```
{
```

```
    agent.SetDestination(agent.transform.position);
```

```
    agent.speed = 2;
```

```
}
```

```
}
```

		Е.Е. КОИ			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Р.В. ПЕТРОСЯН				111
Змн.	Арк.	№ докум.	Підпис	Дата		

Файл EnemyScript це скрипт відповідає за ворого його хп, анімацію

```
using System.Collections;
```

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

```
using UnityEngine;
```

```
using UnityEngine.UI;
```

```
public class EnemyScript : MonoBehaviour
```

```
{
```

```
    private int HP = 100;
```

```
    public Animator animator;
```

```
    public Slider healthBar;
```

```
    public GameObject Sword;
```

```
    // Update is called once per frame
```

```
    void Update()
```

```
    {
```

```
        healthBar.value = HP;
```

```
    }
```

		Е.Е. СОІН			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Р.В. ПЕТРОСЯН				112
Змн.	Арк.	№ докум.	Підпис	Дата		

```

public void TakeDamage(int damageAmount)

{

    HP -= damageAmount;

    if (HP <= 0)

    {

        animator.SetTrigger("death");

        GetComponent<Collider>().enabled = false;

        healthBar.gameObject.SetActive(false);

        Sword.GetComponent<Collider>().enabled = false;

    }

    else

    {

        animator.SetTrigger("damage");

    }

}

}

```

		Е.Е. КОИ			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Р.В. ПЕТРОСЯН				113
Змн.	Арк.	№ докум.	Підпис	Дата		

Файл IdleBeheviour відповідає за стан спокію у ворога:

```
using System.Collections;
```

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

```
using UnityEngine;
```

```
public class IdleBeheviour : StateMachineBehaviour
```

```
{
```

```
    private float timer;
```

```
    private Transform player;
```

```
    private float chaseRange = 10;
```

```
    override public void OnStateEnter(Animator animator, AnimatorStateInfo stateInfo, int layerIndex)
```

```
    {
```

```
        timer = 0;
```

```
        player = GameObject.FindGameObjectWithTag("Player").transform;
```

```
    }
```

```
    override public void OnStateUpdate(Animator animator, AnimatorStateInfo stateInfo, int layerIndex)
```

```
    {
```

```
        timer += Time.deltaTime;
```

```
        if (timer > 5)
```

```
        {
```

		Е.Е. СОІН			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Р.В. ПЕТРОСЯН				114
Змн.	Арк.	№ докум.	Підпис	Дата		

```
    animator.SetBool("IsPatrolling", true);  
  
}
```

```
float distance = Vector3.Distance(animator.transform.position, player.position);  
  
if (distance < chaseRange)  
{  
  
    animator.SetBool("IsChasing", true);  
  
}  
}
```

```
override public void OnStateExit(Animator animator, AnimatorStateInfo stateInfo, int layerIndex)  
{  
  
  
  
}  
  
}
```

Файл PatrolBeheviour відповідає за стан патрулювання:

```
using System.Collections;
```

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

```
using UnityEngine;
```

```
using UnityEngine.AI;
```

```
public class PatrolBeheviour : StateMachineBehaviour
```

```
{
```

```
    private float timer;
```

```
    private List<Transform> points = new List<Transform>();
```

```
    private NavMeshAgent agent;
```

```
    private Transform player;
```

```
    private float chaseRange = 10;
```

```
    override public void OnStateEnter(Animator animator, AnimatorStateInfo stateInfo, int layerIndex)
```

```
{
```

```
    timer = 0;
```

```
    Transform pointsObj = GameObject.FindGameObjectWithTag("Points").transform;
```

		Е.Е. СОІН			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Р.В. ПЕТРОСЯН				116
Змн.	Арк.	№ докум.	Підпис	Дата		

```
foreach (Transform t in pointsObj)
```

```
{
```

```
    points.Add(t);
```

```
}
```

```
agent = animator.GetComponent<NavMeshAgent>();
```

```
agent.SetDestination(points[0].position);
```

```
player = GameObject.FindGameObjectWithTag("Player").transform;
```

```
}
```

```
override public void OnStateUpdate(Animator animator, AnimatorStateInfo stateInfo, int  
layerIndex)
```

```
{
```

```
    if (agent.remainingDistance <= agent.stoppingDistance)
```

```
    {
```

```
        agent.SetDestination(points[Random.Range(0, points.Count)].position);
```

```
    }
```

```
    timer += Time.deltaTime;
```

		Е.Е. СОИ			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Р.В. ПЕТРОСЯН				117
Змн.	Арк.	№ докум.	Підпис	Дата		

```

if (timer > 10)

{

    animator.SetBool("IsPatrolling", false);

}

float distance = Vector3.Distance(animator.transform.position, player.position);

if (distance < chaseRange)

{

    animator.SetBool("IsChasing", true);

}

}

override public void OnStateExit(Animator animator, AnimatorStateInfo stateInfo, int layerIndex)
{

    agent.SetDestination(agent.transform.position);

}

}

```

		Е.Е. СОІН			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Р.В. ПЕТРОСЯН				118
Змн.	Арк.	№ докум.	Підпис	Дата		

Файл CurrentItem відповідає за конкретний айтем

```
using System.Collections;
```

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

```
using UnityEngine;
```

```
using UnityEngine;
```

```
using UnityEngine.EventSystems;
```

```
public class CurrentItem : MonoBehaviour, IPointerClickHandler, IDropHandler
```

```
{
```

```
[HideInInspector]
```

```
public int index;
```

```
private GameObject inventoryObj;
```

```
private Inventory inventory;
```

```
// Start is called before the first frame update
```

```
void Start()
```

```
{
```

```
inventoryObj = GameObject.FindGameObjectWithTag("InventoryManager");
```

```
inventory = inventoryObj.GetComponent<Inventory>();
```

```
}
```

		Е.Е. СОІН			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Р.В. ПЕТРОСЯН				119
Змн.	Арк.	№ докум.	Підпис	Дата		

```
public void OnPointerClick(PointerEventData eventData)

{

    if (eventData.button == PointerEventData.InputButton.Left)

    {

        if (inventory.item[index].customEvent != null)

        {

            inventory.item[index].customEvent.Invoke();

        }

        if (inventory.item[index].isRemoveble)

        {

            Remove();

        }

    }

    if (eventData.button == PointerEventData.InputButton.Right)

    {

        if (inventory.item[index].isDropped)

        {

            Drop();

            Remove();

        }

    }

}
```

```

    }

}

public void Remove()

{

    if (inventory.item[index].id != 0)

    {

        if (inventory.item[index].countItem > 1)

        {

            inventory.item[index].countItem--;

        }

        else

        {

            inventory.item[index] = new Item();

        }

        inventory.DisplayItems();

    }

}

public void Drop()

{

```

		Е.Е. КОИ			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Р.В. ПЕТРОСЯН				121
Змн.	Арк.	№ докум.	Підпис	Дата		


```

        return;

    }

    CurrentItem currentdraggeItem = draggedObj.GetComponent<CurrentItem>();

    if (currentdraggeItem != null)

    {

        Item currentItem = inventory.item[GetComponent<CurrentItem>().index];

        inventory.item[GetComponent<CurrentItem>().index] = inventory.item[currentdraggeItem.index];

        inventory.item[currentdraggeItem.index] = currentItem;

        inventory.DisplayItems();

    }

}

}

```

		Е.Е. СОІН			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Р.В. ПЕТРОСЯН				123
Змн.	Арк.	№ докум.	Підпис	Дата		

Файл Drag відповідає перетягування:

```
using System.Collections;
```

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

```
using TMPro;
```

```
using Unity.VisualScripting;
```

```
using UnityEngine;
```

```
using UnityEngine.EventSystems;
```

```
using UnityEngine.UI;
```

```
public class Drag : MonoBehaviour, IBeginDragHandler, IDragHandler, IEndDragHandler
```

```
{
```

```
    public static GameObject draggedObj;
```

```
    private Inventory inventory;
```

```
    // Start is called before the first frame update
```

```
    void Start()
```

```
{
```

```
        inventory = GameObject.FindGameObjectWithTag("InventoryManager").GetComponent<Inventory>();
```

```
}
```

```
    // Update is called once per frame
```

```
    void Update()
```

		Е.Е. СОІН			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Р.В. ПЕТРОСЯН				124
Змн.	Арк.	№ докум.	Підпис	Дата		

```

{

}

public void OnBeginDrag(PointerEventData eventData)

{

    draggedObj = gameObject;

    inventory.DragPrefab.SetActive(true);

    inventory.DragPrefab.GetComponent<CanvasGroup>().blocksRaycasts = false;

    if (draggedObj.GetComponent<CurrentItem>() != null)

    {

        int index = draggedObj.GetComponent<CurrentItem>().index;

        inventory.DragPrefab.GetComponent<Image>().sprite = inventory.item[index].Icon;

        if (inventory.item[index].countItem > 1)

        {

            inventory.DragPrefab.transform.GetChild(0).GetComponent<TMP_Text>().text =
            inventory.item[index].countItem.ToString();

        }

        else

        {

            inventory.DragPrefab.transform.GetChild(0).GetComponent<TMP_Text>().text = null;

```

		Е.Е. СОИ			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Р.В. ПЕТРОСЯН				125
Змн.	Арк.	№ докум.	Підпис	Дата		

```

    }

    if (inventory.DragPrefab.GetComponent<Image>().sprite == null)

    {

        draggedObj = null;

        inventory.DragPrefab.SetActive(false);

    }

}

}

```

```

public void OnDrag(PointerEventData eventData)

{

    inventory.DragPrefab.transform.position = Input.mousePosition;

}

```

```

public void OnEndDrag(PointerEventData eventData)

{

    if (!EventSystem.current.IsPointerOverGameObject())

    {

        draggedObj.GetComponent<CurrentItem>().Drop();

        draggedObj.GetComponent<CurrentItem>().Remove();

    }

}

```

		Е.Е. КОИН			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Р.В. ПЕТРОСЯН				126
Змн.	Арк.	№ докум.	Підпис	Дата		

```
draggedObj = null;
```

```
inventory.DragPrefab.SetActive(false);
```

```
inventory.DragPrefab.GetComponent<CanvasGroup>().blocksRaycasts = true;
```

```
}
```

```
}
```

		Е.Е. СОИН			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Р.В. ПЕТРОСЯН				127
Змн.	Арк.	№ докум.	Підпис	Дата		

Файл Inventory відповідає за сам інвентар :

```
using System.Collections;
```

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

```
using TMPro;
```

```
using UnityEngine;
```

```
using UnityEngine.UI;
```

```
public class Inventory : MonoBehaviour
```

```
{
```

```
    [HideInInspector]
```

```
    public List<Item> item;
```

```
    public GameObject DataBase;
```

```
    public GameObject cellContainer;
```

```
    public KeyCode ShowInvenotry;
```

```
    public KeyCode TakeButton;
```

		Е.Е. СОІН			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Р.В. ПЕТРОСЯН				128
Змн.	Арк.	№ докум.	Підпис	Дата		

```
public GameObject Point;

[Header("Message")]

public GameObject MessageManager;

public GameObject Message;

[Header("ToolType")]

public float offsetX;

public float offsetY;

public GameObject toolTypeObj;

public Image icon;

public TMP_Text itemName;

public TMP_Text itemDescriptprion;

[Header("Drag")]

public GameObject DragPrefab;

// Start is called before the first frame update

void Start()

{

    toolTypeObj.SetActive(false);
```

```
item = new List<Item>();
```

```
cellContainer.SetActive(false);
```

```
for (int i = 0; i < cellContainer.transform.childCount; i++)
```

```
{
```

```
    cellContainer.transform.GetChild(i).GetComponent<CurrentItem>().index = i;
```

```
}
```

```
for (int i = 0; i < cellContainer.transform.childCount; i++)
```

```
{
```

```
    item.Add(new Item());
```

```
}
```

```
}
```

```
// Update is called once per frame
```

```
void Update()
```

```
{
```

```
    ToggleInventory();
```

```
    if (Input.GetKeyDown(TakeButton))
```

```
{
```

```
    Ray ray = Camera.main.ViewportPointToRay(new Vector3(0.5f, 0.5f, 0.5f));
```

		Е.Е. КОИ			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Р.В. ПЕТРОСЯН				130
Змн.	Арк.	№ докум.	Підпис	Дата		

```

RaycastHit hit;

if (Physics.Raycast(ray, out hit, 10f))

{

    if (hit.collider.GetComponent<Item>())

    {

        AddItem(hit.collider.GetComponent<Item>());

        Item currentItem = hit.collider.GetComponent<Item>();

        Message(currentItem);

    }

}

}

}

void Message(Item currentItem)

{

    GameObject msgObj = Instantiate(Message);

    msgObj.transform.SetParent(MessageManager.transform);

    Image msgImage = msgObj.transform.GetChild(0).GetComponent<Image>();

    msgImage.sprite = currentItem.Icon;

    TMP_Text msgText = msgObj.transform.GetChild(1).GetComponent<TMP_Text>();

```

		Е.Е. СОИ			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Р.В. ПЕТРОСЯН				131
Змн.	Арк.	№ докум.	Підпис	Дата		

```

msgText.text = currentItem.nameItem;

}

void AddItem(Item currentItem)

{

    if (currentItem.isStackable == true)

    {

        AddStackableItem(currentItem);

    }

    else

    {

        AddUnstakableItem(currentItem);

    }

}

void AddUnstakableItem(Item currentItem)

{

    for (int i = 0; i < item.Count; i++)

    {

        if (item[i].id == 0)

        {

```

		Е.Е. СОИН			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Р.В. ПЕТРОСЯН				132
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        item[i] = currentItem;

        item[i].countItem = 1;

        DisplayItems();

        Destroy(currentItem.gameObject);

        break;

    }

}

}

```

```

void AddStackableItem(Item currentItem)

```

```

{

    for (int i = 0; i < item.Count; i++)

    {

        if (item[i].id == currentItem.id)

        {

            item[i].countItem++;

            DisplayItems();

            Destroy(currentItem.gameObject);

            return;

        }

    }

}

```

```

AddUnstakableItem(currentItem);

```

		Е.Е. КОИ			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Р.В. ПЕТРОСЯН				133
Змн.	Арк.	№ докум.	Підпис	Дата		

}

void ToggleInventory()

{

if (Input.GetKeyDown>ShowInvenotry))

{

if (cellContainer.activeSelf)

{

cellContainer.SetActive(false);

Time.timeScale = 1;

Cursor.visible = false;

Point.SetActive(true);

}

else

{

cellContainer.SetActive(true);

Time.timeScale = 0;

Cursor.visible = true;

Cursor.lockState = CursorLockMode.None;

Point.SetActive(false);

}

}

		Е.Е. КОИН			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Р.В. ПЕТРОСЯН				134
Змн.	Арк.	№ докум.	Підпис	Дата		

}

public void DisplayItems()

{

for (int i = 0; i < item.Count; i++)

{

Transform cell = cellContainer.transform.GetChild((i));

Transform icon = cell.GetChild(0);

Transform count = icon.GetChild(0);

TMP_Text txt = count.GetComponent<TMP_Text>();

Image img = icon.GetComponent<Image>();

if (item[i].id != 0)

{

img.enabled = true;

img.sprite = item[i].Icon;

if (item[i].countItem > 1)

{

txt.text = item[i].countItem.ToString();

}

		Е.Е. СОИ			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Р.В. ПЕТРОСЯН				135
Змн.	Арк.	№ докум.	Підпис	Дата		

```

else

{

    txt.text = null;

}

}

else

{

    img.enabled = false;

    img.sprite = null;

    txt.text = null;
}
}

}

}

```

		Е.Е. СОИН			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Р.В. ПЕТРОСЯН				136
Змн.	Арк.	№ докум.	Підпис	Дата		

Файл Item відповідає за предмети у інвентарю:

```
using System.Collections;
```

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

```
using UnityEngine;
```

```
using UnityEngine.Events;
```

```
public class Item : MonoBehaviour
```

```
{
```

```
    public string nameItem;
```

```
    public int id;
```

```
    public int countItem;
```

```
    public bool isStackable;
```

```
    [Multiline(5)]
```

```
    public string descriptionItem;
```

```
    public bool isRemoveable;
```

```
    public bool isDropped;
```

```
    public Sprite Icon;
```

```
    public UnityEvent customEvent;
```

```
}
```

		Е.Е. СОІН			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Р.В. ПЕТРОСЯН				137
Змн.	Арк.	№ докум.	Підпис	Дата		

Файл ToolType відповідає за відображення підказок:

```
using System;
```

```
using System.Collections;
```

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

```
using UnityEngine;
```

```
using UnityEngine.EventSystems;
```

```
public class ToolType : MonoBehaviour, IPointerEnterHandler, IPointerExitHandler
```

```
{
```

```
    private Inventory inventory;
```

```
    private void OnDisable()
```

```
    {
```

```
        inventory.toolTypeObj.SetActive(false);
```

```
    }
```

```
    // Start is called before the first frame update
```

```
    void Start()
```

```
    {
```

```
        inventory = GameObject.FindGameObjectWithTag("InventoryManager").GetComponent<Inventory>();
```

```
    }
```

		Е.Е. СОІН			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Р.В. ПЕТРОСЯН				138
Змн.	Арк.	№ докум.	Підпис	Дата		

```

// Update is called once per frame

void Update()

{

    inventory.toolTypeObj.transform.position    =    new    Vector2(Input.mousePosition.x    +    inventory.offsetX,
Input.mousePosition.y + inventory.offsetY);

}


public void OnPointerEnter(PointerEventData eventData)

{

    CurrentItem currentItem = GetComponent<CurrentItem>();

    Item item = inventory.item[currentItem.index];

    if (item.id != 0)

    {

        inventory.toolTypeObj.SetActive(true);

        inventory.icon.sprite = item.Icon;

        inventory.itemName.text = item.nameItem;

        inventory.itemDescriptprion.text = item.descriptionItem;

    }

}


public void OnPointerExit(PointerEventData eventData)

```

		Е.Е. КОИ			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Р.В. ПЕТРОСЯН				139
Змн.	Арк.	№ докум.	Підпис	Дата		

```
{  
  
    inventory.toolTypeObj.SetActive(false);  
  
}  
  
}
```

		Е.Е. СОИН			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Р.В. ПЕТРОСЯН				140
Змн.	Арк.	№ докум.	Підпис	Дата		