

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

ПОЯСНЮВАЛЬНА ЗАПИСКА

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

«Розробка гри «Player One, Get Ready!» в жанрі
Adventure-RPG засобами Unity для платформи
Windows»

Виконав студент групи КН-19-1
КОСТЮК Михайло Геннадійович

Керівник роботи:
МАРЧУК Галина Вікторівна

Рецензент:
КУЗЬМЕНКО Олександр Вікторович

Житомир – 2023

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

ЗАТВЕРДЖУЮ

Завідувач кафедри комп'ютерних наук

Марина ГРАФ

«05» січня 2023 р.

ЗАВДАННЯ

на кваліфікаційну роботу освітнього ступеня «бакалавр»

за спеціальністю 122 «Комп'ютерні науки»

(освітня програма «Комп'ютерна графіка та розробка ігор»)

Здобувач вищої освіти: КОСТЮК Михайло Геннадійович

Керівник роботи: МАРЧУК Галина Вікторівна

Тема роботи: Розробка гри «Player One, Get Ready!» в жанрі Adventure-RPG засобами Unity для платформи Windows»,

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

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

Вихідні дані роботи (зазначається предмет і об'єкт дослідження): використання рушія Unity 3D для розробки пригодницької-RPG гри з елементами поєднання 2D та 3D графіки.

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

Розділ	Консультант	Дата	
		Завдання видав	Завдання прийняв
1	Галина МАРЧУК	01.09.2022	01.11.2022
2	Галина МАРЧУК	01.11.2022	01.04.2023
3	Галина МАРЧУК	01.04.2023	01.05.2023

Календарний план

№ з/п	Назва етапів випускної кваліфікаційної роботи	Термін виконання етапів роботи	Примітка
1	Постановка задачі. Пошук, огляд та аналіз аналогічних розробок. Формулювання завдання. Опрацювання літературних джерел	15 вересня 2022 – 3 жовтня 2022	Виконано
2	Проектування структури	4 жовтня 2022 – 1 листопада 2022	Виконано
3	Написання програмного коду	1 листопада 2022 – 15 травня 2023	Виконано
4	Тестування системи	16 травня 2023 – 20 травня 2023	Виконано
5	Оформлення пояснювальної записки	21 травня 2023 – 1 червня 2023	Виконано
6	Попередній захист	20 червня 2023	Виконано

Здобувач вищої освіти Михайло КОСТЮК

Керівник Галина МАРЧУК

РЕФЕРАТ

Випускна кваліфікаційна робота бакалавра складається з програмного додатку та пояснювальної записки.

Пояснювальна записка до випускної роботи складається з вступу, трьох розділів, висновку, списку використаної літератури та додатку.

Метою роботи є розробка ігрового програмного додатку за допомогою рушія Unity 3D.

Текстова частина вкладена на 71 сторінках.

Пояснювальна записка має 12 сторінок додатків. Список використаних джерел містить 20 найменувань і займає 2 сторінки. В роботі наведено 53 рисунка та 2 таблиці. Загальний обсяг роботи - 83 сторінки.

Ключові слова: UNITY, ДОДАТОК, ГРА, ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ.

					КН.КР.Б – 122 – 23 – ПЗ			
Змн.	Арк.	№ докум.	Підпис	Дата	Розробка гри «Player One, Get Ready! » в жанрі Adventure-RPG засобами Unity для платформи Windows. Пояснювальна записка	Літ.	Арк.	Аркуші
Розроб.		М.Г. КОСТИЮК					4	83
Керівник		Г.В. МАРЧУК						
Реценз.		О.В. КУЗЬМЕНКО						
Н. Контр.								
Затверд.		М.С. ГРАФ				Житомирська політехніка, група КН-19-1		

ABSTRACT

The graduation thesis of the bachelor consists of a software application and an explanatory note.

The explanatory note to the final qualification work consists of an introduction, three sections, a conclusion, a list of used literature and an appendix.

The purpose of the work is to develop a gaming software application using the Unity 3D engine.

The text part is laid out on 71 pages of printed text.

The explanatory note has 12 pages of annexes. The list of used sources contains 20 names and occupies 2 pages. The work contains 53 figures and 2 tables. The total volume of work is 83 pages.

Keywords: UNITY, APPLICATION, GAME, SOFTWARE.

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

ЗМІСТ

РЕФЕРАТ.....	3
ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ.....	6
ВСТУП.....	7
РОЗДІЛ 1. ДОСЛІДЖЕННЯ ТА ОГЛЯД ЖАНРІВ КОМП'ЮТЕРНИХ ІГОР	10
1.1 Аналіз та класифікація жанрів комп'ютерних ігор.	10
1.2 Вибір інструментарію для розробки ПЗ	12
Висновки до першого розділу	16
РОЗДІЛ 2. ПРОЕКТУВАННЯ ТА РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ.....	17
2.1 Аналіз вимог до програмного продукту	17
2.2 Графічне представлення основних алгоритмів гри.....	18
2.3 Розробка ігрового додатку та основні методи	27
Висновки до другого розділу	50
РОЗДІЛ 3. ОПИС РОБОТИ З ПРОГРАМНИМ ДОДАТКОМ ТА ЙОГО ТЕСТУВАННЯ	52
3.1 Опис роботи з програмним додатком	52
3.2 Тестування роботи програмного забезпечення.....	66
Висновки до третього розділу.....	68
ВИСНОВКИ	69
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	70
ДОДАТКИ.....	72
Додаток А.....	73

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

U3D – Unity 3D.

ОС – Операційна система.

ПЗ – Програмне забезпечення.

UI – User Interface – інтерфейс, елементи, з якими взаємодіє користувач

NPC - Non-Player Character – персонажі, якими гравець не управляє, та з якими зазвичай можна взаємодіяти.

RTS - Real-time strategy – стратегія в реальному часі.

MMOFPS – Massively multiplayer online first-person shooter – масовий багатокористувацький шутер від першої особи.

МОВА - Multiplayer Online Battle Arena – багатокористувацька онлайнова бойова арена.

RPG – role-playing game – рольова гра.

MMORPG – Massively multiplayer online role-playing game – онлайнова рольова гра.

ЛКМ – Ліва кнопка миші.

ПКМ – Права кнопка миші.

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

ВСТУП

Актуальність теми. Протягом минулого десятиліття, світ став свідком експансії нового культурного напрямку – комп'ютерних ігор.

Сучасний розважальний сектор є одним із найбільш прибуткових, а початок ери інформаційних технологій тільки пришвидшив цей розвиток, просуваючи глобальне співтовариство до нового майбутнього. Це майбутнє, в якому комп'ютерні системи безпосередньо вдосконалюються, з метою не лише полегшення повсякденного життя людини, але й забезпечення її дозвілля.

Роль комп'ютерів у житті сучасної людини постійно зростає. Вони стають не лише могутніми інструментами для обчислень, але і центром розваг. Традиційні сфери розваг, такі як кіно, театр та література стикаються з викликами, поставленими новими інтерактивними розвагами, пристроями доповненої реальності та іншими технологіями, впровадженими завдяки розвитку комп'ютерних технологій.

Комп'ютери все більше проникають в наше життя. Вони відчутно впливають на наші звички та стиль життя.

Комп'ютерні ігри додають багатства та насиченості в наше життя, що, в свою чергу, перетворює їх в могутню та молоду економічну сферу, генеруючи великі доходи. Не дивно, що комп'ютерні ігри, що виникли на зорі ери інформаційних технологій, відіграють важливу роль у нашому суспільстві.

Можна вважати комп'ютерні ігри новим видом мистецтва, що має багато спільного з іншими візуальними жанрами. Ігри можуть не тільки розважати, але й заохочувати до роздумів, викликати емоції, порушувати важливі глобальні та психологічні питання. Таким чином, ігри можна вважати сучасним видом мистецтва, що викликає не менше емоцій, ніж перегляд кінофільмів або театральних постановок.

Розробка комп'ютерних ігор - це складний процес, що вимагає великої кількості різноманітних навичок і глибокого розуміння аудиторії, для якої вони створюються.

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

Метою створення випускної кваліфікаційної роботи є проектування та розробка комп'ютерної гри, яка поєднує елементи візуальної новели, RPG та пригодницького жанру, використовуючи рушій Unity 3D. Гра буде ефективно використовувати 2D та 3D графіку для створення ігрових локацій.

Специфічні цілі роботи включають:

- розробку сценарію та історії для ігрового досвіду;
- реалізацію RPG механік;
- створення пригодницьких елементів на рівнях, з метою дослідження локацій;
- використання Unity 3D для створення гри, яка поєднує 2D і 3D графіку;
- тестування і оптимізація гри для забезпечення стабільної роботи додатку.

Об'єктом дослідження є процес проектування та розробки ПЗ, що використовує ігровий рушій Unity 3D і поєднує елементи візуальної новели, RPG та пригодницького жанру.

Предметом дослідження є використання рушія Unity 3D для створення комп'ютерної гри, яка поєднує в собі декілька жанрів. Особлива увага звертається на оповідання сюжету гри.

В роботі були застосовані принципи об'єктно-орієнтованого програмування.

Практичне значення полягає в тому, що результат роботи може бути використаний, як приклад для розробки гри з поєднанням різних ігрових жанрів або просто для розваги.

Теоретична значущість дослідження полягає в аналізі жанрів комп'ютерних ігор та їх класифікації. Вивчення жанрів комп'ютерних ігор є важливим аспектом розуміння і визначення їхніх особливостей, характеристик та механік. Дослідження різних жанрів ігор дозволяє виявити спільні риси та відмінності між ними, а також з'ясувати, які жанри викликають найбільший інтерес у гравців. Класифікація жанрів комп'ютерних ігор допомагає структурувати розуміння цієї широкої сфери розваг та дослідити різні підходи до опису та категоризації ігрових жанрів.

Вивчення жанрів ігор сприяє розширенню розуміння різноманітності інтерактивних досвідів, які надають комп'ютерні ігри. Це дозволяє глибше проаналізувати елементи гри, такі як сюжет, механіки, графіка та звуковий дизайн, і вивчити їхні взаємозв'язки та вплив на гравців.

Таке дослідження є актуальним, оскільки з розвитком комп'ютерних технологій постійно з'являються нові жанри ігор або змінюються вже існуючі.

В рамках теми кваліфікаційної роботи бакалавра було опубліковано тези: Костюк М.Г., Марчук Г.В. ОСОБЛИВОСТІ РОЗРОБКИ ІГОР ЖАНРУ ВІЗУАЛЬНОЇ НОВЕЛИ. XIII Міжнародна науково-технічна конференція. Інформаційні комп'ютерні технології - 2023. 30-31 березня 2023 р. м. Житомир.

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

РОЗДІЛ 1. ДОСЛІДЖЕННЯ ТА ОГЛЯД ЖАНРІВ КОМП'ЮТЕРНИХ ІГОР

1.1 Аналіз та класифікація жанрів комп'ютерних ігор.

Комп'ютерні ігри сьогодні є невід'ємним елементом глобальної культури та індустрії розваг. Вплив відеоігор на сучасне суспільство настільки глибокий, що нині спостерігається тривале зростання гейміфікації в програмному забезпеченні, яке не призначене для ігор.

Така тенденція видна в освітніх установах по всій Європі, де використовують популярні відеоігри, як навчальні інструменти. Військова промисловість, космічні агенства та авіація також активно використовує симулятори для тренування. Кіберспорт отримує офіційне визнання, як вид спорту в багатьох країнах. А в США у 2011 році, відеоігри були офіційно визнані формою мистецтва, порівняно з театром і кіно.

У зв'язку з тим, що критерії для класифікації ігор за жанрами не є ясно визначеними, систематизація комп'ютерних ігор є недостатньою, і дані про жанрову належність конкретної гри можуть відрізнятися в різних джерелах. Однак, в ігровій спільноті розробників існує загальний згоджений підхід, і визначення основного жанру для гри майже завжди можна зробити без сумнівів.

Основний критерій розподілу на жанри – це типові дії, які гравець виконує у межах ігрового процесу. Визначено ключові елементи геймплею, які формують основу будь-якої комп'ютерної гри: навчальний етап, розгадування головоломок, міжперсональна взаємодія, рольова гра, дослідження, збирання об'єктів, уникнення перешкод, знищення, змагальність, технічний аспект, догляд, прогресія, управління, тактичні рішення, стратегічне планування. Ці жанри були класифіковані в три різні класифікації (Табл. 1.1).

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

Класифікація відеоігор за жанрами

Жанр за геймплеєм	Жанр за наративом або тематикою	Жанр за типом взаємодії
Action	Adventure	Single-player
Platformer	Horror	MMO
RPG	Fantasy	RTS
Survival	Military/War	Virtual Reality (VR)
MOBA	Cyberpunk	Augmented Reality (AR)
MMORPG	Space Exploration	Multiplayer
MMOFPS	Science Fiction	Cooperative
Puzzle	Education	Visual Novel

Жанр за геймплеєм. До нього входять такі типи геймплею, як:

- розгадування головоломок;
- рольова гра;
- уникнення перешкод;
- тактичні рішення;
- стратегічне планування.

Жанр за наративом або тематикою відноситься до того, як історія або тема гри розгортається. Він може включати такі елементи, як:

- дослідження;
- збирання предметів;
- знищення;
- прогресія.

Жанр за типом взаємодії описує, як гравці взаємодіють один з одним або з грою. Може включати такі типи взаємодії, як:

- міжперсональна взаємодія;
- технічні аспекти;
- управління.

Завдяки цій класифікації встановлюються жанри, які будуть присутні у грі. Також усі відеоігри поділяються на Single-player або multiplayer. Multiplayer також поділяється на:

- Локальний мультиплеєр;
- Онлайн.

Single-player відноситься до режиму відеоігри, в якому лише один гравець може грати за один час. У таких іграх гравець зазвичай взаємодіє з грою, виконуючи завдання або борючись проти персонажів, якими керує комп'ютер (таких як NPC, або Non-Player Characters). У синглплеєр гри, фокус зазвичай лежить на розгортанні історії, дослідженні світу гри, або виконанні різноманітних завдань.

Після ретельного аналізу класифікацій комп'ютерних ігор, було виявлено великий потенціал в поєднанні жанрів Adventure-RPG, Platformer та Visual Novel. Кожен з цих жанрів має свої унікальні характеристики геймплею, які можуть спільно створювати цікавий ігровий досвід, який може викликати інтерес широкого кола гравців.

1.2 Вибір інструментарію для розробки ПЗ

При розробці відеоігри важливо ретельно обрати набір інструментарію і технологій, які будуть використовуватись в процесі створення. Вибір ігрового рушія залежить від багатьох факторів, включаючи цілі проекту, технічні потреби, вимоги до якості та доступності ресурсів.

Ігрові рушії допомагають програмістам створювати ігри, обробляючи багато складних завдань, таких як рендеринг графіки, обробка фізики, аудіо, та інших складних систем.

Великі компанії, що розробляють відеоігри, часто використовують власні ігрові рушії. Це дозволяє їм мати більше контролю над розробкою, а також оптимізувати рушій під конкретні потреби розробників. Як приклад можна використати ігровий рушій “**Frostbite**” компанії DICE.

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

Frostbite: це ігровий рушій, розроблений компанією DICE, дочірньою компанією EA. Frostbite використовується в багатьох іграх EA, включаючи серії ігор Battlefield та FIFA(рис.1.1).

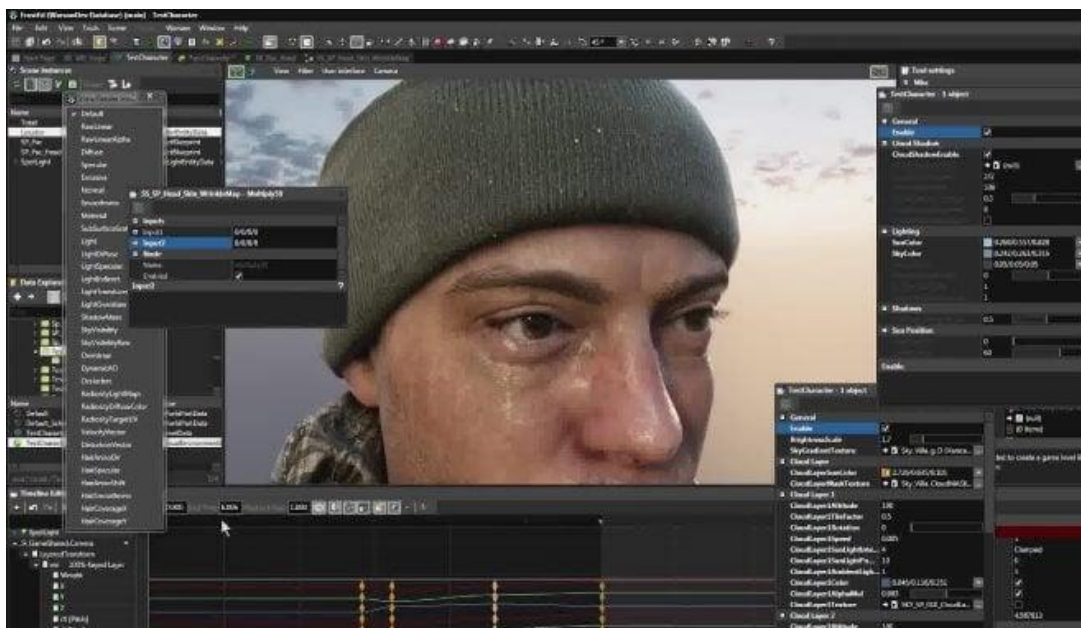


Рисунок 1.1-Рушій Frostbite[12]

RAGE (Rockstar Advanced Game Engine) - це ігровий рушій, який було розроблено компанією Rockstar Games для створення їх розгорнутих, відкритих світів. Rockstar використовує цей рушій для створення більшості своїх ігор з часу випуску Grand Theft Auto IV(рис 1.2).

Цей рушій дозволяє створювати вражаючі графічні ефекти, в тому числі деталізовані відкриті світи, реалістичну анімацію персонажів і складну фізику. Це забезпечує гравцям глибоке і занурювальне досвід, що робить ігри Rockstar такими популярними.

Однак, RAGE не продається або не надається в оренду іншим розробникам, він виключно використовується Rockstar для внутрішніх проектів. Це дозволяє Rockstar мати повний контроль над їх розробкою і забезпечувати, що їх ігри максимально використовують можливості рушія.



Рисунок 1.2 – Логотип рушія Rage[13]

Найбільш популярним сьогодні є ігровий рушій **Unreal Engine (UE4)**(рис.1.3). Рушій розроблений компанією Epic Games, цей рушій використовується у багатьох AAA-іграх, таких як серії ігор Gears of War і Fortnite.

Unreal Engine підтримує велику кількість платформ, включаючи Windows, macOS, Linux, PlayStation, Xbox, Nintendo Switch, iOS, Android, VR і багато інших. Він пропонує широкий спектр інструментів для розробки, включаючи візуальне програмування через систему Blueprints, а також можливість написання коду на C++.

Однією з ключових особливостей Unreal Engine є його потужний редактор 3D-графіки, який дозволяє розробникам створювати імпресивні візуальні ефекти та деталізовані сцени.



Рисунок 1.3 – Рушій Unreal Engine[11]

Unity 3D - це універсальний ігровий рушій, розроблений компанією Unity Technologies. Unity став надзвичайно популярним серед розробників відеоігор у всьому світі завдяки своїй гнучкості, великій кількості функцій та простоті використання. Однією з особливостей Unity є його підтримка величезної кількості платформ, що охоплює практично всі сучасні операційні системи, консолі для відеоігор, мобільні пристрої, а також VR і AR системи. Unity дозволяє розробникам писати код на C# або JavaScript, що робить його доступним для широкого кола розробників. Unity включає багато інструментів для розробки 3D і 2D ігор, включаючи потужний редактор графіки, системи фізики, AI, анімації та аудіо. Завдяки вбудованій підтримці Asset Store, розробники мають змогу легко імпортувати додаткові ресурси та інструменти безпосередньо у свої проекти. Unity широко використовується не тільки для створення відеоігор, але і для розробки інтерактивного контенту в таких галузях, як освіта, архітектура, медицина та кіноіндустрія(рис.1.4).

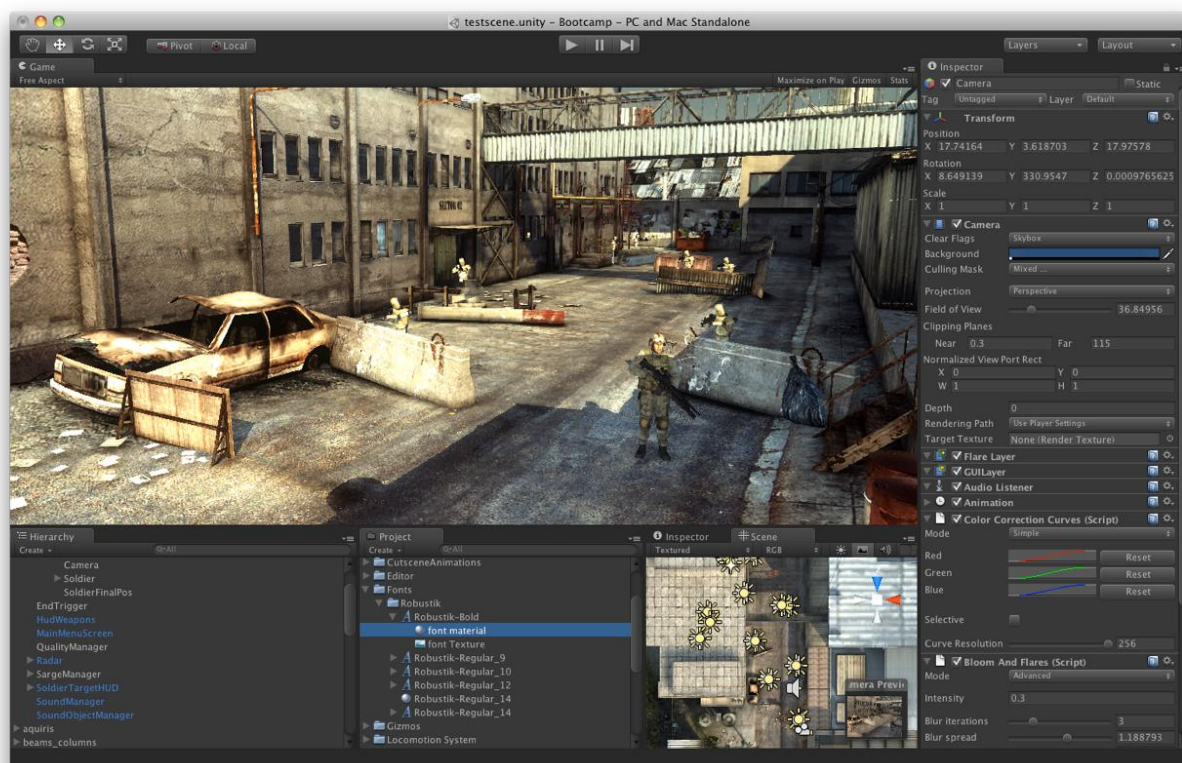


Рисунок 1.4 – Рушій Unity 3D[14]

Висновки до першого розділу

Аналізуючи сучасний стан ринку відеоігор, було виявлено величезний потенціал у гібридному поєднанні жанрів Adventure-RPG, Platformer та Visual Novel. Кожен з цих жанрів володіє своїми унікальними властивостями, що в контексті поєднання можуть створити цікавий та унікальний ігровий досвід, що відрізнятиметься від більшості сучасних ігор.

Щодо технічної сторони проекту, рушій Unity 3D було вибрано як основний інструмент розробки, завдяки його універсальності, широким можливостям, підтримці багатьох платформ та великої спільноти розробників. Unity 3D забезпечує всі необхідні інструменти для створення високоякісних 2D та 3D ігор, що робить його оптимальним рішенням для проекту.

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

РОЗДІЛ 2. ПРОЕКТУВАННЯ ТА РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Аналіз вимог до програмного продукту

Функціональні вимоги:

Управління персонажем залежить від поточної сцени. Реалізоване керування за допомогою клавіш A, S, D, Space, Left mouse button, Right mouse button. Взаємодія з інвентарем – J, виклик меню паузи – ESC, клавіші взаємодії – E, F.

Геймплей: користувач матиме можливість ознайомитися з трьома різними жанрами відеоігор в одному проекті. Гравець зможе обирати варіанти діалогу або дії головного персонажа, тобто гра буде інтерактивна. Гравець зможе виконувати квести і використовувати ігрові механіки для проходження рівнів.

Гра являється single-player проектом, тому під час гри користувач може перебувати в режимі офлайн.

Не функціональні вимоги:

Час відповіді системи на дії гравця зазвичай не повинен перевищувати 1-2 секунди.

Інтерфейс програмного додатку повинен бути простим і інтуїтивно зрозумілим для гравця.

Гра повинна підтримувати безперебійну роботу.

Системні вимоги:

- ОС: Windows 8
- Процесор: Intel Core i3-3220 або аналог від AMD
- Оперативна пам'ять: 8 GB
- Внутрішня пам'ять: від 1 до 3 GB
- Відеокарта: NVIDIA GeForce GTX 1050 або AMD Radeon RX 550

Вимоги користувача:

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

Користувач має доступ до наступних функцій:

- розпочати нову гру;
- продовжити гру;
- зберегти гру поточного рівня;
- налаштувати аудіо параметри;
- керувати персонажем;
- обирати варіанти відповіді;
- вихід з ігрового додатку.



Рисунок 2.1 – Вимоги користувача

2.2 Графічне представлення основних алгоритмів гри

Проектування програмного додатку є одним з найважливіших етапів у процесі розробки програмного забезпечення. Воно передбачає розробку архітектури та дизайн системи, моделювання взаємодії між її компонентами та визначення основних алгоритмів і даних, які будуть використовуватися.

Цей етап починається з формлювання вимог до системи, що базуються на потребах користувача або бізнес-вимогах. Після цього, на основі визначених вимог, створюється детальний план або проект системи.

Однією з ключових частин проектування є створення діаграм, що відображають взаємодію між компонентами системи та алгоритми роботи

окремих модулів додатку. Це може включати діаграми класів для відображення структури даних, діаграми послідовностей для відображення взаємодій між об'єктами та діаграми активності для відображення потоку виконання в системі.

В рамках розробки кваліфікаційної роботи, одним з основних елементів програмного додатку є головне меню. Головне меню додатку зустрічає користувача при кожному вході у гру. На рисунку 2.1 наведено алгоритм роботи головного меню гри використовуючи діаграму активності. Діаграма активності, представлена на рисунку, ілюструє основні дії та потоки взаємодії користувача в головному меню комп'ютерної гри, розробленої в рамках дослідження.

Розпочинаючи взаємодію з додатком, користувач потрапляє до головного меню. Це служить вступним пунктом для всіх подальших дій в системі.

Система автоматично перевіряє наявність збереженого стану гри. Це реалізовано через умовний блок, який визначає, чи існує відповідний файл збереження. Якщо такий файл існує, користувачу пропонується продовжити попередню гру.

Наступна дія в меню – можливість почати нову гру. Це є стандартною опцією для всіх відеоігор, що дозволяє користувачам почати гру з початку.

Після цього користувачу надається доступ до налаштувань гри. В цьому розділі користувач може змінити параметри гри, такі як звук та ознайомитися з управлінням.

Накінець користувачу пропонується вийти з гри. Ця опція завершує роботу програми. Алгоритм поведінки представлено на рисунку 2.2.



Рисунок 2.2 – Алгоритм поведінки головного меню

На діаграмі враховано, що при відсутності збереженого файлу, користувачу надається можливість лише розпочати нову гру, перейти до налаштувань або вийти з гри. Опція «Продовжити» доступна лише у випадку, коли існує збережений файл гри.

В основі програмного додатку лежить гра у форматі візуальної новели, де основним компонентом є інтерактивний наратив. Цей жанр вибрано завдяки його здатності забезпечити глибоку іммерсію в історію і високий рівень взаємодії з користувачем.

При виборі опції «Нова Гра», гравець одразу занурюється у першу сцену гри. Алгоритм поведінки гри та взаємодії з гравцем був викладений за допомогою діаграми активності UML. Стартовою точкою алгоритму є «Scene-1», що відповідає початку гри.

У процесі гри, гравець може продовжити історію натискаючи ліву кнопку миші. Цей вибір автоматично переносить гравця до наступного сегменту історії, але також відкриває можливість для додаткових виборів, які можуть змінити хід історії. Альтернативно, гравець може натиснути клавішу "Esc" для виклику меню паузи. У меню паузи пропонуються наступні варіанти дій: відновити гру, зберегти гру, перейти до налаштувань, повернутися до головного меню або вийти з гри. Алгоритм поведінки представлено на рисунку 2.3.

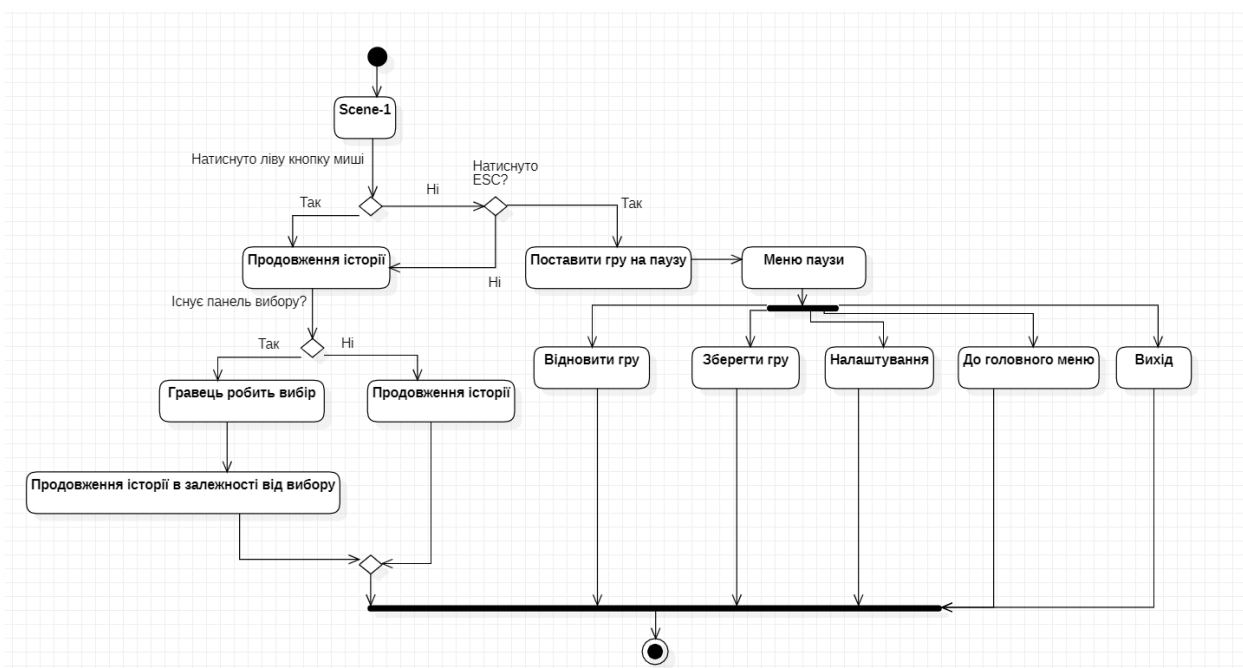


Рисунок 2.3 – Алгоритм поведінки Scene-1

Рухаючись далі, після завершення Scene-1, гра переходить до Scene-2, яка представляє собою 2D платформер. При переході на цю сцену, гравець стикається з новим набором можливостей та взаємодій. На цьому етапі гравець має можливість обирати персонажів та перемикатися між двома персонажами: Лисом та Солдатом. Це дає гравцеві нові можливості з ігровим середовищем: Солдат, наприклад, може стріляти, тоді як Лис не має такої можливості. Але колайдери Лиса менші, і Лис може потрапляти в місця, куди Солдат ніяк не пролізе.

Також гравцю відкривається можливість взаємодії з NPC, використовувати інвентарь, взаємодіяти зі світом, та виконувати квести. Ці

елементи вносять більшість змін та різноманітності в геймплей, надаючи гравцеві більше волі та контролю над своїм досвідом від гри.

Якщо всі квести були виконані, гравець переходить на Scene-3. Якщо ж квести не виконані, гра на Scene-2 продовжується.

Алгоритм поведінки, який регулює перехід та взаємодію гравця з Scene-2, представлений на рисунку 2.4.

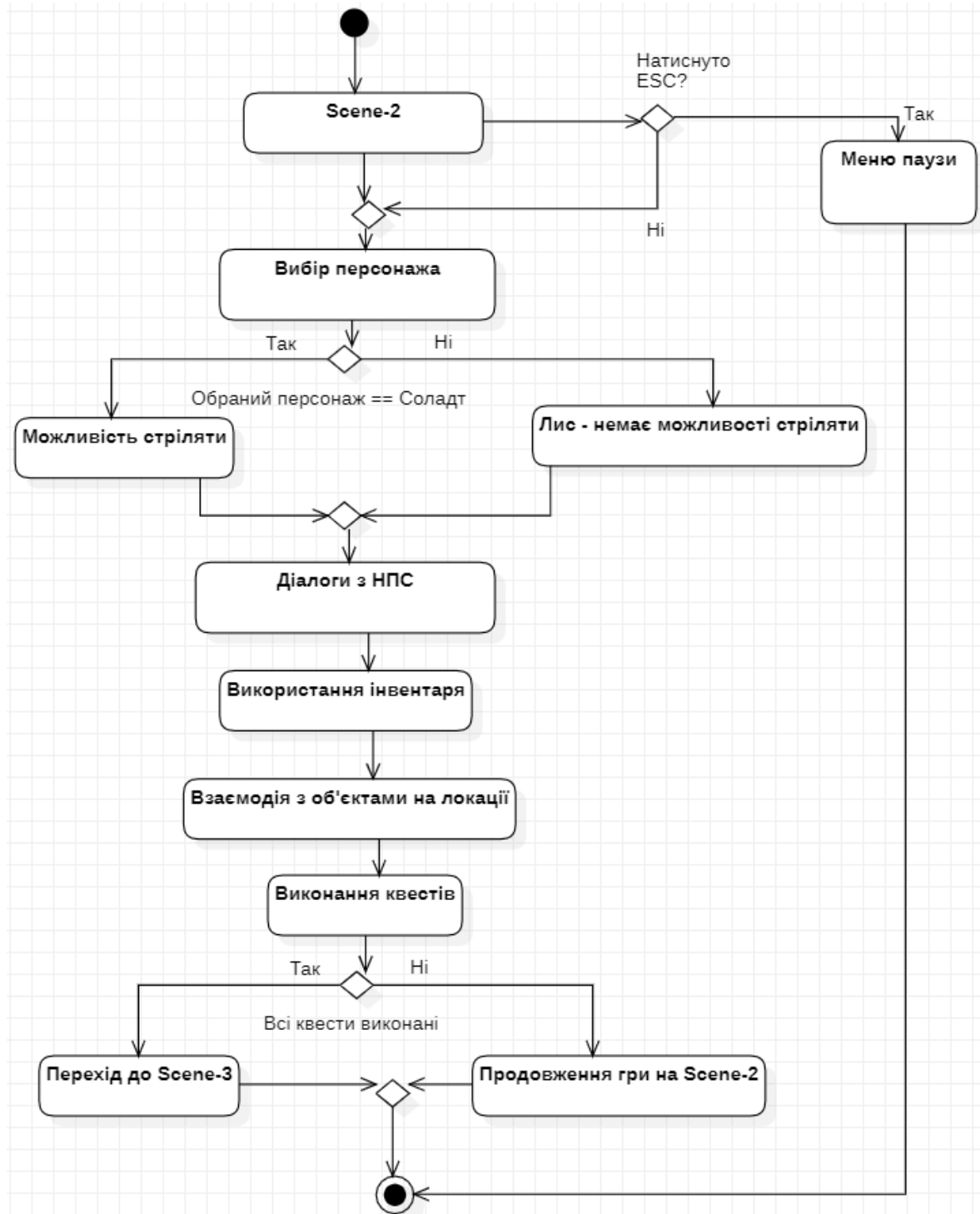


Рисунок 2.4 – Алгоритм проходження Scene-2

Після завершення Scene-2, гравець переходить до Scene-3, яка представляє собою 3D RPG. На цій стадії, гра розширює свій набір взаємодій та можливостей для гравця.

Гра пропонує гравцеві розбавити вже відомі ігрові механіки декількома новинками. Ці елементи допомагають підтримувати інтерес гравця до гри, а також збагачують ігровий досвід через різноманітні сценарії та задачі.

Подальша взаємодія з ігровим світом здійснюється через систему «Interected» та інвентаря, які дозволяють гравцю взаємодіяти з різними предметами на NPC's у грі. Предмети можуть бути або звичайними предметами класу Equipment, які можна вдягнути на персонажа, видалити з інвентаря, або Quest Item, які потрібні для активації, або завершення квесту. Предмети Quest Item видалити з інвентаря неможливо.

Важливою особливістю цього рівня є наявність ворогів, з якими гравець може вступати в бойові зустрічі. У випадку перемоги гравця, ворог зникає з ігрового світу. Якщо ж ворог перемагає, гравцеві пропонується можливість відновити гру з останнього місця збереження. Після виконання всіх квестів на Scene-3, гравець потрапляє до Scene-4.

Описаний алгоритм взаємодії гравця з Сценою-3 представлено на рисунку 2.5.

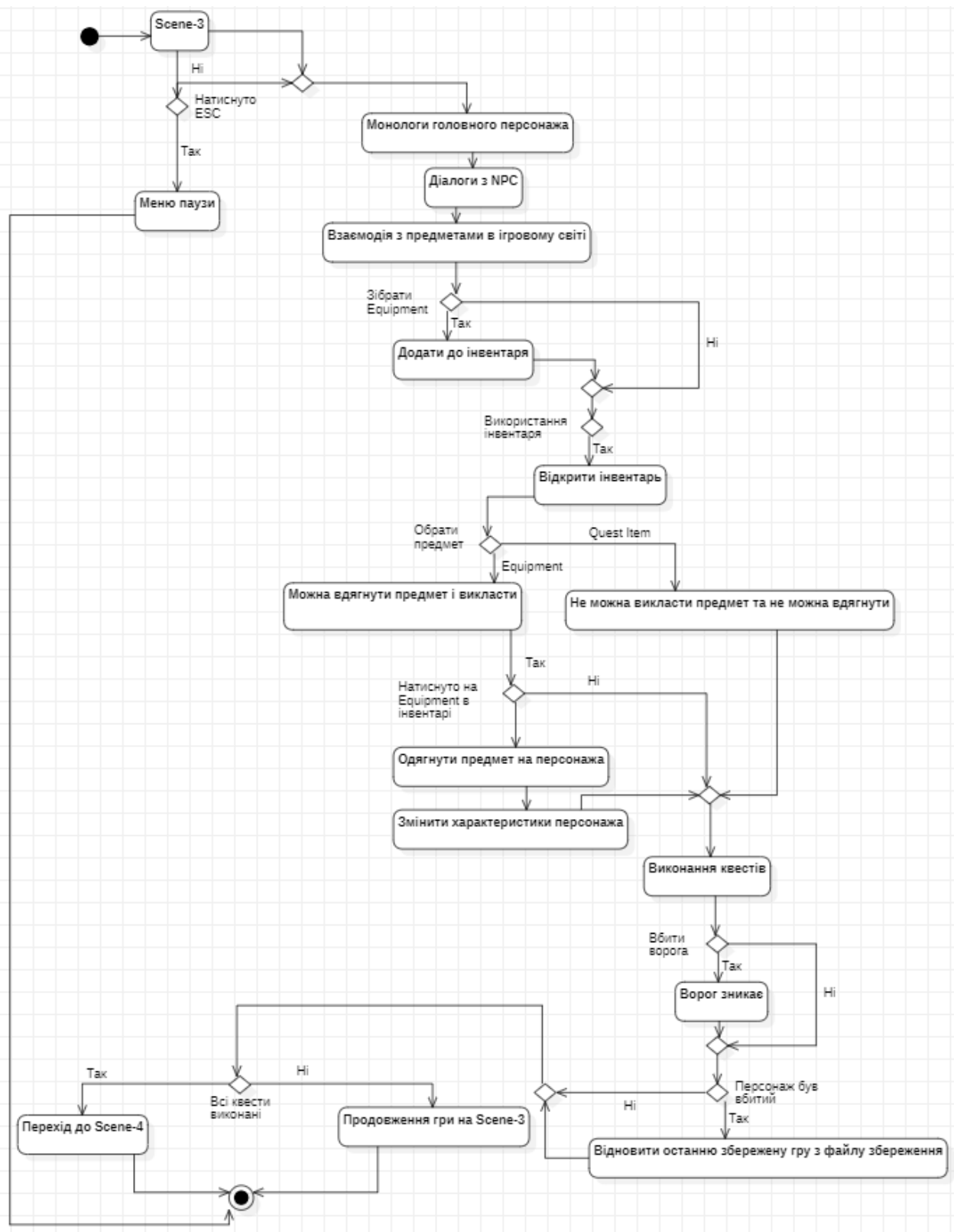


Рисунок 2.5 – Алгоритм Scene-3

Після завершення всіх активностей на Scene-3, гравець переходить до Scene-4, яка є зміненою версією Scene-2. Механіки Scene-4 в основному аналогічні з Scene-2, але алгоритм дій гравця відрізняється.

Спочатку відбувається завантаження сцени. Після цього гравець може рухатися по локації, подібно до Scene-2.

Коли гравець наближається до NPC, він може ініціювати діалог. Цей діалог є ключовим переходом на наступну сцену. Якщо діалог з NPC не завершено, гра продовжується на Scene-4. Але як тільки діалог завершено, автоматично відбувається завантаження Scene-5.

Детальний алгоритм гравця на Scene-4 представлено на рисунку 2.6.

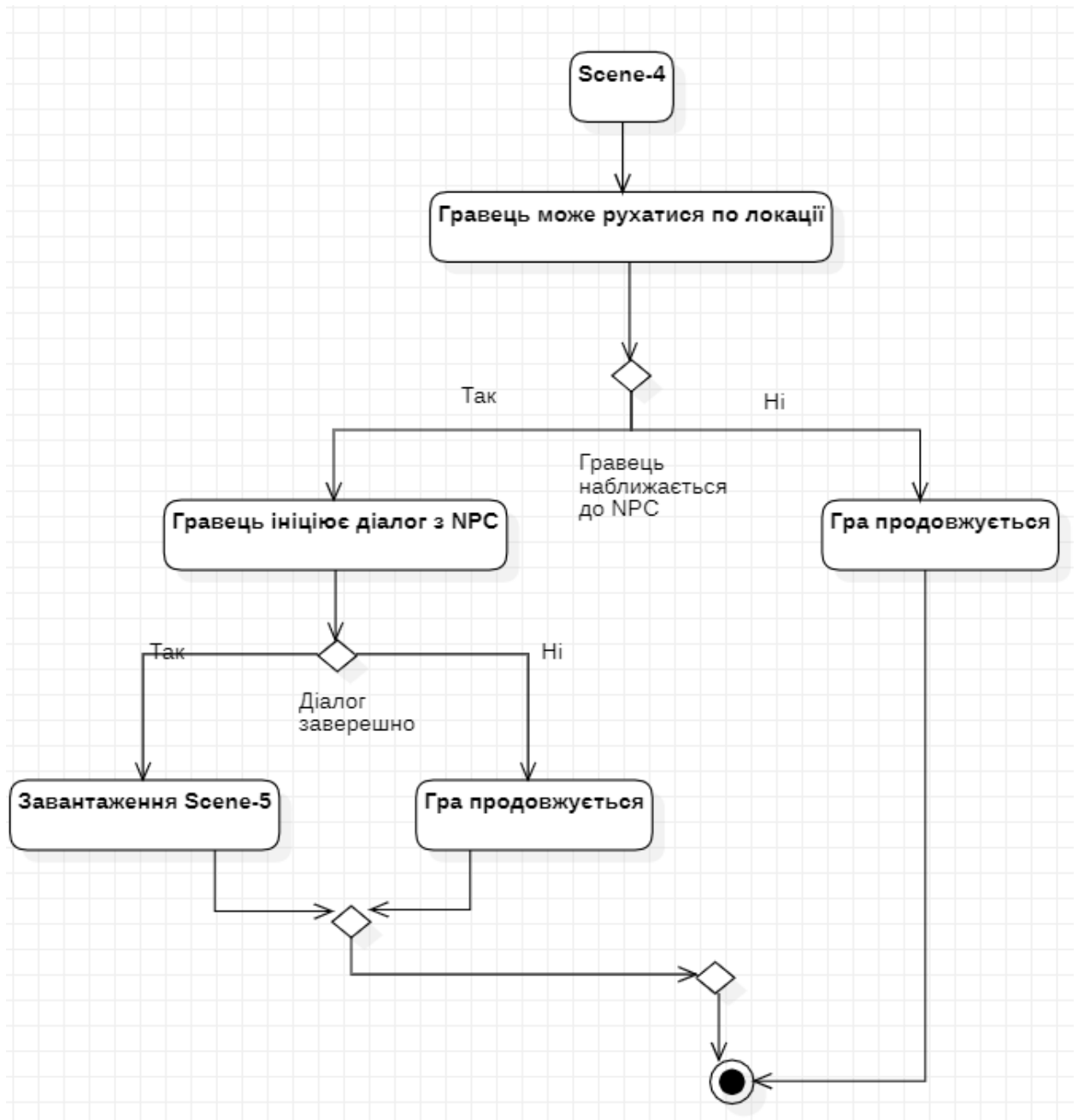


Рисунок 2.6 – Алгоритм Scene-4

Scene-5 повертає гравця до геймплею в стилі візуальної новели, відтворюючи механіки та алгоритми Scene-1. Гравець просувається через сюжет, вибираючи різні діалогові лінії та дії, що впливають на подальший розвиток сюжету. Як і на Scene-1, гравець може продовжувати історію натискаючи ліву кнопку миші. Завершивши Scene-5, гравець переходить до фінальної Scene-6. Ця сцена відображає фінальні титри гри, завершуючи гравецький досвід та підсумовуючи його подорож через різні сцени та механіки гри.

Додатковою важливою функцією, доступною гравцю на будь-якому з доступних рівнів гри, є меню паузи. Меню паузи може бути викликане в будь-який момент гри натисканням клавіші "ESC". Воно пропонує ряд опцій, що дозволяють гравцю керувати станом гри.

Першою з цих опцій є можливість відновити гру. Якщо гравець вибирає цю опцію, гра продовжується з місця, де було натиснуто клавішу ESC.

Другою опцією є можливість зберегти поточний стан гри. Після збереження, гра продовжується без перерв.

Третя опція - це перехід до налаштувань. Тут гравець може змінити різноманітні параметри гри, такі як звук і керування.

Четвертою опцією є вихід до головного меню. Це дозволяє гравцю вийти з поточної гри та повернутися до головного меню, звідки він може почати нову гру, завантажити збережену гру або змінити налаштування.

Остання опція - це вихід з гри. Це повністю завершує гру та закриває програму.

Ця гнучкість дозволяє гравцям керувати своїм ігровим часом та насолоджуватися грою за власним розкладом.

Алгоритм роботи меню паузи наведено на рисунку 2.7.

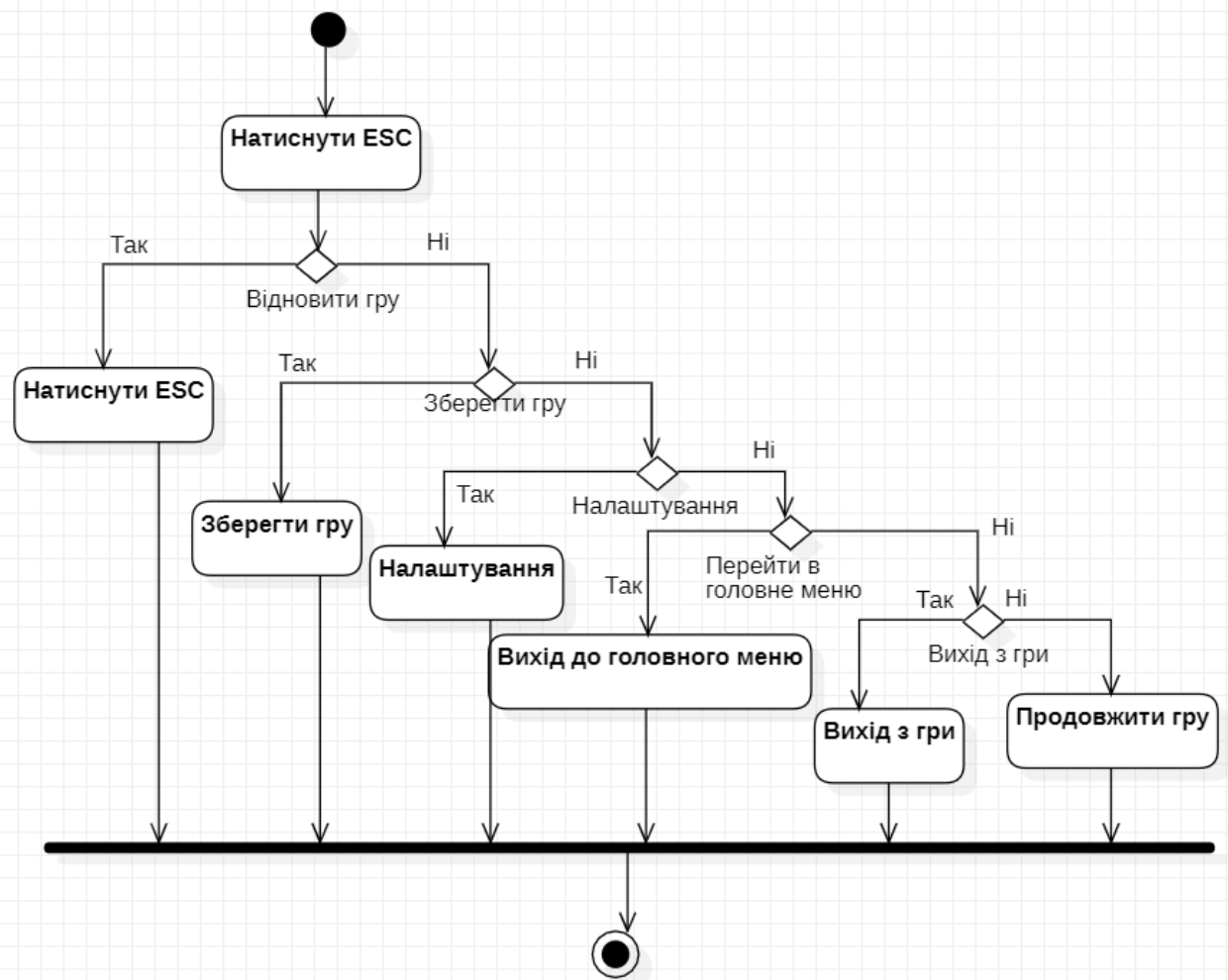


Рисунок 2.7 – Алгоритм роботи меню паузи.

2.3 Розробка ігрового додатку та основні методи

Почнемо з розгляду головного контролера меню. Контролер головного меню - це клас, який керує різними функціями, пов'язаними з головним меню гри. Цей клас був розроблений за допомогою мови програмування C# в середовищі Unity.

Клас **MainMenuController** управляє головним меню гри, надаючи користувачу можливість почати нову гру, завантажити збережену гру або вийти з гри. Він використовує ряд різних компонентів Unity, включаючи Image, Button, TextMeshProUGUI та кілька інших. Ці компоненти використовуються для створення візуальних та аудіо ефектів головного меню.

В класі MainMenuController два основних методи, які запускають нову гру та завантажують збережену гру, це PlayGame() та Load().

Метод `PlayGame()` використовується для запуску нової гри. Спочатку він викликає метод `CacheObjects()`, що очищує кеш з ресурсами, які в даний час не використовуються. Після цього він викликає метод `ClearSavedGame()` з класу `SaveManager` для видалення будь-якої існуючої збереженої гри. Він зупиняє відтворення музики через `audioController.musicSource.Stop()` та зупиняє анімацію поїзда `trainAnimation.StopTrainAnimation()`. Нарешті, він запускає корутину `LoadNewGame()` для асинхронного завантаження наступної сцени, що означає початок нової гри.

Метод `Load()` використовується для завантаження збереженої гри. Спочатку він зупиняє анімацію поїзда та відтворення музики, аналогічно до методу `PlayGame()`. Потім він викликає корутину `LoadSavedGame()`, що служить для асинхронного завантаження збереженої гри.

Корутина `LoadSavedGame()` використовує `SceneManager` для асинхронного завантаження наступної сцени. Спочатку вона отримує дані збереження через `SaveManager.LoadGame()`. Потім вона обробляє ці дані, щоб відновити стан гри до моменту збереження. Це включає відновлення позицій персонажів, індексу діалогу, індексу репліки в діалозі та інше. Після цього вона асинхронно завантажує сцену, в якій була збережена гра. Під час завантаження сцени відображається візуальний індикатор завантаження, і коли завантаження завершено, гравець може продовжити гру, натиснувши клавішу `E`.

Корутина `LoadNewGame()`, схожа на `LoadSavedGame()`, також використовує `SceneManager` для асинхронного завантаження наступної сцени. Однак вона завантажує сцену для нової гри, а не збереженої. Так само як і в `LoadSavedGame()`, вона відображає індикатор завантаження, і гравець може почати гру, натиснувши клавішу `E`, коли завантаження завершено.

Корутини для завантаження гри представлені на рисунку 2.8.

```

1 reference
private IEnumerator LoadSavedGame()
{
    int sceneToLoad = SceneManager.GetActiveScene().buildIndex + 1; // Змінна для визначення наступної сцени для завантаження
    SaveData data = SaveManager.LoadGame(); // Завантажуємо дані збереження
    if (SaveManager.IsGameSaved())
    {
        ///... Завантажуємо дані збереження
        sceneToLoad = data.currentScene != 0 ? data.currentScene : sceneToLoad; // Вибираємо сцену для завантаження

        // Завантажуємо сцену асинхронно
        AsyncOperation asyncLoad = SceneManager.LoadSceneAsync(sceneToLoad, LoadSceneMode.Single);
        loadingImage.gameObject.SetActive(true);
        asyncLoad.allowSceneActivation = false;
        while (!asyncLoad.isDone) // Показуємо загрузку
        {
            float progress = Mathf.Clamp01(asyncLoad.progress / 0.9f);
            float fillAmount = progress * 360f;
            circleImg.fillAmount = fillAmount / 360f;
            if ([asyncLoad.progress >= 0.9f])
            {
                if (Input.GetKeyDown(KeyCode.E))
                {
                    asyncLoad.allowSceneActivation = true;
                }
                pressE.gameObject.SetActive(true);
            }
            yield return null;
        }
    }
}

1 reference
IEnumerator LoadNewGame()
{
    // Завантажуємо нову гру
    AsyncOperation asyncLoad = SceneManager.LoadSceneAsync(SceneManager.GetActiveScene().buildIndex + 1);
    loadingImage.gameObject.SetActive(true);
    asyncLoad.allowSceneActivation = false;
    while (!asyncLoad.isDone)
    {
        float progress = Mathf.Clamp01(asyncLoad.progress / 0.9f);
        float fillAmount = progress * 360f;
        circleImg.fillAmount = fillAmount / 360f;
        if (asyncLoad.progress >= 0.9f)
        {
            if (Input.GetKeyDown(KeyCode.E))
            {
                asyncLoad.allowSceneActivation = true;
            }
            pressE.gameObject.SetActive(true);
        }
        yield return null;
    }
}

```

Рисунок 2.8 – Корутини для завантаження гри

Також у головному меню проекту використовуються анімація, яка виконуються за допомогою корутин у Unity.

Клас TrainAnimation відповідає за управління анімацією поїзда. Ця анімація включає в себе зміну прозорості знаків, поворот бар'єрів, рух поїзда, а також відтворення звукових ефектів. Всі ці елементи взаємодіють між собою за допомогою корутин, що дозволяє плавно керувати перехід між окремими елементами анімації.

Під час кожного циклу анімації, поїзд спочатку виїжджає з правої сторони, потім повертається назад. Цей цикл повторюється нескінченно.

Клас `CloudsAnimation` відповідає за анімацію хмар. Хмари переміщуються зліва направо і назад, створюючи ілюзію динамічного неба. Подібно до `TrainAnimation`, ця анімація реалізована за допомогою корутини, що переміщає хмари в одному або іншому напрямку в залежності від їх поточної позиції.

Клас `SpriteFadeInOut` є відповідальним за анімацію зникнення та з'явлення лучів сонця на сцені. Анімація реалізована за допомогою зміни прозорості (альфа-компоненту) спрайту за допомогою корутини.

Під час запуску, компонент `SpriteRenderer` отримується з поточного об'єкта, і його альфа-компонент спочатку встановлюється на нуль, що робить спрайт повністю прозорим.

Корутина `FadeInOut` постійно виконується, чергуючи між двома станами: з'явленням та зникненням спрайту. Під час з'явлення, альфа-компоненту від 0 змінюється до 1 протягом певного періоду часу (`fadeDuration`), роблячи спрайт повністю непрозорим. Після невеликої паузи (`waitDuration`), відбувається зникнення, під час якого альфа-компонент спрайту змінюється від 1 до 0, роблячи спрайт знову прозорим. Цикл потім повторюється знову і знову, створюючи ефект пульсації лучів сонця.

Ці три класи демонструють використання корутин в Unity для створення складних і динамічних анімацій на сцені, контролюючи різні елементи гри, такі як поїзд, хмари та сонячні промені, і створюючи цікаві візуальні ефекти.

На рисунку 2.9 зображено діаграму класів, які використовуються в головному меню.

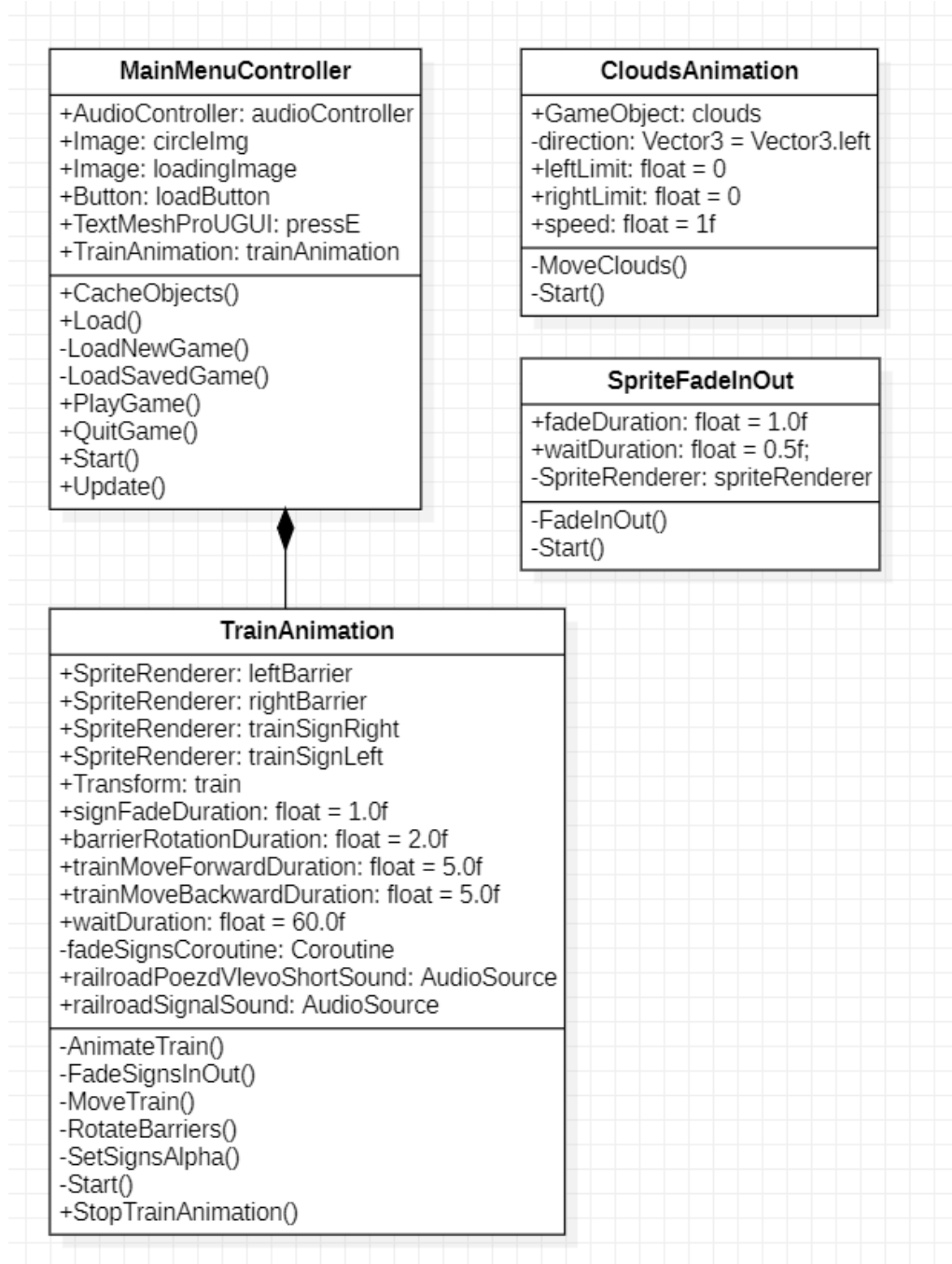


Рисунок 2.9 – Діаграма класів головного меню.

Одним з жанрів проекту виступає жанр Visual Novel. Головними складовими цього жанру є спрайти персонажів, фони локацій, відображення речень і аудіосупровід. Для досягнення цих ефектів використовуються два основних класи: StoryScene і ChooseScene.

Клас StoryScene відповідає за представлення ігрової сцени візуальної новели. У ньому зберігаються список речень (sentences), спрайт фону (background) та наступна сцена (nextScene). Кожне речення має свою музичну,

звукову та оточуючу аудіо доріжку, текст речення, спікера і список дій. Кожна дія включає спікера, індекс спрайту, тип дії (поява, рух, зникнення), координати та швидкість руху. Клас StoryScene успадковує базовий клас GameScene, який визначає загальні атрибути для ігрових сцен.

Клас ChooseScene відповідає за сценарій вибору візуальної новели. Він містить список варіантів вибору (labels), кожен з яких має текст вибору та наступну сцену, яка буде відтворена після вибору. Наступна сцена може бути представлена як екземпляр класу StoryScene або як назва наступної сцени.

Центральним елементом є клас GameScene, який слугує базовим класом для двох інших класів: StoryScene та ChooseScene, які, в свою чергу, наслідуються від GameScene. Це демонструє використання наслідування, де класи StoryScene та ChooseScene наслідують властивості та методи від базового класу GameScene.

GameScene використовується для визначення загальних атрибутів і функціональності, які спільні для всіх типів ігрових сцен у проекті. Цей клас може містити спільні поля, методи або властивості, які будуть спільно використовуватись у підкласах, наприклад StoryScene і ChooseScene. Використання GameScene як базового класу дозволяє уникнути дублювання коду та спростити управління ігровими сценами. Крім того, використання ScriptableObject дозволяє зберігати екземпляри GameScene як ресурси у проекті, що полегшує редагування та налаштування сцен в редакторі Unity.

На рисунку 2.10 зображено діаграму класів, яка репрезентує структуру та взаємодію між класами.

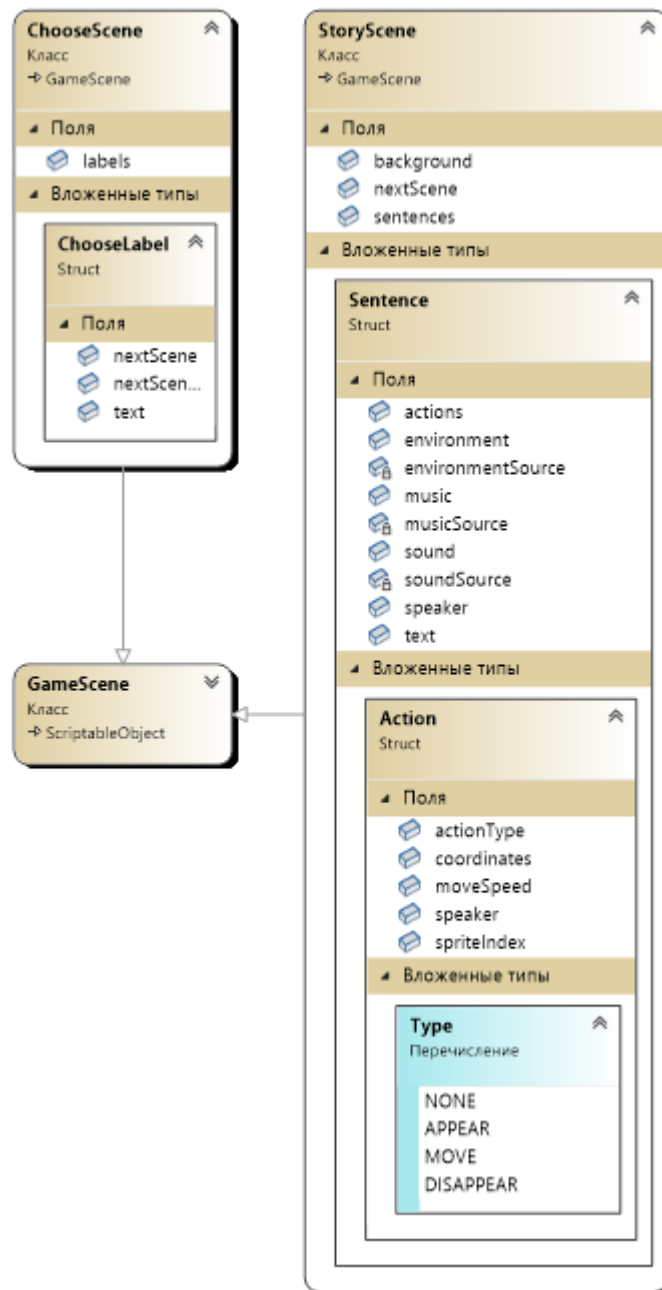


Рисунок 2.10 – Діаграма класів для створення локацій Scene-1

Головний скрипт для Scene-1 носить назву "GameController". Цей скрипт управляє головною логікою гри, інтерфейсом користувача, керує сценами та управляє аудіо.

З початку гри, скрипт зберігає поточну сцену у змінній currentScene. У скрипті GameController використовуються такі компоненти як bottomBar, backgroundController, chooseController та audioController, які відповідно контролюють панель інтерфейсу користувача, зміну фону, контролер вибору та аудіо.

В методі Start() викликається метод StartLevel(), що використовується для запуску переходу на сцену та видалення невикористовуваних ресурсів для оптимізації пам'яті. Також тут відбувається перевірка, чи збережена гра, і якщо так, то відновлює стан гри.

Метод Update() використовується для перевірки вводу користувача і виклику відповідних методів для переходу до наступних сцен або зміни інтерфейсу.

Метод PlayScene() використовується для переходу до нової сцени, і він використовує корутину SwitchScene() для гладкого переходу. Метод PlayAudio() використовується для відтворення аудіо для конкретної сцени.

Метод "SaveGame()" використовується для збереження стану гри, що включає індекс поточного речення та історію пройдених сцен. Значення цих полів зберігаються у форматі, який підтримує клас SaveData, а потім передається у SaveManager, який забезпечує збереження цих даних. На рисунку 2.11 зображений метод SaveGame().

Цей скрипт управляє взаємодією гравця з сценою і обробляє переходи між сценами, вибір сцен та відтворення аудіо, які є ключовими аспектами геймплею у цій грі.

На рисунку 2.12 продемонстровані всі інші класи для Scene-1

```
0 references
public void SaveGame()
{
    Debug.Log("Збереження гри...");
    basePauseMenu.PlaySaveAnimation();
    List<int> historyIndices = new List<int>(); // Створюємо список для зберігання індексів сцен історії
    history.ForEach(scene =>
    {
        historyIndices.Add(this.data.scenes.IndexOf(scene)); // Додаємо індекси сцен історії до списку індексів
    });
    SaveData data = new SaveData
    {
        sentence = bottomBar.GetSentenceIndex(), // Отримуємо індекс поточного речення знизу екрану
        prevScenes = historyIndices // Передаємо список індексів сцен історії в об'єкт збереження
    };
    SaveManager.SaveGame(data); // Зберігаємо гру за допомогою менеджера збережень
}
```

Рисунок 2.11 – Метод SaveGame()

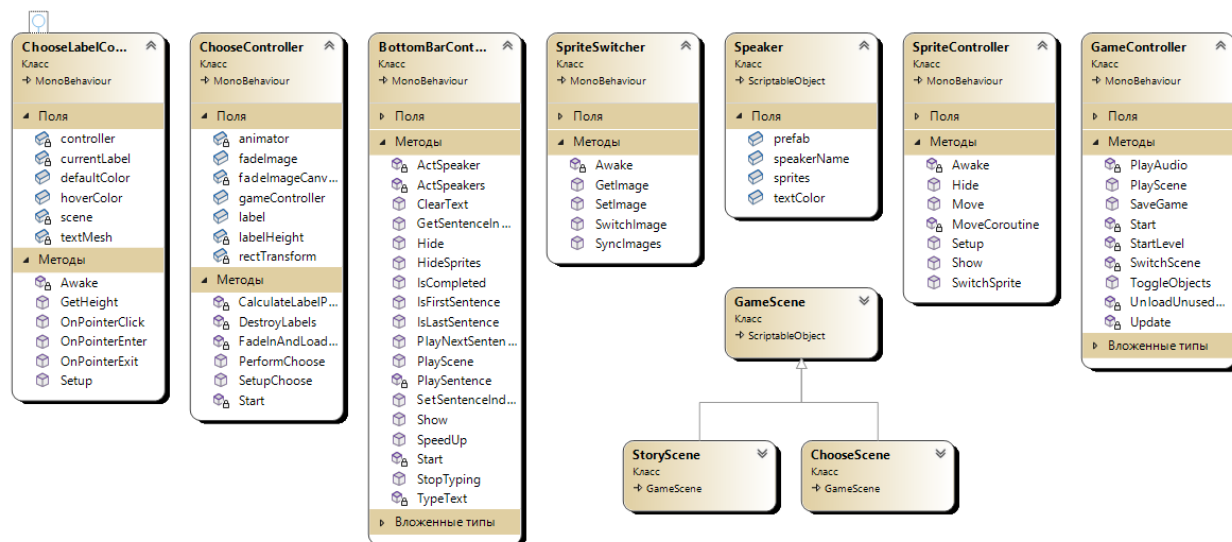


Рисунок 2.12 – Діаграма класів Scene-1

Для Scene-2 LevelManager є важливим компонентом в ігровій сцені. Його головною відповідальністю є керування різними аспектами рівнів гри, включаючи завантаження і збереження прогресу гравця, а також контроль аудіо та візуальних ефектів.

Змінні: Об'єкти та параметри, які керують різними аспектами рівня, такі як продовження анімації затемнення (fadeDuration), поточну сцену (currentScene), музику рівня (levelMusic), меню паузи (pauseMenu) та багато інших.

Start(): Ініціалізує різні аспекти рівня при початку. Це включає отримання посилань на об'єкти гри (наприклад, Player, HintManager), запуск анімації (PlayAnimationAndDeactivateCanvas), а також завантаження прогресу гри, якщо гра була збережена.

LoadLevelProgress() і SavePlayerProgress(): Ці методи відповідають за завантаження і збереження прогресу гравця відповідно.

GetLevelMusicPath(): Цей метод повертає шлях до музичного файлу, який повинен відтворюватися на цьому рівні.

FadeInLevel() і PlayAnimationAndDeactivateCanvas(): Ці корутини відповідають за анімаційні ефекти на початку рівня. FadeInLevel() поступово зменшує прозорість об'єкта StartImage, створюючи ефект затемнення, в той час

як `PlayAnimationAndDeactivateCanvas()` відключає об'єкт `StartImage`, коли анімація завершена.

Цей клас є центром управління рівнем, що координує дії між різними об'єктами і системами.

Для реалізації квестів було створено клас `QuestSystem`. Клас `QuestSystem` є основою системи квестів в грі. Цей клас відповідає за керування всіма квестами гравця та відображенням інформації про них. Він зберігає всі квести в `questList` та відображає інформацію про активний квест на ігровому інтерфейсі. Клас також керує активацією та завершенням квестів, використовуючи методи `StartQuest` та `CompleteQuest`, відповідно. Він має систему підказок, яка відображає інформацію, коли гравець натискає відповідну кнопку, та обробляє анімації затемнення для панелі квестів.

Якщо у гравця немає активних квестів, система квестів веде себе інакше.

По-перше, коли гравець намагається відкрити панель квестів (наприклад, натискаючи `Tab`), замість показу панелі квестів запускається корутина `ShowAndHideQuestEndedPanel()`, яка відображає спеціальну панель, що показує, що всі квести завершено.

По-друге, якщо гравець спробує активувати або завершити квест за допомогою методів `StartQuest()` або `CompleteQuest()`, ці методи не виконуються, тому що вони мають умови, які перевіряють, чи є квест активним і чи не завершено його вже.

Таким чином, система квестів коректно обробляє ситуації, коли квести відсутні.

Метод `HandleQuestComplete()` викликається, коли квест виконується. Він змінює стан виконаного квесту на неактивний, відключає тригер квесту (якщо він існує) та активує наступний квест в списку `questList` (якщо він існує).

Методи `HandleQuestComplete`, `StartQuest` та `CompleteQuest` наведені на рисунку 2.13

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

```

3 references
public void HandleQuestComplete(Quest quest)
{
    Debug.Log($"Квест {quest.Name} виконано!");
    quest.IsActive = false;
    if (quest.TriggerObject != null)
        quest.TriggerObject.enabled = false;

    // Активувати наступний квест, якщо він існує
    int currentIndex = questList.IndexOf(quest);
    if (currentIndex + 1 < questList.Count)
    {
        questList[currentIndex + 1].IsActive = true;
        if (questList[currentIndex + 1].TriggerObject != null)
            questList[currentIndex + 1].TriggerObject.enabled = true;
    }
}

7 references
public void StartQuest(Quest quest)
{
    quest.IsActive = true;
    quest.TriggerObject.enabled = true;
    StartCoroutine(FadeIn());
}

7 references
public void CompleteQuest(Quest quest)
{
    if (quest.IsActive && !quest.IsComplete)
    {
        quest.IsComplete = true;
        quest.TriggerObject.enabled = false;
        HandleQuestComplete(quest);
    }
}

```

Рисунок 2.13 – Методи StartQuest, CompleteQuest та HandleQuestComplete.

DialogReader є класом, що відповідає за керування та відображення діалогів у грі.

Клас містить ряд важливих полів, таких як dialogData, яке зберігає дані діалогу, speakerColors, що містить колір тексту для кожного динамічного спікера, dialogJson та dialogJsonAfterBossDies, що представляють собою JSON файли діалогів.

Клас також містить два індекси, currentDialogIndex та currentSentenceIndex, що слідкують за поточним станом діалогу та речення відповідно.

У класі визначено методи для відображення діалогу (DisplayDialog), завантаження нового діалогу (LoadDialog), визначення того, чи курсор

знаходиться над елементом UI (IsPointerOverUIElement), а також методи для отримання та встановлення поточного індексу діалогу і речення (GetCurrentDialogIndex, GetCurrentSentenceIndex, SetCurrentDialogIndex, SetCurrentSentenceIndex).

Важливий аспект - це подія OnDialogComplete, яка викликається, коли весь діалог пройдено. Клас також забезпечує збереження індексів діалогу і речення, що дозволяє продовжувати діалог з місця, де гравець зупинився.

На рисунку 2.14 зображені методи для відновлення поточного стану діалогу.

```
// Додаємо відкритий метод для отримання поточного індексу діалогу
2 references
public int GetCurrentDialogIndex()
{
    return currentDialogIndex;
}

2 references
public int GetCurrentSentenceIndex()
{
    return currentSentenceIndex;
}

8 references
public void SetCurrentDialogIndex(int index)
{
    currentDialogIndex = index;
}

8 references
public void SetCurrentSentenceIndex(int index)
{
    currentSentenceIndex = index;
}
```

Рисунок 2.14 – Методи для контролю діалогу

Однією з головних механік - це рух персонажа. Скрипт CharacterController2D є серцем усіх дій, пов'язаних з двомірним персонажем в Unity. Він відповідає за керування рухом персонажа, включаючи горизонтальний рух, присідання, стрибки, і повороти. Важливим методом, що виконує ці функції, є Move(), який використовує параметри для виконання відповідних дій. Особливості цього скрипту включають можливість керувати

персонажем під час знаходження в повітрі, а також застосування м'якого згладжування руху для природної анімації руху персонажа.

Скрипт Player відображає взаємодію гравця з грою. Він управляє рухом гравця, включаючи присідання, стрибки, а також керує здоров'ям гравця. Основні методи включають RegenHealth(), який постійно перевіряє стан здоров'я гравця і відновлює його за потреби, та TakeDamage(int damage), який регулює зменшення здоров'я при отриманні шкоди. Окрім того, цей скрипт також включає ряд анімацій, що динамічно реагують на взаємодії гравця з грою.

Все взаємодіє таким чином, що Player використовує методи, визначені в CharacterController2D, для реалізації своїх власних функцій. Таким чином, обидва скрипти доповнюють один одного.

У грі також було реалізовано механіку зміни персонажів. Механіка зміни персонажа в цій грі реалізована за допомогою двох основних класів - CharacterSelectionPanelController та PlayerSwitchController.

В класі CharacterSelectionPanelController на початку гри ініціалізуються об'єкт: playerSwitchController. Об'єкт відповідає за виклик зміни персонажа.

У методі Update() класу CharacterSelectionPanelController виконується перевірка натиснення клавіші LeftAlt. Якщо клавіша натиснута протягом декількох секунд (зазначено у altHoldDuration), то активується панель вибору персонажів (FadeIn()). Натискання на зображеннях персонажів (лисиці та солдата) активує відповідний метод зміни персонажа в класі PlayerSwitchController (SwitchToFox() або SwitchToSoldaten()).

У класі PlayerSwitchController реалізована логіка зміни персонажа. При виклику SwitchToFox() або SwitchToSoldaten(), активний в даний момент персонаж стає неактивним, а новий персонаж стає активним. Здоров'я нового персонажа стає рівним здоров'ю попереднього персонажа.

Метод SyncCharacterTransforms() гарантує, що новий персонаж з'явиться на місці попереднього.

Механіки персонажів також різні. Наприклад, механіка стрільби для персонажа солдата реалізована за допомогою класу RayCastWeapon. В даному

					КН.КР.Б – 122 – 23 – ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		39

кодi використовуються об'єктні пули для керування інстанціюванням та поверненням об'єктів, таких як Bullet і LaserBullet.

Механіка стрільби для персонажа солдата реалізована за допомогою класу RayCastWeapon. В даному кодi використовуються об'єктні пули для керування інстанціюванням та поверненням об'єктів, таких як куля і лазерна куля.

Код класу RayCastWeapon містить метод Update(), в якому перевіряється натискання кнопки вогню (Fire1) і час вогню. Якщо умови виконуються, то викликаються методи ShootLaserBullet() або GrenadeBullet(), в залежності від типу амуніції.

Методи ShootLaserBullet() та GrenadeBullet() створюють об'єкти з відповідних пулів. Наприклад, в методі ShootLaserBullet() створюється об'єкт лазерної кулі з пулу laserBulletPool, і йому присвоюються потрібні параметри (швидкість, пошкодження). Після цього об'єкт додається до сцени, а через Destroy() встановлюється час життя об'єкта.

У методі Awake() створюються пули об'єктів для кожного типу об'єкта: bulletPool, laserBulletPool та impactEffectPool. Кожен пул ініціалізується з вказанням функції, яка створює новий об'єкт за потреби, функції для знищення об'єкта та функції, яка буде викликатися при знищенні пулу.

Метод DelayedRelease() викликається після закінчення життя об'єкта (наприклад, кулі). Він приймає об'єкт і затримку, після чого у відповідності до тегу об'єкту відбувається повернення його до відповідного пулу (bulletPool, laserBulletPool, impactEffectPool). Зауважте, що повернення об'єкта до пулу дозволяє йому бути повторно використаним, замість створення нового об'єкта, що поліпшує продуктивність гри та уникне надмірного використання пам'яті.

Використання пулів дозволяє оптимізувати роботу з об'єктами, зменшуючи витрати пам'яті і продуктивність, оскільки не потрібно постійно створювати і знищувати об'єкти, а можна використовувати їх у повторному режимі.

Класи LaserBullet і Bullet відповідають за зіткнення куль з об'єктами. У методі OnTriggerEnter2D(Collider2D hitInfo) перевіряється тип об'єкта, з яким

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

стикається куля, і відповідно до цього здійснюється зменшення здоров'я боса, створення ефекту удару та знищення кулі.

Цей код використовує об'єктні пулі для ефективного управління пам'яттю і уникнення частого створення та знищення об'єктів. Використання пулів дозволяє повторно використовувати об'єкти, замість створення нових кожного разу, що поліпшує продуктивність гри.

Методи для створення об'єктів з пулу LaserBullet і Bullet наведені на рисунку 2.15.

```
1 reference
void ShootLaserBullet(Vector2 shootingDirection)
{
    // Створюємо об'єкт лазерної кулі з пулу
    GameObject laserBullet = Instantiate(laserBulletPrefab, firePoint.position, Quaternion.Euler(0, 0, Mathf.Atan2(shootingDirection.y, shootingDirection.x) * Mathf.Rad2Deg));
    LaserBullet laserBulletScript = laserBullet.GetComponent<LaserBullet>();
    laserBulletScript.damage = laserBulletDamage;
    laserBulletScript.speed = laserBulletSpeed;

    // Перевіряємо, чи об'єкт не є null перед доступом до нього
    if (laserBullet != null)
    {
        // Виконуємо потрібні дії з об'єктом
        Destroy(laserBullet, laserBulletLifeTime);
    }
}

1 reference
void GrenadeBullet(Vector2 shootingDirection)
{
    // Створюємо об'єкт лазерної кулі з пулу
    GameObject grenadeBullet = Instantiate(bulletPrefab, firePoint.position, Quaternion.Euler(0, 0, Mathf.Atan2(shootingDirection.y, shootingDirection.x) * Mathf.Rad2Deg));
    Bullet grenadeBulletScript = grenadeBullet.GetComponent<Bullet>();
    grenadeBulletScript.damage = bulletDamage;
    grenadeBulletScript.speed = grenadeBulletSpeed;

    // Перевіряємо, чи об'єкт не є null перед доступом до нього
    if (grenadeBullet != null)
    {
        // Виконуємо потрібні дії з об'єктом
        grenadeBullet.SetActive(true);
        StartCoroutine(DelayedRelease(grenadeBullet, bulletPrefabLifeTime));
    }
}
```

Рисунок 2.15 – Створення об'єктів з пулу

Scene-2 містить різноманітні скрипти, які сприяють доповненню ігрового досвіду та забезпечують різноманітний функціонал. Серед цих скриптів є WizzardController, HintManager та InventoryManager, які грають важливу роль у грі, надаючи гравцеві нові можливості та покращуючи геймплей.

WizzardController відповідає за управління та логіку руху NPC Чаклуна, діалогові системи, виконання завдань та надання вказівок гравцеві.

HintManager відповідає за надання підказок та порад гравцеві під час гри. Цей скрипт аналізує поведінку гравця, його дії та ситуацію в грі, і надає інформацію, що допомагає гравцеві просунутися далі.

InventoryManager відповідає за керування інвентарем гравця. Цей скрипт дозволяє гравцеві збирати, управляти та використовувати предмети, знаряддя та ресурси, які він знаходить у грі. Класи використані для Scene-2 зображені на рисунку 2.16.



Рисунок 2.16 – Діаграма класів для Scene-2

Далі на рисунку 2.17 наведена діаграма класів для Scene-3.

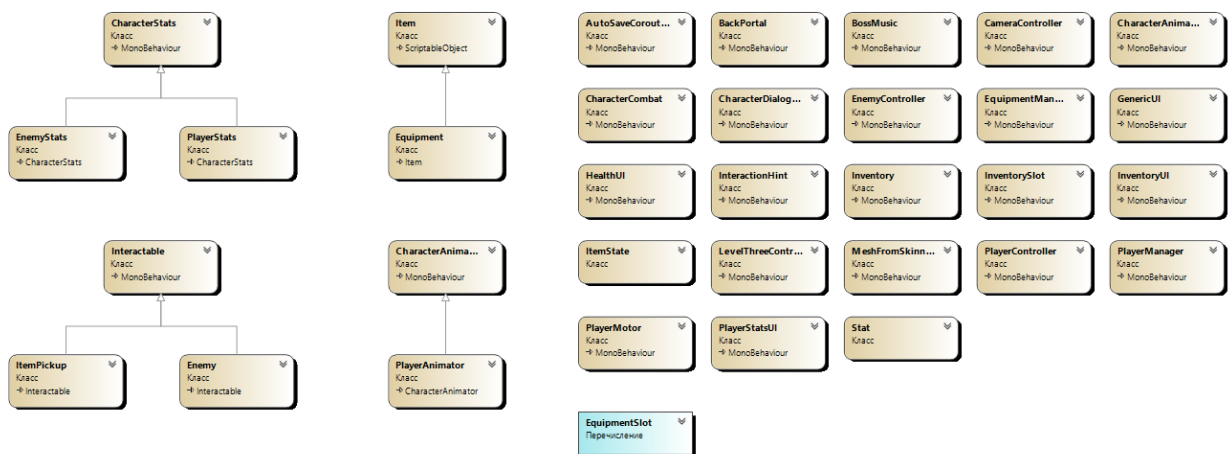


Рисунок 2.17– Діаграма класів для Scene-3

На діаграмі класів можна побачити три класи: CharacterStats, PlayerStats та EnemyStats. Клас CharacterStats виступає базовим класом, від якого успадковуються два інших класи, що репрезентують типи персонажів у грі: гравця (PlayerStats) та ворога (EnemyStats).

CharacterStats визначає загальні характеристики, які можуть мати всі персонажі гри, включаючи максимальне здоров'я, поточне здоров'я, час затримки, рівень пошкодження та броню. Він також містить методи для виконання загальних дій, таких як отримання пошкоджень та вмирання.

Клас PlayerStats розширює CharacterStats і має додаткову взаємодію з EquipmentManager і Equipment для управління модифікаторами броні та пошкоджень. Він також перевизначає метод Die() для виконання специфічних для гравця дій у випадку його смерті.

Клас EnemyStats також розширює CharacterStats і додає додаткові властивості та методи, які використовуються для визначення поведінки ворогів у грі. Він включає механізми, які взаємодіють з системою квестів та контролером рівнів, що дозволяє встановити специфічні реакції на смерть ворога(рис.2.18).

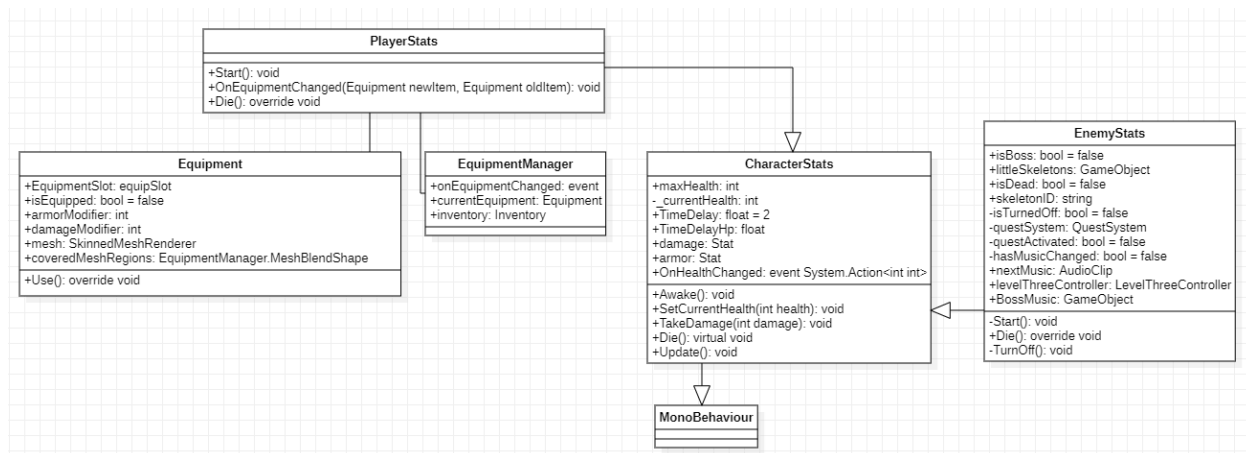


Рисунок 2.18 – Взаємодія класів для характеристики пресонажів

Діаграма класів, що зображена на рисунку 2.19 відображає класи та їхні взаємозв'язки в системі предметів та інвентаря в грі. Ці класи представляють різні аспекти інвентаря (Inventory), предметів (Item, Equipment), взаємодії з предметами (Interactable, ItemPickup) та пов'язаних з ними систем (HintManager, QuestSystem). Класи наслідуються від MonoBehaviour або інших класів, що дозволяє їм використовувати усі функції та можливості Unity3D.

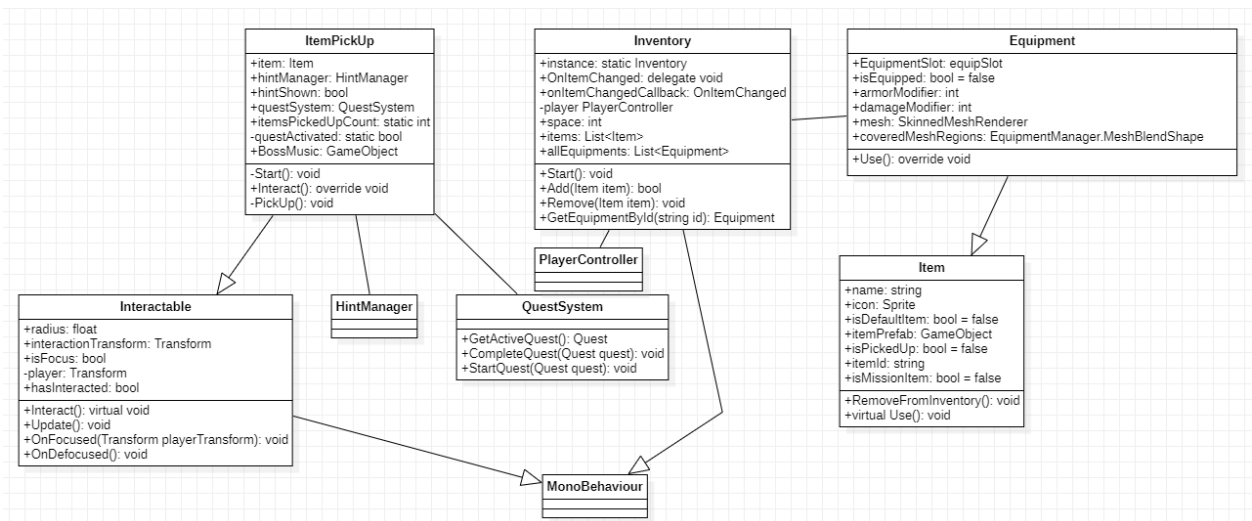


Рисунок 2.19 – Взаємозв'язки предметів та інвентаря

Клас Inventory відповідає за керування інвентарем гравця в грі. В процесі ініціалізації, в методі Awake(), перевіряється, чи існує вже екземпляр класу Inventory. Якщо так, то створювати новий екземпляр не потрібно. Якщо ні, то поточний екземпляр встановлюється як єдиний існуючий екземпляр, використовуючи патерн Singleton.

У методі Start() знаходиться компонент PlayerController гравця і зберігається посилання на нього.

Методи Add() і Remove() використовуються для додавання та видалення предметів з інвентаря. Коли предмет додається або видаляється, сповіщення про зміну інвентаря відправляється через делегат OnItemChanged, що активує колбек onItemChangedCallback.

Метод Add() включає в себе перевірку на наявність вільного місця в інвентарі перед додаванням предмета. Якщо місця немає, повідомлення про це виводиться в консоль, і предмет не додається.

Метод Remove() видаляє предмет з інвентаря, за умови, що предмет не є предметом для завдання.

Також є метод GetEquipmentById(), який повертає обладунок за ідентифікатором. Він проходить по всіх елементах списку allEquipments, шукаючи обладунок з відповідним itemId.

Клас ItemPickup розширює клас Interactable та керує логікою збирання предметів гравцем в грі.

Під час ініціалізації, яка відбувається в методі Start(), система знаходить в сцені компоненти HintManager та QuestSystem для подальшого їх використання. Цей метод також враховує попередній стан предметів, якщо такий існує, уникнувши повторного збирання вже піднятих предметів.

Метод Interact(), який визначає, що станеться при взаємодії гравця з предметом, виконує кілька ключових дій. Воно запускає процес збирання предмету, викликаючи метод Pickup().

Метод Pickup() відповідає за конкретну логіку збирання предмету. Цей процес передбачає додавання предмету до інвентаря гравця і, у разі успіху, видалення предмету з ігрового середовища. Додатково, цей метод збільшує лічильник піднятих предметів і показує відповідну підказку гравцю.

Отже, клас ItemPickup відіграє важливу роль у взаємодії гравця з ігровим середовищем, дозволяючи збирати предмети, що в свою чергу впливає на розвиток сюжету гри.

За управлінням персонажем відповідає клас PlayerMotor.

Цей клас використовує NavMeshAgent для навігації по сцені. У цьому класі визначено декілька важливих методів:

- MoveToPoint(): цей метод приймає точку в просторі і змушує NavMeshAgent переміщуватися до цієї точки.
- FollowTarget(): цей метод приймає ціль для переслідування. Це може бути будь-який об'єкт, що наслідує Interactable.
- StopFollowingTarget(): як впливає з назви, цей метод зупиняє переслідування цілі.
- SetPosition(): цей метод використовується для того, щоб помістити гравця на певну позицію на NavMesh.

Клас PlayerController відповідає за керування взаємодією гравця з об'єктами і оточенням. Він використовує PlayerMotor для руху гравця і обробляє взаємодії, такі як натискання кнопки миші. У цьому класі також визначені декілька важливих методів:

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

- SetFocus(): цей метод встановлює фокус на об'єкт Interactable, з яким гравець взаємодіє.
- RemoveFocus(): цей метод видаляє фокус з об'єкта Interactable.

На діаграмі класів PlayerMotor пов'язаний з PlayerController, що відображає те, що PlayerController керує PlayerMotor. Стрілка вказує на PlayerController, що підкреслює, що PlayerMotor контролюється PlayerController.

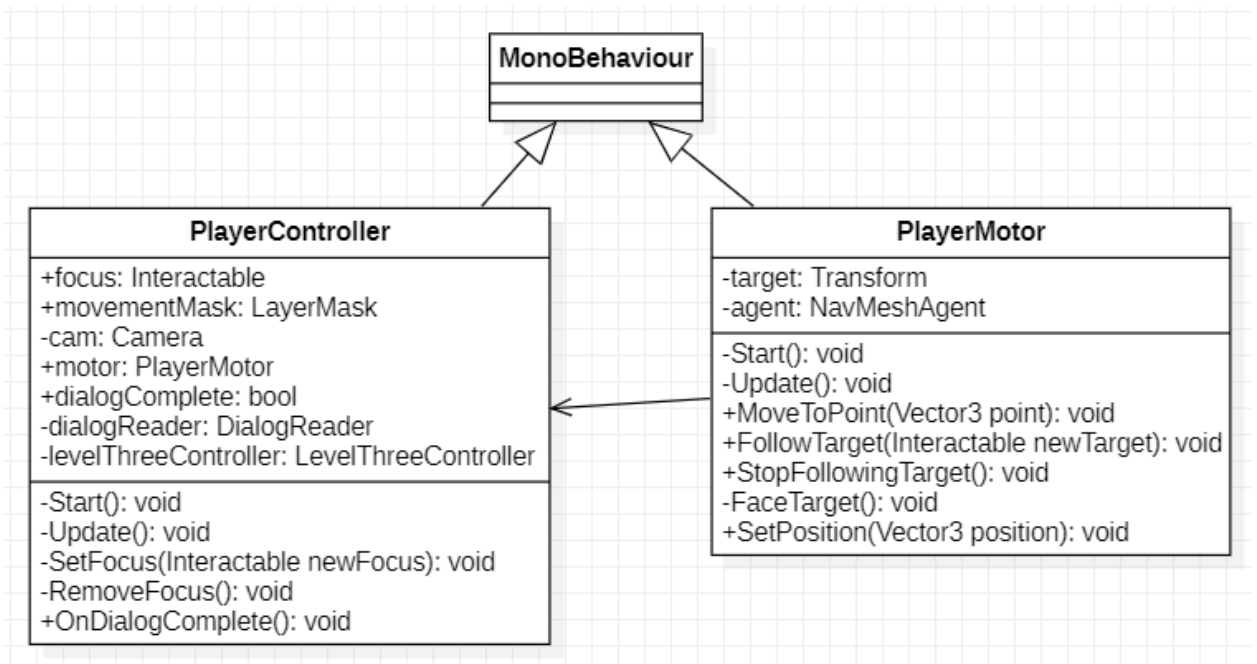


Рисунок 2.20 – Діаграма класів для управління персонажем

Одним з головних класів Scene-3 є клас LevelThreeController.

LevelThreeController є класом, що відповідає за управління третім рівнем гри. Він має різні властивості та компоненти, які виконують різноманітні функції. Наприклад, властивість currentScene відстежує поточну сцену гри. Компоненти, такі як pauseMenu, playerMotor та dialogReader, дозволяють керувати паузою гри, рухом гравця та відтворенням діалогів відповідно.

LevelThreeController також має компоненти, пов'язані з аудіо, такі як levelMusic та audioController, які відтворюють музику на рівні.

Крім того, клас включає компоненти, які відповідають за управління інтерфейсом користувача. Наприклад, inventoryUI оновлює інформацію про інвентар гравця, а hintManager показує підказки гравцю відповідно до поточного стану гри.

LevelThreeController також виконує функції збереження та завантаження прогресу гри на Scene-3. Він зберігає координати гравця, стан діалогів, стан квестів, предмети у інвентарі та інше. Це дозволяє гравцеві продовжити гру з того місця, де він зупинився.

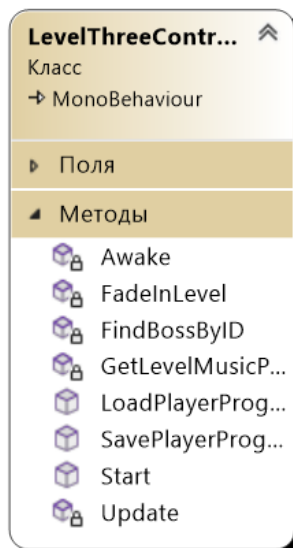


Рисунок 2.21 – LevelThreeController

Для розробки меню паузи було використано паттерн «Шаблонний метод».

Шаблонний метод — це поведінковий патерн проектування, що визначає каркас алгоритму в базовому класі та дозволяє підкласам перевизначати деякі кроки алгоритму, не змінюючи його загальну структуру. В випускній кваліфікаційній роботі клас BasePauseMenu виступає як базовий клас, що визначає загальний алгоритм для керування паузою в грі. Це абстрактний базовий клас, який визначає загальні елементи і поведінку для меню паузи. Він містить методи, які дозволяють продовжити гру (Resume), поставити її на паузу (Pause), включити і виключити відображення курсора (ToggleCursor) і повернутися до головного меню (ToMainMenu). Також клас має метод QuitGame для виходу з гри, а також метод PlaySaveAnimation, що запускає анімацію збереження.

Клас PauseMenuController наслідується від BasePauseMenu і керує поведінкою меню паузи з урахуванням взаємодії з користувачем. Він перевизначає методи Resume і Pause, додаючи анімацію затухання до звукових

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

джерел. Також він має метод IgnoreMouseClicked, що дозволяє ігнорувати натискання миші.

Класс PauseMenu також наслідується від BasePauseMenu і містить специфічну для рівня логіку збереження стану гри. Він перевизначає методи Resume і Pause, додаючи анімацію затухання до звукових джерел. Він має методи SavePlayerProgress і SavePlayerProgressLevelThree, які зберігають прогрес гравця на різних рівнях.

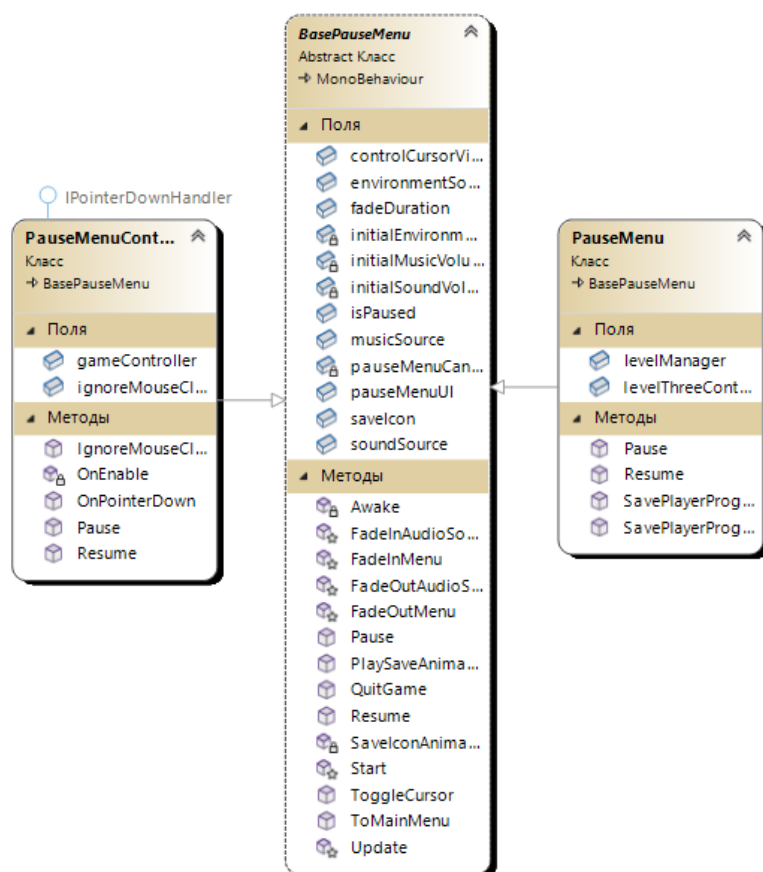


Рисунок 2.22 – Діаграма класів для меню паузи.

Для реалізації збереження і завантаження прогресу гри було розроблено класс SaveManager.

Клас SaveManager відповідає за збереження та завантаження гри. Він містить статичні методи SaveGame(), LoadGame(), IsGameSaved() та ClearSavedGame(), які дозволяють зберігати гру, завантажувати її, перевіряти наявність збереженої гри та видаляти її відповідно. Метод SaveGame() перетворює дані гри у формат JSON та зберігає їх у файлі. Метод LoadGame()

завантажує збережені дані з файлу та повертає їх у вигляді об'єкту SaveData. Метод IsGameSaved() перевіряє наявність файлу збереження, а метод ClearSavedGame() видаляє цей файл, якщо він існує.

Структура SaveData використовується для зберігання даних гри. Вона містить різні поля для зберігання позицій об'єктів, номерів сцен, індексів діалогів, станів квестів та іншої інформації, яка необхідна для збереження прогресу гравця.

Клас DataHolder є нащадком ScriptableObject і використовується для зберігання списку сцен гри до Scene-1 жанру візуальної новели. Цей клас дозволяє зручно організувати та зберігати дані про різні сцени гри, що може бути корисним для подальшого використання та редагування.

Ці класи та структури використовуються для ефективного збереження та завантаження даних гри, а також для забезпечення можливості збереження та відновлення прогресу гравця.

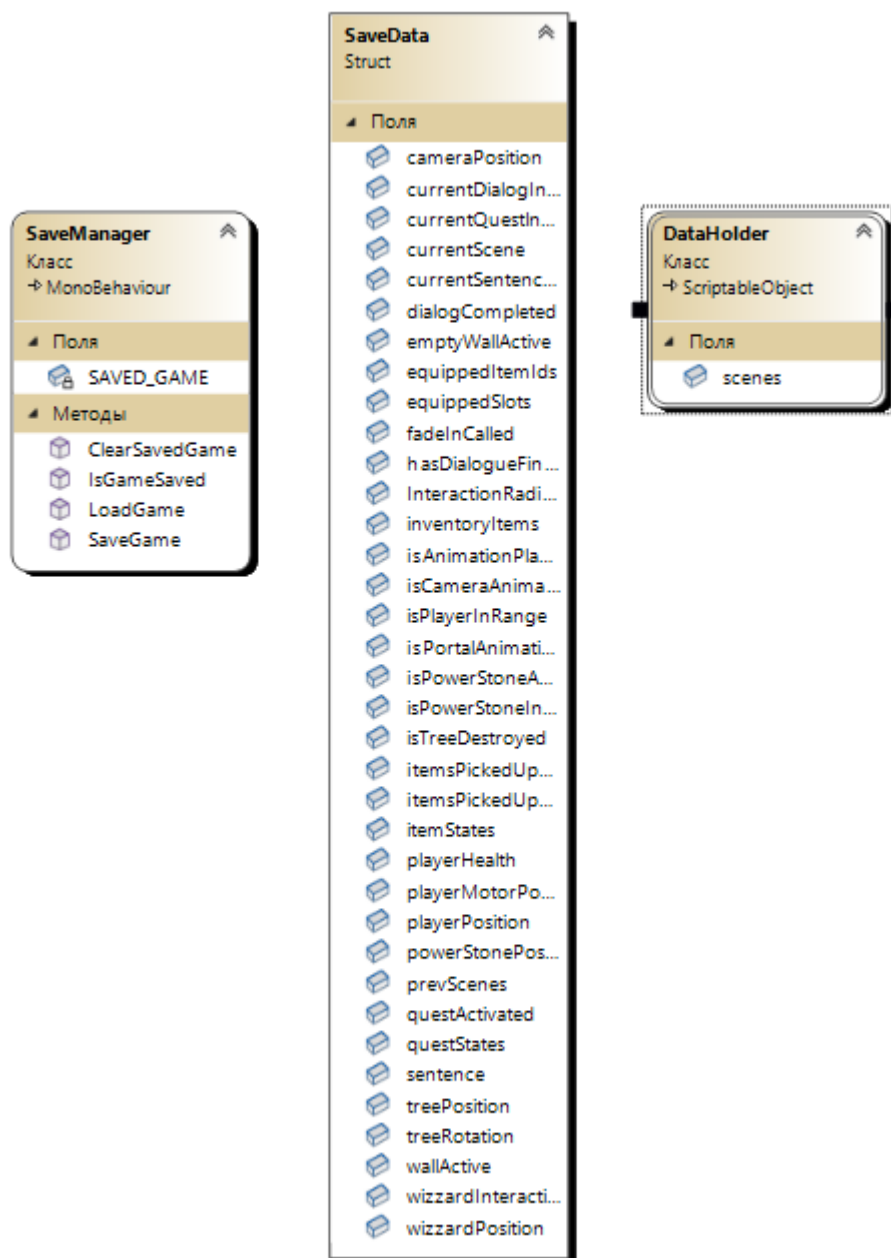


Рисунок 2.23 – Діаграма класів для збереження гри

Scene-4 та Scene-5 використовують ті ж самі класи та методи, які вже були представлені раніше в Scene-1 та Scene-2. Це означає, що алгоритм роботи цих сцен ідентичний, і вони використовують ті ж самі функції.

Висновки до другого розділу

Розділ 2 присвячений аналізу вимог до програмного продукту, графічному представленню основних алгоритмів гри та розробці ігрового додатку разом з основними методами.

Було проведено аналіз вимог до програмного продукту. Цей етап дозволив з'ясувати потреби та очікування користувачів, а також визначити

функціональні та нефункціональні вимоги до гри. Результатом аналізу вимог є чітко сформульовані вимоги, які будуть використовуватися в процесі подальшої розробки. Також було розглянуто графічне представлення основних алгоритмів гри. Це включає в себе створення візуальних схем та діаграм роботи алгоритмів гри. Графічне представлення допомагає краще зрозуміти логіку гри та спрощує процес розробки.

І нарешті була проведена розробка ігрового додатку та реалізація методів та демонстрація основних методів гри. Цей етап включав створення необхідних класів, скриптів та функцій для реалізації функціональності гри. Основні методи, які були розроблені, включають управління паузою, збереження та завантаження гри, обробку взаємодії з об'єктами гри та інші.

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

РОЗДІЛ 3. ОПИС РОБОТИ З ПРОГРАМНИМ ДОДАТКОМ ТА ЙОГО ТЕСТУВАННЯ

3.1 Опис роботи з програмним додатком

Ігровий додаток «Player One, Get Ready!» було створено на рушії Unity 3D. Після запуску додатку користувачі відразу бачать головне меню. Головне меню містить такі пункти, як «Нова Гра», «Продовжити», «Налаштування» або «Вихід»(Рис.3.1).

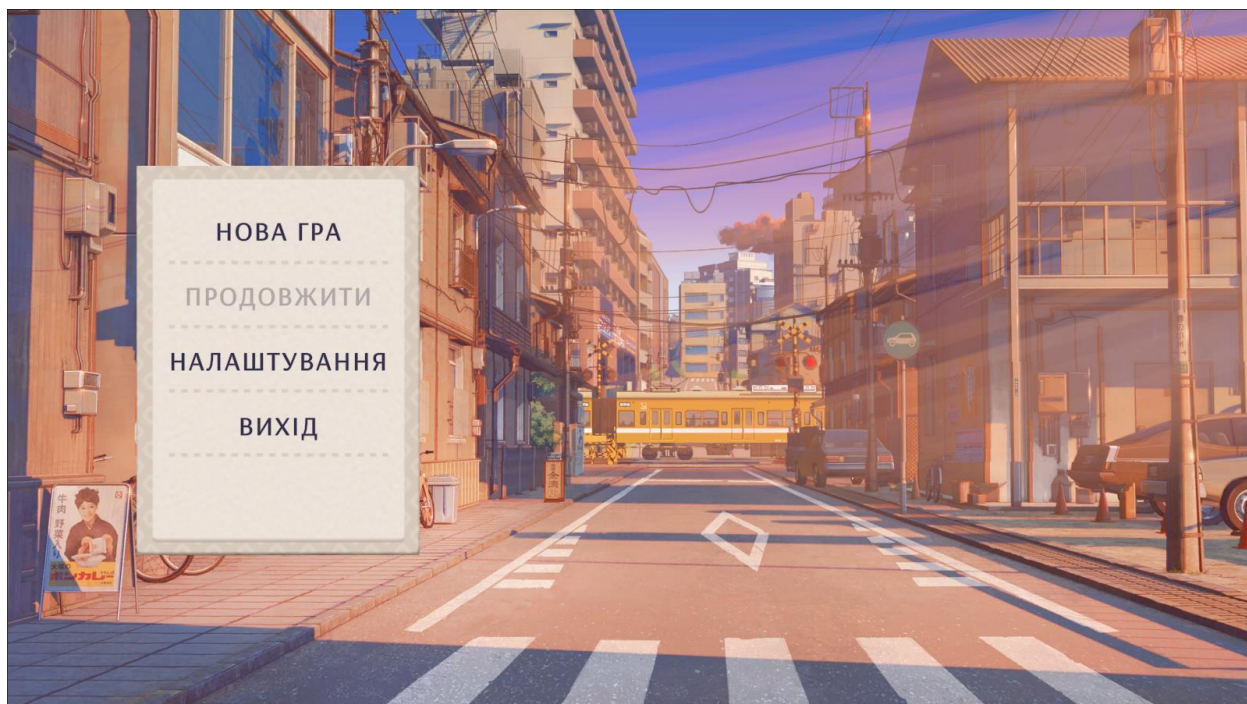


Рисунок 3.1 – Головне меню гри

При натисканні кнопки «Налаштування» користувач переходить до вікна налаштувань, де може налаштувати параметри звуку або переглянути параметри управління(рис.3.2).

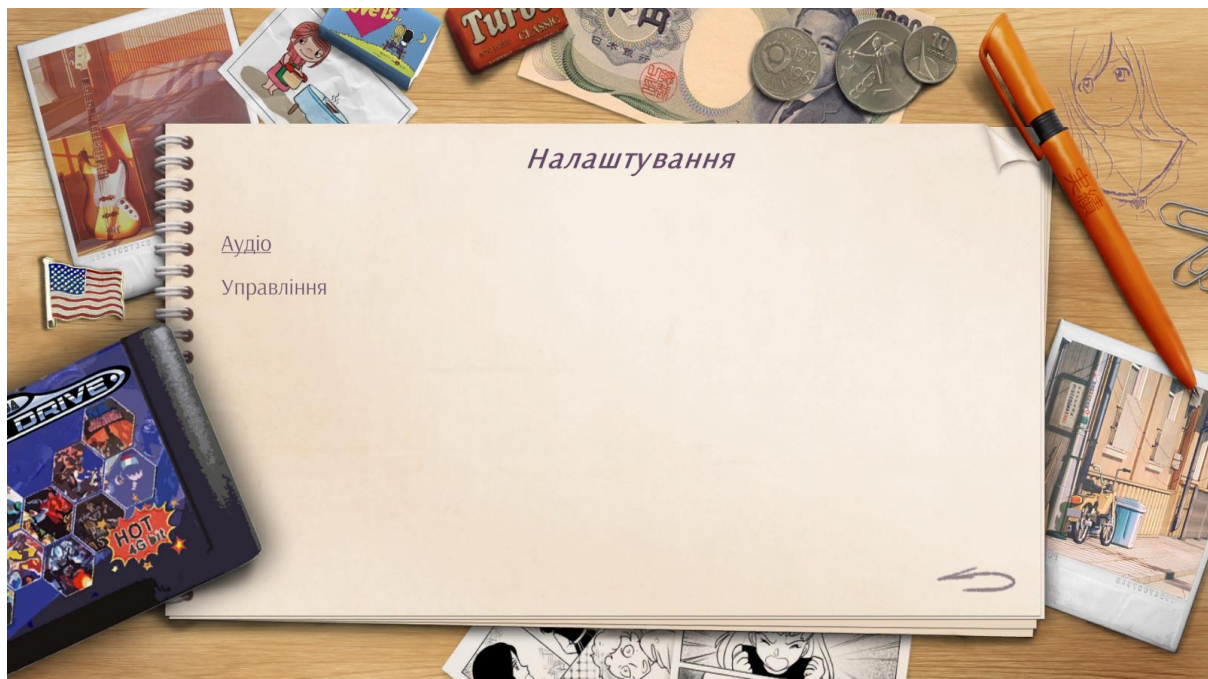


Рисунок 3.2 – Налаштування

Повноцінний проект являє собою гру побудований з трьох жанрів. На рисунку 3.3 відображено першу частину гри. Жанр гри – візуальна новела.

Візуальна новела - це жанр гри, який зосереджується на розповіді історії через текст та візуальні елементи(рис.3.3). Scene-1 налічує собою 28 локацій з фонами для локації, аудіосупровідом, спрайтами персонажів, реченнями для історії та 2 варіанта інтерактивності.



Рисунок 3.3 – Ігровий процес Scene-1

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

Рухатися по сюжету можна за допомогою натискання клавіші ЛКМ або пробілу. При натисканні ESC або UI елементів гра переходить в режим паузи (рис.3.4).

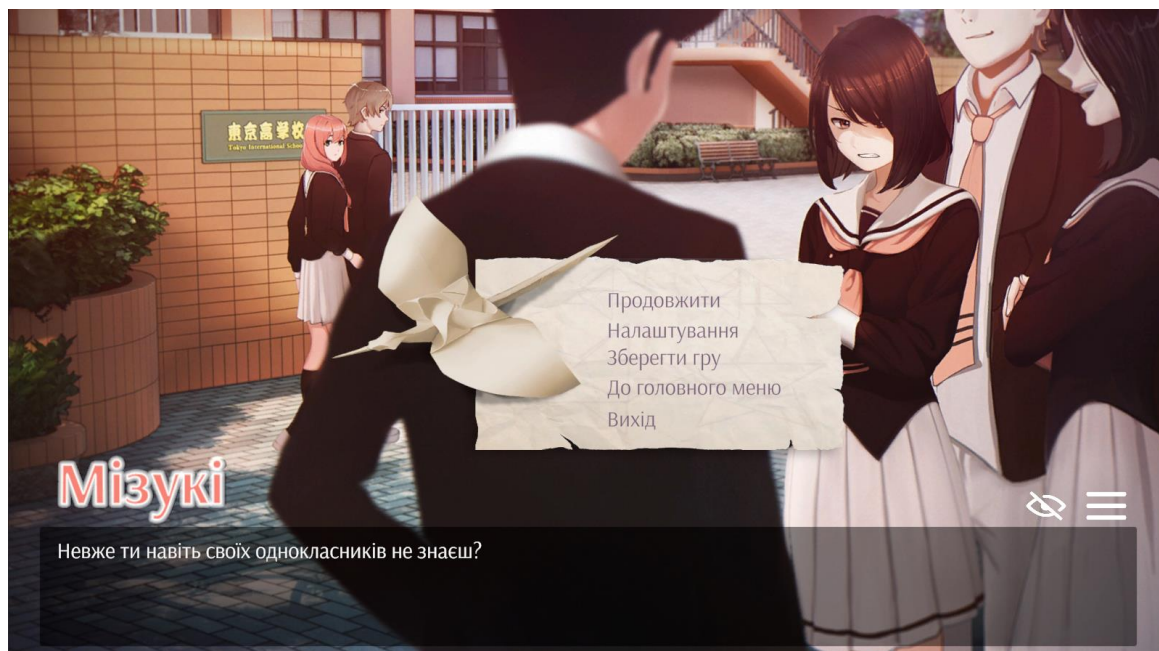


Рисунок 3.4 – Меню паузи

Якщо гравець натискає на UI елемент «приховати інтерфейс», то елементи гри BottomBar зникають і можна переглянути чистий фон(рис.3.5).



Рисунок 3.5 – Демонстрація при натисканні на UI елемент «приховати інтерфейс»

Було реалізовано інтерактивну частину. Гравець може обирати ту чи іншу дію, щоб краще відчувати себе на місці головного героя. Для вибору було

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

реалізовано UI панель з варіантами вибору. На рисунку 3.6 видно інтерактивну частину гри, а саме можливість обрати діалог, або дії головного персонажа.

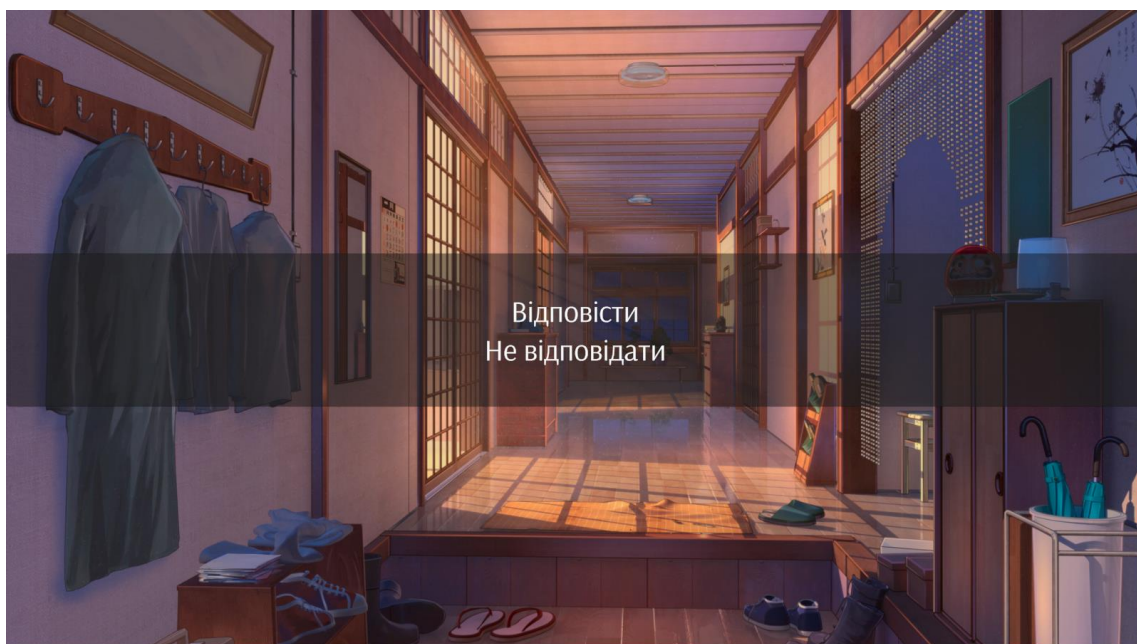


Рисунок 3.6 – Вибір дії або діалогу.

Друга частина гри, представляє собою 2D платформер з елементами RPG. В цій частині гравець отримує можливість виконувати квести, вести діалог з персонажами, міняти облік персонажа. Управління персонажа виконується за допомогою клавіш A/D/S/Space, де A – вліво, D – вправо, S – присісти, Space – прижок(рис.3.7).



Рисунок 3.7 – Початок гри

Інтерфейс має всі необхідні елементи. Він достатньо зрозумілий та простий(рис.3.8).

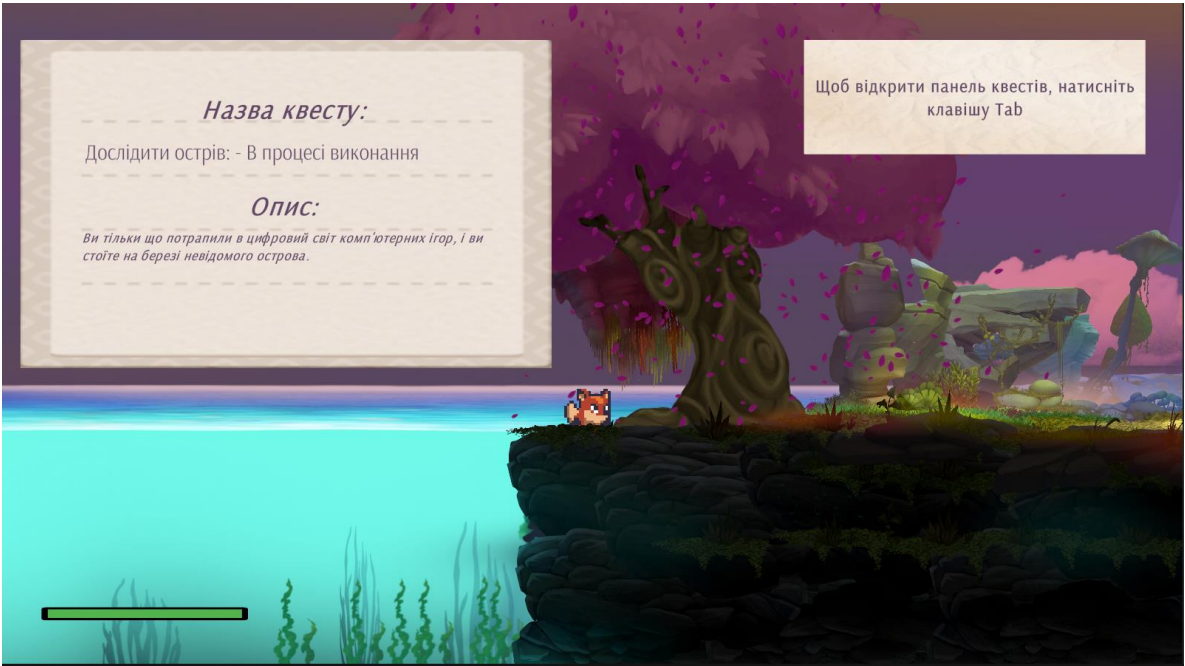


Рисунок 3.8 – Інтерфейс гри

Якщо обрати солдата, то гравець також може атакувати, постріл відбувається за допомогою клавіші ЛКМ, щоб змінити варіанти пострілу, можна натиснути клавішу «1» – лазерні кулі, «2» – гранатомет(рис.3.9).



Рисунок 3.9 – UI для зміни персонажа

Також було реалізовано систему монологу головного персонажа(рис3.10). Система монологу була реалізована, щоб надати характер герою. Монологи можуть допомогти гравцям краще розуміти внутрішній світ головного героя. Вони можуть використовуватись для розкриття його мотивацій, цілей, сумнівів, страхів або внутрішнього конфлікту. Це дозволяє створити більш глибокого і цікавого персонажа.

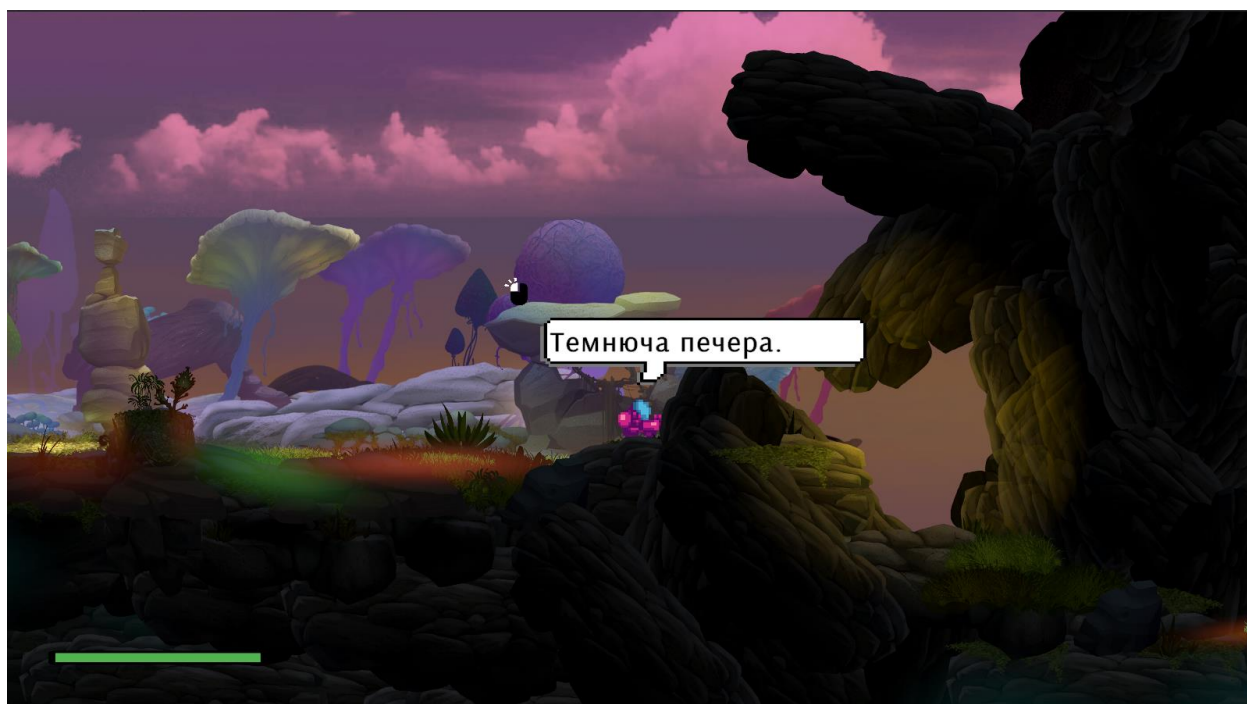


Рисунок 3.10 – Монолог головного персонажа

Квести змінюються після виконання минулого квесту в списку і замінюються наступним. Наприклад, якщо гравець підніме квестовий предмет до інвентаря, то попередній квест буде завершено, а новий розпочато(рис.3.11).



Рисунок 3.11 – Система інвентаря

Також у гру було додано боса. Битва з босом неминуча після завершення одного з квестів. Якщо бос помирає то активується наступний квест, якщо гравець помирає, то можна відновити гру з файлу збереження в головному меню(рис.3.12).



Рисунок 3.12 – Битва з босом

Після виконання всіх квестів на локації, гравцю необхідно знову ініціювати діалог з чаклуном для переходу до Scene-3(рис.3.13).

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



Рисунок 3.13 – Ініціація діалогу з чаклуном

Після діалогу з чаклуном гравцю буде запропоновано перейти до третього рівня(рис.3.14).



Рисунок 3.14 – Портал, який веде до Scene-3

Scene-3 представляє собою повноцінну 3D локацію з квестами, різними NPC, інтерактивними предметами(рис3.15).



Рисунок 3.15 – початок Scene-3

Інтерфейс рівня простий і зрозумілий. Також вбудована система квестів і підказок. Управляти персонажем можливо за допомогою клавіші ПКМ. Управління реалізовано як в МОБА іграх. Положення камери можна регулювати клавішами A/D(рис3.16).



Рисунок 3.16 – Інтерфейс рівня Scene-3

Також вбудована система діалогів з персонажами(рис.3.17). Діалоги потрібні для визначення цілі гравця на Scene-3.

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



Рисунок 3.17 – Діалоги з персонажами

З діалогу стає зрозуміло, що на локації розкидані предмети, які гравець може збирати за для проходження гри(рис3.18). Якщо гравець збере всі необхідні для виконання квесту предмети, то розпочнеться наступний із списку квест.



Рисунок 3.18 – Інвентарь з предметами

Гравець може надягати предмети, які знайшов. Предмети будуть змінювати характеристики гравця. Щоб підібрати предмет, потрібно

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

натиснути на нього ПКМ(рис.3.19). Підібрати предмет гравець може в радіусі 3f. Різні предмети змінюю різні параметри персонажу, наприклад щит збільшує параметр броні, а меч параметр шкоди по ворогам.



Рисунок 3.19 – Змінені характеристики

На локації присутні вороги, вороги можуть наносити шкоду гравцю, а гравець ворогам. Предмети, які гравець екіпірував допоможуть гравцю вносити більше шкоди ворогам, а також посилити значення armor. Щоб атакувати ворога, потрібно натискати на нього клавішею(рис3.20).



Рисунок 3.20 – Битва з ворогами

Після виконання половини квестів гравцеві відкривається битва з босом. Фінальний квест, щоб отримати предмет за для подальшого проходження гри (рис.3.21). Бос має більші характеристики шкоди і броні, тому для проходження гри, гравцеві знадобиться знайти і вдягнути все екіпірування, щоб посилити свої характеристики.



Рисунок 3.21 – Битва з босом

Після битви з босом активується останній квест. У інвентарі гравця з'являється предмет для квесту, який неможливо викинути з інвентаря(рис.3.22).



Рисунок 3.22 – Фінальний квест Scene-3

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

Після виконання фінального квесту активується портал, який при взаємодії з ним перенесе гравця на четвертий рівень - Scene-4(рис.3.23).



Рисунок 3.23 – Активація portalу.

Після того, як гравець натисне клавішу F біля portalу, гравця загрузить на четвертий рівень – Scene-4. Scene-4 являє собою локацію з Scene-2 з усіма виконаними квестами. Для продовження проходження потрібно лише ініціювати діалог з чаклуном(рис.3.24).



Рисунок 3.24 – Scene-4

					КН.КР.Б – 122 – 23 – ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		64

Після діалогу автоматично загрузається Scene-5. Фінальна сцена гри, яка представляє собою візуальну новелу і логічно закінчує ігровий процес. Scene-5 налічує 21 локацію, які містять різні фони, аудіосупровід, спрайти персонажів і речення для проходження гри. Геймплейно Scene-5 повторює всі механіки вже рінаше описаної Scene-1.



Рисунок 3.25 – Scene-5

Після проходження гри, гравця зустрічає сцена з фінальними титрами розробника(рис.3.26.).



Рисунок 3.26 – Фінальні титри гри

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

Після показу титрів гравця автоматично перенесе до головного меню.

3.2 Тестування роботи програмного забезпечення

Проведено тестування гри для виявлення та усунення недоліків.

Таблиця 3.1

Результати тестування інтерфейсу та взаємодії в різних сценах гри

Сцена	Вхідні дані	Очікувані дані	Спостережені дані
Головне меню	Натиск на кнопку «Нова Гра»	Розпочинається нова гра	Розпочинається нова гра
	Натиск на кнопку «Продовжити»	Гра продовжується з файлу загрузки	Гра продовжується з файлу загрузки
	Натиск на кнопку «Налаштування»	Відкривається панель налаштувань	Відкривається панель налаштувань
	Натиск на кнопк «Вихід»	Виконується вихід з гри	Виконується вихід з гри
Scene-1	Натискати ЛКМ або пробіл	Перемикання речення	Перемикання речення
	Обирати варіант дії або діалогу	Обрано і показано правильну сцену	Обрано і показано правильну сцену
	Натиск кнопки ESC	Гра ставить на паузу, показано меню паузи	Гра ставить на паузу, показано меню паузи
Scene-2	Натискання кнопок управління персонажем	Персонаж рухається відносно натиснутої кнопки	Персонаж рухається відносно натиснутої кнопки
	Натиск кнопки Alt	Показана панель зміни персонажів	Показана панель зміни персонажів
	Обрати іншого персонажа	Обрано іншого персонажа	Обрано іншого персонажа
	Натиск кнопки ЛКМ, під час гри за солдата	Гравець починає стріляти	Гравець починає стріляти
	Натиск клавіші E	Взаємодія з предметами/NPC	Взаємодія з предметами/NPC

	Натиск клавіші ESC	Гра ставить на паузу, показано меню паузи	Гра ставить на паузу, показано меню паузи
Scene-3	Натискати ПКМ	Герой рухається до точки куди натиснув гравець	Герой рухається до точки куди натиснув гравець
	Натиснути на предмет клавішею ПКМ	Предмет підбирається в інвентарь	Предмет підбирається в інвентарь
	Натиснути на ворога клавішею ПКМ	Герой вносить шкоду ворогу	Герой вносить шкоду ворогу
	Натиск кнопки ESC	Гра ставить на паузу, показано меню паузи	Гра ставить на паузу, показано меню паузи
	Натиснути кнопку F	Взаємодія з предметами/NPC	Взаємодія з предметами/NPC
Scene-4	Керування грою відбувається аналогічно Scene-2		
Scene-5	Керування грою відбувається аналогічно Scene-1		

При тестуванні були виявлені помилки, незручності та недоліки гри. Наприклад, через деякі об'єкти на Scene-3 персонаж проходив, хоча не повинен був. Це було виправлено шляхом додавання відповідних колайдерів та регулювання системи NavMesh.

Було багато помилок з системою регенування здоров'я, серед яких здоров'я гравця на шкалі регенувалося а по даним гри – ні.

Висновки до третього розділу

У третьому розділі детально розглянуто та описано інтерфейс користувача у грі. Наведено приклади скриншотів з додатку та розказано, які дії може виконати гравець при певних обставинах.

Описано процес тестування: які дії повинні до чого призвести, та що відбувалося насправді. Виявлено деякі помилки. Їх було проаналізовано та виправлено.

ВИСНОВКИ

У результаті виконання випускної кваліфікаційної роботи було проведено глибокий аналіз ринку відеоігор, визначено потенціал для нововведень через гібридизацію жанрів Adventure-RPG, Platformer та Visual Novel. Використання рушія Unity 3D дозволило реалізувати цей ігровий досвід, використовуючи його розширені можливості та гнучкість.

У процесі аналізу вимог до програмного продукту було чітко визначено очікування користувачів та встановлені функціональні та нефункціональні вимоги до гри. Це дозволило чітко визначити обсяг робіт та спрямувати зусилля на найважливіші аспекти розробки. Детальний аналіз алгоритмів гри, а також розробка та демонстрація основних методів, дозволили успішно реалізувати плановану функціональність гри. Опис та аналіз інтерфейсу користувача забезпечили підтвердження зручності та інтуїтивності ігрового процесу. Тестування гри дозволило виявити та виправити декілька помилок, що в кінцевому результаті забезпечило високу якість продукту.

У цілому, завдання, поставлені перед випускною кваліфікаційною роботою, були успішно виконані. Результати роботи показують, що Unity 3D є міцним інструментом для розробки ігрових додатків, і з його допомогою було створено високоякісний продукт.

					КН.КР.Б – 122 – 23 – ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		69

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Створення руху персонажа і робота з камерою Cinemachine [Електронний ресурс] – Режим доступу до ресурсу:
<https://www.youtube.com/watch?v=4HpC--2iowE>
2. Створення системи діалогів [Електронний ресурс] – Режим доступу до ресурсу: <https://youtu.be/8oTYabhj248>
3. Створення системи квестів [Електронний ресурс] – Режим доступу до ресурсу: <https://youtu.be/-65u991cdtw>
4. Створення сцен і тексту візуальної новели [Електронний ресурс] – Режим доступу до ресурсу: <https://youtu.be/33XDR6E5W7g>
5. Асинхронне завантаження [Електронний ресурс] – Режим доступу до ресурсу:
<https://docs.unity3d.com/ScriptReference/SceneManager.LoadSceneAsync.html>
6. Система зберігання та завантаження на основі запису у файли [Електронний ресурс] – Режим доступу до ресурсу:
https://youtu.be/XOjd_qU2Ido
7. Внутрішньоігрові налаштування [Електронний ресурс] – Режим доступу до ресурсу: <https://youtu.be/YOaYQrN1oYQ>
8. Робота з AudioManager [Електронний ресурс] – Режим доступу до ресурсу: <https://docs.unity3d.com/ScriptReference/Audio.AudioMixer.html>
9. Використання NavMesh у Unity [Електронний ресурс] – Режим доступу до ресурсу: <https://docs.unity3d.com/ScriptReference/AI.NavMesh.html>
10. Як тестувати гру [Електронний ресурс] – Режим доступу до ресурсу:
https://youtu.be/b9U1dcUS_-w
11. Unreal Engine [Електронний ресурс] – Режим доступу до ресурсу:
<https://www.unrealengine.com/en-US>
12. Frostbite Engine [Електронний ресурс] – Режим доступу до ресурсу:
[https://en.wikipedia.org/wiki/Frostbite_\(game_engine\)](https://en.wikipedia.org/wiki/Frostbite_(game_engine))

13. Rockstar Advanced Game Engine [Електронний ресурс] – Режим доступу до ресурсу:
https://en.wikipedia.org/wiki/Rockstar_Advanced_Game_Engine
14. Unity Documentation [Електронний ресурс] – Режим доступу до ресурсу: <https://docs.unity3d.com/Manual/index.html>
15. Жанри відеоігор [Електронний ресурс] – Режим доступу до ресурсу:
<https://www.hp.com/us-en/shop/tech-takes/video-game-genres>
16. Як створити 2D платформер [Електронний ресурс] – Режим доступу до ресурсу: <https://youtu.be/UbPiCgCkHTE>
17. Створення системи інвентаря [Електронний ресурс] – Режим доступу до ресурсу: https://youtu.be/w6_fetj9PIw
18. Unity 3d пул об'єктів [Електронний ресурс] – Режим доступу до ресурсу: https://docs.unity3d.com/ScriptReference/Pool.ObjectPool_1.html
19. Unity 3d Sprite Renderer [Електронний ресурс] – Режим доступу до ресурсу: <https://docs.unity3d.com/Manual/class-SpriteRenderer.html>
20. 2D освітлення в Unity [Електронний ресурс] – Режим доступу до ресурсу: <https://youtu.be/nkgGyO9VG54>

ДОДАТКИ

Найбільш важливі функції

Скрипт для зберігання гри:

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

public class SaveManager : MonoBehaviour
{
    private static string SAVED_GAME = "/saveData.json";

    // Збереження гри
    public static void SaveGame(SaveData data)
    {
        string json = JsonUtility.ToJson(data);
        File.WriteAllText(Application.persistentDataPath + SAVED_GAME, json);
    }

    // Завантаження гри
    public static SaveData LoadGame()
    {
        string path = Application.persistentDataPath + SAVED_GAME;
        if (File.Exists(path))
        {
            string json = File.ReadAllText(path);
            Debug.Log(Application.persistentDataPath);
            return JsonUtility.FromJson<SaveData>(json);
        }
        else
        {
            Debug.LogError("Save file not found in " + path);
            return new SaveData();
        }
    }

    // Перевірка наявності збереженої гри
    public static bool IsGameSaved()
    {
        string path = Application.persistentDataPath + SAVED_GAME;
        return File.Exists(path);
    }

    // Видалення збереженої гри
    public static void ClearSavedGame()
    {
        string path = Application.persistentDataPath + SAVED_GAME;
        if (File.Exists(path))
        {
            File.Delete(path);
        }
    }
}

// Структура для зберігання даних гри
public struct SaveData
{
    public Vector3 cameraPosition; // Позиція камери
    public Vector3 playerPosition; // Позиція гравця
    public Vector3 playerMotorPosition; // Позиція гравця (сцена 3)
    public Vector3 wizzardPosition; // Позиція чарівника
}
```

```

public Vector3 treePosition;// Позиція дерева
public Vector3 powerStonePosition; // Позиція каменю сили
public int sentence;// Номер речення
public int currentScene;// Номер сцени
public int currentDialogIndex;// Номер діалогу
public int currentSentenceIndex; // Поточний індекс речення
public int currentQuestIndex; // Поточний індекс квесту
public int playerHealth; // Здоров'я гравця
public float InteractionRadius; // Радіус взаємодії
public float wizzardInteractionDistance; // Відстань взаємодії з чарівником
public bool dialogCompleted; // Діалог завершено
public bool isTreeDestroyed; // Дерево знищено
public bool emptyWallActive; // Стан стіни
public bool wallActive; // Стан стіни
public bool isPowerStoneInInventory; // Камінь сили в інвентарі
public bool isPowerStoneActive; // Камінь сили активний
public bool questActivated; // Квест активовано
public bool isAnimationPlayed; // Анімація відтворена
public bool isCameraAnimating; // Анімація камери
public bool isPortalAnimationActive; // Анімація порталу
public List<Quest> questStates; // Список квестів
public List<int> prevScenes; // Список попередніх сцен
public Quaternion treeRotation;// Поворот дерева
public bool fadeInCalled; // Анімація панелі квестів
public bool isPlayerInRange; // Гравець у зоні взаємодії
public bool hasDialogueFinished; // Діалог завершено
public List<Item> inventoryItems; // Список предметів у інвентарі
public List<bool> itemsPickedUpStates; // Список станів предметів
public List<ItemState> itemStates; // Список станів предметів
public string[] equippedItemIds; // Список id предметів
public EquipmentSlot[] equippedSlots; // Список слотів
public int itemsPickedUpCount; // Поле для itemsPickedUpCount
}

```

Головний скрипт для меню паузи:

```

using System.Collections;
using UnityEngine;
using UnityEngine.SceneManagement;
using UnityEngine.UI;

public abstract class BasePauseMenu : MonoBehaviour
{
    public GameObject pauseMenuUI;
    private CanvasGroup pauseMenuCanvasGroup;
    public AudioSource musicSource;
    public AudioSource soundSource;
    public AudioSource environmentSource;
    public static bool isPaused = false;
    public float fadeDuration = 0.5f;
    public Image saveIcon;
    private float initialMusicVolume;
    private float initialSoundVolume;
    private float initialEnvironmentVolume;
    [SerializeField]
    public bool controlCursorVisibility = true;

    private void Awake()
    {
        initialMusicVolume = musicSource.volume;
        initialSoundVolume = soundSource.volume;
        initialEnvironmentVolume = environmentSource.volume;
    }
}

```

```

    }

    protected virtual void Start()
    {
        pauseMenuCanvasGroup = pauseMenuUI.GetComponent<CanvasGroup>();
    }

    protected virtual void Update()
    {
        if (Input.GetKeyDown(KeyCode.Escape))
        {
            if (isPaused)
            {
                Resume();
            }
            else
            {
                Pause();
            }
        }

        if (Input.GetMouseButtonDown(1) && isPaused)
        {
            Resume();
        }
    }

    public virtual void Resume()
    {
        StartCoroutine(FadeOutMenu());
    }

    public virtual void Pause()
    {
        StartCoroutine(FadeInMenu());
    }

    public void ToggleCursor(bool visible)
    {
        Cursor.visible = visible;
        Cursor.lockState = visible ? CursorLockMode.None : CursorLockMode.Locked;
    }

    protected IEnumerator FadeInMenu()
    {
        pauseMenuUI.SetActive(true);
        float elapsedTime = 0f;
        while (elapsedTime < fadeDuration)
        {
            elapsedTime += Time.unscaledDeltaTime;
            pauseMenuCanvasGroup.alpha = Mathf.Lerp(0, 1, elapsedTime / fadeDuration);
            yield return null;
        }
        pauseMenuCanvasGroup.alpha = 1;
        Time.timeScale = 0f;
        isPaused = true;
        if (controlCursorVisibility)
        {
            ToggleCursor(true);
        }
    }

    protected IEnumerator FadeOutMenu()
    {
        float elapsedTime = 0f;
    }

```

```

        while (elapsedTime < fadeDuration)
        {
            elapsedTime += Time.unscaledDeltaTime;
            pauseMenuCanvasGroup.alpha = Mathf.Lerp(1, 0, elapsedTime / fadeDuration);
            yield return null;
        }
        pauseMenuCanvasGroup.alpha = 0;
        pauseMenuUI.SetActive(false);
        Time.timeScale = 1f;
        isPaused = false;
        if (controlCursorVisibility)
        {
            ToggleCursor(false);
        }
    }

    protected IEnumerator FadeOutAudioSources()
    {
        float elapsedTime = 0f;
        while (elapsedTime < fadeDuration)
        {
            elapsedTime += Time.unscaledDeltaTime;

            soundSource.volume = Mathf.Lerp(initialSoundVolume, 0, elapsedTime /
fadeDuration);
            environmentSource.volume = Mathf.Lerp(initialEnvironmentVolume, 0, elapsedTime
/ fadeDuration);
            yield return null;
        }
        soundSource.Pause();
        environmentSource.Pause();
    }

    protected IEnumerator FadeInAudioSources()
    {
        soundSource.UnPause();
        environmentSource.UnPause();

        float elapsedTime = 0f;
        while (elapsedTime < fadeDuration)
        {
            elapsedTime += Time.unscaledDeltaTime;
            soundSource.volume = Mathf.Lerp(0, initialSoundVolume, elapsedTime /
fadeDuration);
            environmentSource.volume = Mathf.Lerp(0, initialEnvironmentVolume, elapsedTime
/ fadeDuration);
            yield return null;
        }
    }

    public void ToMainMenu()
    {
        Time.timeScale = 1f;
        if (isPaused)
        {
            isPaused = false;
        }
        SceneManager.LoadScene("MainMenu");
    }

    public void QuitGame()
    {
        Debug.Log("Quit");
        Application.Quit();
    }

```

```

public void PlaySaveAnimation()
{
    StartCoroutine(SaveIconAnimation());
}

private IEnumerator SaveIconAnimation()
{
    float fadeInDuration = 0.2f;
    float rotationDuration = 0.2f;
    float fadeOutDuration = 0.2f;

    float elapsedTime = 0f;
    saveIcon.gameObject.SetActive(true);

    // Поява картинки
    while (elapsedTime < fadeInDuration)
    {
        elapsedTime += Time.unscaledDeltaTime;
        saveIcon.color = new Color(saveIcon.color.r, saveIcon.color.g,
saveIcon.color.b, Mathf.Lerp(0, 1, elapsedTime / fadeInDuration));
        yield return null;
    }
    saveIcon.color = new Color(saveIcon.color.r, saveIcon.color.g, saveIcon.color.b,
1);

    elapsedTime = 0f;

    // Обертання картинки
    float startRotation = saveIcon.rectTransform.eulerAngles.z;
    float targetRotation = startRotation + 360f;
    while (elapsedTime < rotationDuration)
    {
        elapsedTime += Time.unscaledDeltaTime;
        float currentRotation = Mathf.Lerp(startRotation, targetRotation, elapsedTime
/ rotationDuration);
        saveIcon.rectTransform.eulerAngles = new Vector3(0, 0, currentRotation);
        yield return null;
    }
    saveIcon.rectTransform.eulerAngles = new Vector3(0, 0, targetRotation);

    elapsedTime = 0f;

    // Зникнення картинки
    while (elapsedTime < fadeOutDuration)
    {
        elapsedTime += Time.unscaledDeltaTime;
        saveIcon.color = new Color(saveIcon.color.r, saveIcon.color.g,
saveIcon.color.b, Mathf.Lerp(1, 0, elapsedTime / fadeOutDuration));
        yield return null;
    }
    saveIcon.color = new Color(saveIcon.color.r, saveIcon.color.g, saveIcon.color.b,
0);
    saveIcon.gameObject.SetActive(false);
}
}

```

GameController для Scene-1:

```

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

```

```

using UnityEngine;
using UnityEngine.EventSystems;
using UnityEngine.UI;
using TMPro;
public class GameController : MonoBehaviour
{
    public GameScene currentScene; // Посилання на поточну сцену, що відтворюється
    public BottomBarController bottomBar; // Посилання на компонент BottomBarController
    public SpriteSwitcher backgroundController; // Посилання на компонент SpriteSwitcher
    public ChooseController chooseController; // Посилання на компонент ChooseController
    public AudioController audioController; // Посилання на компонент AudioController

    private bool leftMouseButtonEnabled = true;
    private bool rightMouseButtonEnabled = true;
    private bool spacebarEnabled = true;
    private BasePauseMenu basePauseMenu;
    public static bool GameIsPaused = false;
    public DataHolder data;

    public Image bottomBarImage;

    public TextMeshProUGUI textObject1;
    public TextMeshProUGUI textObject2;
    public GameObject panel;
    public Animator transition;
    private State state = State.IDLE; // Enum для збереження поточного стану гри
    private List<StoryScene> history = new List<StoryScene>(); // Список для збереження
історії відтворених сцен
    private enum State // Enum для зберігання можливих станів гри
    {
        IDLE, ANIMATE, CHOOSE
    }

    // Вивантажуємо всі невикористані ресурси з пам'яті
    private void UnloadUnusedAssets()
    {
        Resources.UnloadUnusedAssets();
        System.GC.Collect();
    }

    void Start()
    {
        basePauseMenu = FindObjectOfType<BasePauseMenu>();
        basePauseMenu.ToggleCursor(true);
        StartLevel();
        UnloadUnusedAssets();
        if (SaveManager.IsGameSaved())
        {
            SaveData data = SaveManager.LoadGame();
            data.prevScenes.ForEach(scene =>
            {
                history.Add(this.data.scenes[scene] as StoryScene);
            });
            if (history.Count > 0) // Перевірте, чи історія не порожня
            {
                currentScene = history[history.Count - 1];
                history.RemoveAt(history.Count - 1);
                bottomBar.SetSentenceIndex(data.sentence - 1);
            }
        }

        // Перевірка якщо поточна сцена є StoryScene
        if (currentScene is StoryScene)
        {

```

```

        StoryScene storyScene = currentScene as StoryScene; // Транслюємо поточну
        сцену до StoryScene і зберігаємо її у змінній
        history.Add(storyScene); // Додати storyScene до списку history
        bottomBar.PlayScene(storyScene, bottomBar.GetSentenceIndex()); // Відтворення
        сюжетної сцени за допомогою BottomBarController
        backgroundController.SetImage(storyScene.background); // Встановлюємо фонове
        зображення за допомогою компонента SpriteSwitcher
        PlayAudio(storyScene.sentences[bottomBar.GetSentenceIndex()]); // Відтворення
        звукової доріжки для сцени
    }
}

IEnumerator StartLevel()
{
    transition.SetTrigger("Start");
    yield return new WaitForSeconds(1f);
}

void Update()
{
    if (state != State.IDLE || PauseMenuController.isPaused)
    {
        return;
    }
    bool isClickingIgnoredUI = false;
    if (EventSystem.current.IsPointerOverGameObject())
    {
        GameObject clickedObject = EventSystem.current.currentSelectedGameObject;
        if (clickedObject != null && clickedObject.CompareTag("IgnoreTextOutput"))
        {
            isClickingIgnoredUI = true;
        }
    }

    if (textObject1.gameObject.activeSelf && textObject2.gameObject.activeSelf &&
    bottomBarImage.enabled && panel.gameObject.activeSelf && Input.GetMouseButtonDown(0))
    {
        if (!isClickingIgnoredUI && (Input.GetKeyDown(KeyCode.Space) &&
        spacebarEnabled && leftMouseButtonEnabled && rightMouseButtonEnabled ||
        leftMouseButtonEnabled))
        {
            if (bottomBar.IsCompleted())
            {
                bottomBar.StopTyping();
                if (bottomBar.IsLastSentence())
                {
                    audioController.StopAllAudio();
                    PlayScene((currentScene as StoryScene).nextScene);
                }
                else
                {
                    bottomBar.PlayNextSentence();
                    PlayAudio((currentScene as
                    StoryScene).sentences[bottomBar.GetSentenceIndex()]);
                }
            }
            else
            {
                bottomBar.SpeedUp();
            }
        }
    }
    else if (Input.GetMouseButtonDown(0))
    {
        ToggleObjects();
    }
}

```

```

    }

    if (Input.GetMouseButtonDown(2))
    {
        ToggleObjects();
    }
}

public void ToggleObjects()
{
    bool setActive = !textObject1.gameObject.activeSelf;

    bottomBarImage.enabled = setActive;
    textObject1.gameObject.SetActive(setActive);
    textObject2.gameObject.SetActive(setActive);
    leftMouseButtonEnabled = setActive;
    rightMouseButtonEnabled = setActive;
    spacebarEnabled = setActive;
    panel.gameObject.SetActive(setActive);
}

public void SaveGame()
{
    Debug.Log("Збереження гри...");
    basePauseMenu.PlaySaveAnimation();
    List<int> historyIndicies = new List<int>(); // Створюємо список для зберігання
індексів сцен історії
    history.ForEach(scene =>
    {
        historyIndicies.Add(this.data.scenes.IndexOf(scene)); // Додаємо індекси сцен
історії до списку індексів
    });
    SaveData data = new SaveData
    {
        sentence = bottomBar.GetSentenceIndex(), // Отримуємо індекс поточного речення
знизу екрану
        prevScenes = historyIndicies // Передаємо список індексів сцен історії в
об'єкт збереження
    };
    SaveManager.SaveGame(data); // Зберігаємо гру за допомогою менеджера збережень
}

// PlayScene метод дозволяє відтворити певну сцену в гри
// Якщо для прапорця isAnimated встановлено значення true, сцена відтворюватиметься з
анімацією
public void PlayScene(GameScene scene, int sentenceIndex = -1, bool isAnimated = true)
{
    StartCoroutine(SwitchScene(scene, sentenceIndex, isAnimated)); // Запуск coroutine
(співпрограми) для перемикання на вказану сцену
}

// Ця корутина обробляє фактичне перемикання сцен
private IEnumerator SwitchScene(GameScene scene, int sentenceIndex = -1, bool
isAnimated = true)
{
    state = State.ANIMATE; // Встановити стан на ANIMATE, який вказує на те, що
анімація триває
    currentScene = scene; // Встановлюємо поточну сцену на вказану сцену

    if (isAnimated) // Якщо isAnimated = true, ховаємо bottomBar і чекаємо 1 секунду
    {
        bottomBar.Hide();
        yield return new WaitForSeconds(1f);
    }
}

```

```

        // Якщо сцена представляє собою StoryScene, відтворюємо вказане речення та
оновлюємо фонове зображення та нижню панель
        if (scene is StoryScene)
        {
            StoryScene storyScene = scene as StoryScene; // Транслюємо сцену до StoryScene
і додаємо її до списку історії
            history.Add(storyScene);
            PlayAudio(storyScene.sentences[sentenceIndex + 1]); // Відтворення аудіо

            // Якщо isAnimated = true, змінюємо фонове зображення з анімацією. В іншому
випадку просто встановлюємо зображення
            if (isAnimated)
            {
                backgroundController.SwitchImage(storyScene.background);
                yield return new WaitForSeconds(1f);
            }
            else
            {
                backgroundController.SetImage(storyScene.background);
            }

            // Очищаємо текст у bottomBar та показуємо його з анімацією чи без неї залежно
від прапорця isAnimated
            bottomBar.ClearText();
            if (isAnimated)
            {
                bottomBar.Show();
                yield return new WaitForSeconds(1f);
            }
            bottomBar.PlayScene(storyScene, sentenceIndex, isAnimated);
            // Встановити стан IDLE, який вказує на те, що сцена готова прийняти введення
від гравця
            state = State.IDLE;
        }
        // Якщо сцена представляє собою ChooseScene, встановлюємо стан на CHOOSE та
показуємо вибір гравцеві
        else if (scene is ChooseScene)
        {
            state = State.CHOOSE;
            chooseController.SetupChoose(scene as ChooseScene);
        }
    }
    // Цей метод відтворює аудіо для певного речення в StoryScene
    private void PlayAudio(StoryScene.Sentence sentence)
    {
        audioController.PlayAudio(sentence.music, sentence.sound, sentence.environment);
    }
}

```

PlayerSwitcher.cs – скрипт для зміни гравця

```

using UnityEngine;
using Cinemachine;
using UnityEngine.UI;

public class PlayerSwitchController : MonoBehaviour
{
    public GameObject foxPlayer;
    public GameObject soldatenPlayer;
}

```

```

public GameObject currentPlayer;
public CinemachineVirtualCamera virtualCamera;
public BasePauseMenu pauseMenu;
private InventoryUIController inventoryController;
private Player foxPlayerScript;
private Player soldatenPlayerScript;
void Start()
{
    currentPlayer = foxPlayer;
    soldatenPlayer.SetActive(false);
    inventoryController = FindObjectOfType<InventoryUIController>();
    foxPlayerScript = foxPlayer.GetComponent<Player>();
    soldatenPlayerScript = soldatenPlayer.GetComponent<Player>();
}

void Update()
{
    SyncCharacterTransforms();
    HandleCursorVisibility();
}

public void SwitchToFox()
{
    currentPlayer = foxPlayer;
    foxPlayer.SetActive(true);
    foxPlayerScript.StartCoroutine(foxPlayerScript.RegenHealth());
    int tempHealth = soldatenPlayerScript.health; // Зберігаємо поточне здоров'я
солдата
    foxPlayerScript.health = tempHealth; // Встановлюємо здоров'я лисиці таким же, як
у солдата
    foxPlayerScript.healthSlider.maxValue = 250; // Встановлюємо максимальне значення
слайдера
    foxPlayerScript.healthSlider.value = tempHealth; // Оновлюємо значення слайдера
здоров'я
    soldatenPlayer.SetActive(false);
    pauseMenu.ToggleCursor(false);
}

public void SwitchToSoldaten()
{
    currentPlayer = soldatenPlayer;
    soldatenPlayer.SetActive(true);
    soldatenPlayerScript.StartCoroutine(soldatenPlayerScript.RegenHealth());
    int tempHealth = foxPlayerScript.health; // Зберігаємо поточне здоров'я лисиці
    soldatenPlayerScript.health = tempHealth; // Встановлюємо здоров'я солдата таким
же, як у лисиці
    soldatenPlayerScript.healthSlider.maxValue = 250; // Встановлюємо максимальне
значення слайдера
    soldatenPlayerScript.healthSlider.value = tempHealth; // Оновлюємо значення
слайдера здоров'я
    foxPlayer.SetActive(false);

```

```

        pauseMenu.ToggleCursor(visible: true);
    }

    void ResetAnimator(Animator animator)
    {
        animator.Play("Idle", 0, 0f);
        animator.SetBool("IsJumping", false);
        animator.SetBool("IsCrouching", false);
    }

    void SyncCharacterTransforms()
    {
        if (currentPlayer == foxPlayer)
        {
            soldatenPlayer.transform.position = currentPlayer.transform.position;
        }
        else
        {
            foxPlayer.transform.position = currentPlayer.transform.position;
        }
    }

    void HandleCursorVisibility()
    {
        if (currentPlayer == foxPlayer)
        {
            pauseMenu.ToggleCursor(false);
        }
        else
        {
            pauseMenu.ToggleCursor(true);
        }
        if (inventoryController.inventoryPanel.alpha > 0)
        {
            pauseMenu.ToggleCursor(true);
        }
        if (BasePauseMenu.isPaused)
        {
            pauseMenu.ToggleCursor(true);
        }
    }
}

```