

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 комп'ютерної гри «ZTURush»»

Виконала студентка групи КН-20-1
ЯЦИШИН-КУЛІШ Анна Святославівна

Керівник роботи:
ГРАФ Марина Сергіївна

Рецензент:
КУЗЬМЕНКО Олександр Вікторович

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

ЗАТВЕРДЖУЮ

Завідувач кафедри комп'ютерних наук

Марина ГРАФ

«25» січня 2024 р.

ЗАВДАННЯ

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

Здобувач вищої освіти: **ЯЦИШИН-КУЛІШ Анна Святославівна**

Керівник роботи: **ГРАФ Марина Сергіївна**

Тема роботи: **«Розробка комп'ютерної гри «ZTURush»»,**

затверджена наказом закладу вищої освіти від **«25» січня 2024 р., №22/с**

Термін здачі закінченої роботи: **«15» червня 2024 р.**

Вихідні дані роботи (зазначається предмет і об'єкт дослідження):

Об'єктом дослідження є процес розробки комп'ютерних ігор з навчально-розважальним змістом. **Предметом дослідження** є алгоритми та методи створення інтерактивних сценаріїв ігрового процесу в освітніх комп'ютерних іграх.

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

Розділ	Консультант	Дата	
		Завдання видав	Завдання прийняв
1	Граф М.С.	05.03.2024р.	05.03.2024р.
2	Граф М.С.	10.04.2024р.	10.04.2014р.
3	Граф М.С.	23.05.2024р.	23.05.2024р.

Календарний план

№ з/п	Назва етапів випускної кваліфікаційної роботи	Термін виконання етапів роботи	Примітка
1	Постановка задачі. Пошук, огляд та аналіз аналогічних розробок. Формулювання технічного завдання. Опрацювання літературних джерел.	2 лютого 2024 – 5 березня 2024	Виконано
2	Проектування структури системи	6 березня 2024 – 10 квітня 2024	Виконано
3	Написання програмного коду	10 квітня 2024 – 6 травня 2024	Виконано
4	Тестування системи	6 травня 2024 – 23 травня 2024	Виконано
5	Оформлення пояснювальної записки	23 травня 2024 – 15 червня 2024	Виконано

Здобувач вищої освіти **Анна ЯЦИШИН-КУЛІШ**
Керівник **Марина ГРАФ**

РЕФЕРАТ

Випускна робота бакалавра складається з програмного комплексу грм «ZTURush» та пояснювальної записки. Пояснювальна записка до випускної роботи містить 88 сторінок тексту, 75 ілюстрацій, 3 таблиці, 4 додатків, налічує 20 найменувань літературних джерел.

Метою кваліфікаційної роботи є розробка комп'ютерної гри "ZTURush", яка поєднує елементи розваг та освіти, створює захоплюючий ігровий процес, а також стимулює гравців до вирішення логічних завдань та виконання навчальних квестів.

В роботі розглянуто аналітичну частину сфери ігрової розробки, визначено основні завдання на розробку гри та проаналізовано схожі додатки. Наведено сценарії роботи програмного комплексу, опис алгоритмів взаємодії окремих модулів, а також спроектовано та описано інтерфейс ігрового додатку.

Ключові слова: ГРА, UNITY, 3D, ОСВІТНЯ ГРА, ГРАФІЧНИЙ ІНТЕРФЕЙС, РУШІЙ, СЕРЕДОВИЩЕ РОЗРОБКИ, СЕРЕДОВИЩЕ МОДЕЛЮВАННЯ, C#.

					КН.КР.Б – 122 – 24 – ПЗ		
Змн.	Арк.	№ докум.	Підпис	Дата	Розробка комп'ютерної гри «ZTURush» Пояснювальна записка		
Розроб.	А.С. ЯЦИШИН-КУЛИШ						
Керівник	М.С. ГРАФ						
Рецензент	О.В. КУЗЬМЕНКО						
Н. контр.							
Зав. каф.	М.С. ГРАФ				Літ. Арк. Аркушів		
						3	88
					Житомирська політехніка, група КН-20-1		

ABSTRACT

The bachelor's thesis consists of the ZTURush software package and an explanatory note. The explanatory note to the graduation thesis contains 88 pages of text, 75 illustrations, 3 tables, 4 appendices, and 20 references.

The purpose of the qualification work is to develop a computer game "ZTURush", which combines elements of entertainment and education, creates an exciting gameplay, and stimulates players to solve logical problems and complete educational quests.

The paper considers the analytical part of the game development sphere, identifies the main tasks for game development and analyzes similar applications. Scenarios of the software system operation, description of the interaction algorithms of individual modules, as well as the game application interface are presented and described.

Keywords: GAME, UNITY, 3D, EDUCATIONAL GAME, GRAPHICAL INTERFACE, ENGINE, DEVELOPMENT ENVIRONMENT, MODELING ENVIRONMENT, C#.

					КН.КР.Б – 122 – 24 – ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		4

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ.....	6
ВСТУП	7
РОЗДІЛ 1 ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ.....	9
1.1 Аналіз ігрової індустрії на сьогодні.....	9
1.2 Постановка задачі.....	12
1.3 Аналіз аналогів	14
Висновки до першого розділу	21
РОЗДІЛ 2 ПРОЕКТУВАННЯ	22
2.1 Вибір середовища та інструментів реалізації.....	22
2.2 Концепція та сюжет гри.....	30
2.3 Ігрові механіки.....	32
2.4 Діаграми та алгоритми роботи.....	33
Висновки до другого розділу.....	41
РОЗДІЛ 3 РЕАЛІЗАЦІЯ.....	42
3.1 Розробка ігрової сцени та моделей.....	42
3.2 Розробка інтерфейсу гри.....	47
3.3 Реалізація геймплею	51
3.3.1 Керування персонажем	51
3.3.2 Взаємодія з NPC.....	55
3.3.3 Діалоги та квести	56
3.3.4 Життя та нагороди.....	62
3.3.5 Алгоритм збирання предметів	63
3.3.6 Керування проректором.....	65
3.4 Аудіосупровід та налаштування гучності звуку	68
Висновки до третього розділу	72
ВИСНОВКИ.....	73
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	74
ДОДАТКИ.....	76

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

AR - доповнена реальність;

FBX – технологія та формат файлів для обміну 3D-моделями;

FPS – кількість кадрів в секунду на екрані монітора;

NPC – неігровий персонаж;

VR – віртуальна реальність;

ШІ – штучний інтелект;

					КН.КР.Б – 122 – 24 – ПЗ	Арк.
						6
Змн.	Арк.	№ докум.	Підпис	Дата		

ВСТУП

Актуальність теми. Комп'ютерні ігри стали невід'ємною частиною сучасної культури та індустрії розваг. Вони дозволяють реалізувати різноманітні сценарії та втілювати креативні ідеї, які знаходять відгук у мільйонів гравців по всьому світу. Зокрема, освітні ігри набувають дедалі більшої популярності, оскільки вони поєднують розваги та навчання, сприяючи розвитку критичного мислення та логіки. Освітні ігри створюють інтерактивне середовище, де гравці можуть практикувати навички та отримувати знання в контексті реальних або вигаданих ситуацій. Цей підхід робить навчання більш захоплюючим і ефективним. Ігри можуть допомогти розвивати різноманітні навички, такі як критичне мислення, вирішення проблем, стратегічне планування та співпраця. Вони можуть також покращувати моторику та координацію. Ігри забезпечують високий рівень мотивації завдяки винагородам, досягненням та прогресу, що стимулює гравців продовжувати навчання та вдосконалення своїх навичок.

Гра може бути використана як додатковий засіб для підтримки навчального процесу. Вона може інтегруватися в навчальні програми як інтерактивний інструмент для практики та закріплення знань. Гра може використовуватися для самостійного навчання та саморозвитку, допомагаючи студентам покращувати свої знання та навички у вільний час. Залежно від складності завдань, гра може бути адаптована для різних рівнів підготовки студентів, що робить її універсальною для різних вікових категорій та рівнів знань.

Таким чином, освітні ігри не лише розважають, але й мають значний навчально-пізнавальний потенціал, що робить їх актуальними для освітніх програм та саморозвитку. Вони допомагають студентам розвивати важливі навички, підтримуючи їх мотивацію та зацікавленість у навчанні.

Мета дослідження. Мета роботи полягає у розробці комп'ютерної гри "ZTURush", яка поєднує елементи розваг та освіти, створює захоплюючий

					КН.КР.Б – 122 – 24 – ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		7

ігровий процес, а також стимулює гравців до вирішення логічних завдань та виконання навчальних квестів.

Для досягнення визначеної мети по основні завдання:

1. Розробити концепцію та сценарій гри "ZTURush";
2. Створити ігрових персонажів та визначити їх характеристики;
3. Спроекувати ігрове середовище та інтерфейс користувача;
4. Розробити алгоритми взаємодії гравця з NPC та реалізувати діалогові системи;
5. Реалізувати механіку збору предметів та систему нагород.

Об'єктом дослідження є процес розробки комп'ютерних ігор з навчально-розважальним змістом.

Предметом дослідження є алгоритми та методи створення інтерактивних сценаріїв ігрового процесу в освітніх комп'ютерних іграх.

Практичне значення отриманих результатів. Розроблена гра "ZTURush" має потенціал стати ефективним інструментом для освітніх установ та самостійного навчання, оскільки вона поєднує розважальний та навчальний аспекти. Результати даної роботи можуть бути використані для створення подібних ігор, які сприяють розвитку навичок та знань у гравців.

					КН.КР.Б – 122 – 24 – ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		8

РОЗДІЛ 1 ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Аналіз ігрової індустрії на сьогодні

За останні 20 років ігри та ігрова індустрія в цілому зробили великий крок вперед. Це стосується як графічної складової, так і поширення ігрової індустрії по всьому світу.

Графічні можливості ігор значно покращилися завдяки розвитку технологій. Від простих 2D-спрайтів здійснено перехід до складних 3D-моделей з високою деталізацією. З появою більш потужних комп'ютерів та ігрових консолей, а також широкого розповсюдження інтернету, ігри стали більш великими, складними та інтерактивними. Ігри тепер є важливою частиною глобальної культури. Вони використовуються не лише для розваги, але й для навчання, соціалізації та навіть терапії. Ігрова індустрія тепер є мультимільярдною галуззю, яка створює тисячі робочих місць по всьому світу.

Проте, минулий рік став роком масових скорочень в ІТ: по всьому світу роботу втратили понад 260 тис. спеціалістів, зокрема у геймдеві. У 2024 році ми також побачимо скорочення розробників ігор та бюджету, але масштаби будуть меншими. 2023 рік відкрив світу десятки успішних ігор. Однак цього року буде менше релізів, що позначиться на доходах студій та маркетплейсів, так як індустрія намагатиметься уникати ризиків і розробники зроблять ставку на перевірені франшизи. У той час як великі студії будуть зайняті випуском франшизи, інді-розробники збережуть ініціативу щодо створення свого наступного шедевра[1].

Хоч ситуація із скороченням не втішна, проте, ігрова індустрія продовжує стрімко розвиватися, завдяки ряду ключових технологій та змін у вподобаннях споживачів. Зростання популярності ігор на різних платформах, включаючи консолі, мобільні пристрої та ПК, продовжує стимулювати інновації та конкуренцію серед розробників. Це дозволяє розвиватися та бути однією з найдинамічніших сфері, такій як кіберспорт.

					КН.КР.Б – 122 – 24 – ПЗ	Арк.
						9
Змн.	Арк.	№ докум.	Підпис	Дата		

Кіберспорт, або електронний спорт, перетворився з ніші на глобальне явище, що приваблює мільйони глядачів і значні інвестиції, а також популяризує ігрову індустрію.

Турніри з популярних кіберспортивних дисциплін, таких як Dota 2, League of Legends, CS, Overwatch та Fortnite, значно підвищують популярність цих ігор. Висвітлення кіберспорту у ЗМІ та на стрімінгових платформах допомагає залучати нових гравців та збільшує продажі ігор та пов'язаних з ними товарів. Завдяки кіберспорту підвищується рівень конкурентоспроможності в ігровій індустрії, змушуючи розробників створювати більш якісні, збалансовані та цікаві ігри. Успіх гри на кіберспортивній сцені може значно вплинути на її популярність та комерційний успіх, тому розробники прагнуть забезпечити найкращий можливий ігровий досвід. Також важливим аспектом є стимулювання розвитку нових технологій, таких як 5G, VR, AR та AI, які використовуються для покращення якості трансляцій та геймплею.

Розвиток технологій доповненої та віртуальної реальності, які відкривають нові можливості для створення захопливого ігрового досвіду і занурюючи гравців у більш реальні та інтерактивні світи, відіграє значну роль у ігровій індустрії. VR дозволяє гравцям вступити в роль своїх героїв, сприяючи глибшому зв'язку з ігровим світом, використовуючи спеціальні гарнітури та контролери. Відчуття реалістичності надає змогу повністю взаємодіяти з оточенням та персонажами та значно підвищує рівень емоційного залучення та задоволення від гри.

VR відкриває нові можливості для розробників у створенні ігрових механік. Ігри можуть включати фізичну активність, де гравці повинні рухатися, нахилятися, ховатися або використовувати жестові контролери для взаємодії з об'єктами. Це створює більш динамічні процеси.

Використання VR в ігровій індустрії розширює бачення меж традиційних ігор. Дана технологія використовується для навчальних та тренувальних ігрових програм, що дозволяє створювати симуляції реальних ситуацій у безпечному

					КН.КР.Б – 122 – 24 – ПЗ	Арк.
						10
Змн.	Арк.	№ докум.	Підпис	Дата		

віртуальному середовищі. Це включає тренування для військових, медичних працівників та інших професій, де точність та практика мають важливе значення.

Також віртуальна реальність надає розробникам повну свободу у створенні унікальних ігрових досвідів, дозволяє деталізувати світ, так як це було б неможливо у звичайних іграх. Це відкриває нові горизонти для художників, дизайнерів та програмістів.

У свою чергу, доповнена реальність створює можливості для нових жанрів ігор, таких як геолокаційні ігри, квести в реальному світі та інтерактивні навчальні ігри. AR-технології активно використовуються у створенні навчальних ігор, де віртуальні елементи допомагають гравцям краще засвоювати матеріал. Це може бути особливо корисно у сфері освіти, де інтерактивні ігрові досвіди сприяють ефективнішому навчанню та залученню учнів[2].

Ще однією технологією, що впливає на ігрову індустрію, надаючи можливість створювати більш складні та адаптивні ігрові світи, є штучний інтелект (ШІ). Завдяки ШІ, NPC можуть поводитися більш реалістично, реагувати на дії гравця та адаптувати свої стратегії. Це створює нові виклики для гравців.

ШІ дозволяє створювати процедурно згенеровані світи та контент, що робить кожен ігрову сесію унікальною. Це використовується у іграх з відкритим світом та у roguelike-іграх, де гравці досліджують нові рівні та світи, створені на основі алгоритмів ШІ. Технологія допомагає у покращенні графіки та анімацій, використовуючи алгоритми машинного навчання для створення більш реалістичних текстур, освітлення та рухів персонажів. Це підвищує загальну якість візуального досвіду у іграх.

Також ШІ використовується для збору та аналізу даних про поведінку гравців, що допомагає розробникам краще розуміти своїх користувачів. Це дозволяє адаптувати ігровий контент під потреби та вподобання гравців, підвищуючи активність та утримання аудиторії. Дана технологія може бути використана для створення інтелектуальних помічників, які допомагають гравцям у проходженні гри. Це можуть бути віртуальні тренери, які надають

					КН.КР.Б – 122 – 24 – ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		11

поради та підказки, або помічники, які допомагають з навігацією та вирішенням задач. Загалом, штучний інтелект сприяє розвитку технологій розширеної реальності, забезпечуючи більш реалістичні взаємодії, що включає розпізнавання рухів, голосове управління та інші інновації[3].

Технологія 5G забезпечує швидкий та надійний доступ до онлайн-ігор, значно зменшуючи затримки та покращуючи загальний досвід гравців. Це важливо для багатокористувацьких онлайн-ігор, де якість зв'язку є критичною.

5G сприяє розвитку хмарного геймінгу, дозволяючи гравцям запускати та грати в ігри, що обробляються на віддалених серверах, а не на локальних пристроях. Це означає, що користувачі можуть грати в ресурсомісткі ігри на будь-якому пристрої, незалежно від його потужності.

Хмарні технології стали невід'ємною частиною сучасної ігрової індустрії. Вони дозволяють зберігати та обробляти великі обсяги даних, забезпечувати безперебійний доступ до ігор та ресурсів, а також підтримувати різноманітні моделі монетизації. Хмарні сервіси також сприяють розвитку гібридних моделей ігор, де частина обчислювальних процесів виконується на сервері, що дозволяє зменшити вимоги до апаратного забезпечення кінцевих користувачів. Це відкриває нові можливості для розробників, дозволяючи їм створювати складніші та більш інтерактивні ігри без обмежень апаратних ресурсів[4].

1.2 Постановка задачі

У сучасному світі комп'ютерні ігри здобувають все більшу популярність як ефективний засіб розваги, навчання та розвитку різних навичок. Особливо це стосується ігор, які комбінують у собі елементи навчання з цікавим і захоплюючим ігровим процесом. Одним із напрямків розвитку комп'ютерних ігор є створення проєктів, спрямованих на навчання та розвиток особистості.

У даному контексті виникає актуальна задача розробки ігрового продукту, який не лише надихає та розважає, але й сприяє підвищенню навчальних можливостей аудиторії. В рамках цієї роботи виникає ідея створення

					КН.КР.Б – 122 – 24 – ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		12

комп'ютерної гри "ZTURush", яка забезпечить унікальне поєднання навчання та розваги для студентів університету.

Ця гра виступатиме як інноваційна освітня платформа, що дозволить студентам отримати новий досвід навчання, взаємодії та розвитку в захопливій та цікавій формі. Метою цієї роботи є розробка та реалізація ігрового продукту, який буде активно залучати аудиторію до навчання та сприяти їхньому особистісному зростанню та розвитку.

Гра "ZTURush" є low poly 3D грою, яка поєднує в собі нескладну, але захоплюючу геймплейну механіку з комічними діалогами.

1. Розробка теми, сценарію і сюжету гри:

- Створення гумористичного сценарію та сюжету;
- Розробка діалогів із комічним підтекстом для стимулювання інтересу гравців;

2. Створення персонажів та їхніх характеристик:

- Проектування унікальних low-poly 3D моделей персонажів;
- Розробка взаємодії з персонажами;

3. Проектування ігрового світу та інтерфейсу користувача:

- Створення привабливого 3D ігрового світу, який буде відображати тематику та атмосферу гри;
- Розробка інтуїтивного та зрозумілого інтерфейсу для більшого занурення гравців у гру;

4. Розробка інтерактивних завдань і геймплею:

- Створення логічних квестів та завдань, які сприятимуть розвитку навичок;
- Проектування логічної структури гри та визначення послідовності подій;

5. Звуковий дизайн та озвучення:

- Розробка звукових ефектів та музичного супроводу;
- Озвучення діалогів та ситуацій, що підсилює занурення у гру.

					КН.КР.Б – 122 – 24 – ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		13

Дані етапи охоплюють широкий спектр завдань – від вибору теми та сценарію до створення звукових ефектів та озвучення. Грамотна взаємодія всіх компонентів та аналіз особливостей жанру дозволить створити якісний та захоплюючий ігровий продукт, що відповідатиме зазначеній меті та зацікавить користувачів.

1.3 Аналіз аналогів

Як було зазначено раніше, аналіз жанру допоможе створити якісну комп'ютерну гру. Жанр, який буде розглянуто, як головний критерій «Пригодницька відеогра або квест». Оскільки ігор даного жанру існує велика кількість та вони істотно відрізняються між собою, було вирішено порівняти їх на основі якості графіки та простоти розуміння геймплею. Також буде розглянуто мови програмування та рушії, на яких були створені додатки.

У якості аналогів розглянемо такі ігри як:

- Bully, на базі рушія RenderWare;
- Hogwarts Legacy, на базі рушія Unreal Engine;
- Syberia: The World Before, на базі рушія Unity;

Усі вищезазначені ігри належать до жанру пригодницьких квестів і виконані у форматі 3D. У цьому розділі буде детально описано сюжети цих ігор, а також надано ілюстрації, що демонструють графіку та окремі елементи геймплею.

Огляд цих додатків допоможе визначити основні технічні характеристики, що є важливими при розробці власної гри. Такий аналіз дозволить врахувати ключові аспекти дизайну та функціональності, необхідні для створення якісного ігрового продукту.

Комп'ютерна гра **Bully** (рис. 1.1) – відеогра в жанрі пригодницький бойовик від третьої особи з відкритим світом, дія якої відбувається у шкільному середовищі. Гравець керує 15-річним підлітком-бунтарем Джеймсом Джиммі

					КН.КР.Б – 122 – 24 – ПЗ	Арк.
						14
Змн.	Арк.	№ докум.	Підпис	Дата		

Хопкінсом, який з самого початку гри демонструє свій важкий характер і проблемне минуле. Сюжет починається з того, що Джиммі відправляють до Булвортської Академії, вигаданого інтернату в Новій Англії. Гравець може вільно досліджувати шкільний кампус і місто або виконувати основні місії. У грі присутні численні міні-ігри, які використовуються як для заробітку грошей, так і для покращення навичок Джиммі.

Уроки в грі проводяться у формі міні-ігор, розбитих на п'ять рівнів зростаючої складності. Наприклад, граматику представлена як словесна сутичка, і по мірі проходження уроків Джиммі вчиться краще насміхатися, вибачатися перед хуліганами, старостами та поліцейськими.

Джиммі має доступ до широкого арсеналу зброї. Спочатку гри доступні рогатка, супер-рогатка, мішки з кульками, бомби-смердючки, а пізніше з'являються пляшкова ракетниця та картоплетет. Також у Джиммі є різноманітні транспортні засоби: переважно велосипеди з різними характеристиками, скутер, карт, скейтборд і, у разі покарання, газонокосарка[6].



Рисунок 1.1 – Представлення гри Bully

Гра “Bully” (або “Bully: Scholarship Edition”) була розроблена компанією Rockstar Vancouver і вперше випущена в 2006 році. “Bully” була побудована на рушії RenderWare, який був розроблений компанією Criterion Software. RenderWare був популярним серед розробників в той час і використовувався для

					КН.КР.Б – 122 – 24 – ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		15

створення ігор на різних платформах. Будучи написаним мовою C++ , двигун використовує програмні інтерфейси OpenGL і DirectX , що дозволяє розробникам створювати мультиплатформні ігри[6].

Рекомендовані системні вимоги для запуску даної гри досить не великі це:

- Операційна система: Windows XP/Windows Vista;
- Процесор: Intel Pentium 4 (3+ GHZ) / AMD Athlon 3000+;
- Пам'ять: 1 GB оперативна пам'ять, 4.7 GB вільного місця;
- Відеокарта: Nvidia 6800/7300, ATI Radeon X1300 (або краще) з підтримкою Shader 3.0;

Гра доступна на платформі Steam за низькою ціною.

Hogwarts Legacy (рис. 1.2) — це відеогра від третьої особи у жанрі рольового бойовика. Гравець контролює учня Гогвортської школи чарів і чаклунства, маючи можливість налаштовувати голос, зовнішність, стать персонажа та вибрати гуртожиток на початку гри. Загалом можна створити до чотирьох різних персонажів. Під час гри гравець виконує сюжетні та побічні квести для просування історії, налаштовує складність ігрового процесу, вивчає нові заклинання та здібності, варить зілля для використання в боях чи для розв'язання головоломок, а також приборкує магічних істот.

Досліджуючи відкритий світ гри, гравець знаходить спорядження для свого персонажа, яке можна вдосконалювати за внутрішньоігрову валюту — місячні камені. Гравець також може використовувати мітлу або приборканих істот для швидшого переміщення світом, причому характеристики мітли можна покращувати в міру проходження гри.

У грі є Кімната на вимогу, інтер'єр якої можна прикрашати різними предметами. Через портал у цій кімнаті гравець потрапляє до ділянки зі своїми магічними істотами, де він може їх годувати та доглядати за ними, отримуючи матеріали для створення спорядження.

Гравцю також доступні кілька бойових арен, де він змагається з неігровими персонажами і може протестувати свої здібності перед їх розблокуванням. За

					КН.КР.Б – 122 – 24 – ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		16

виконання бойових досягнень гравець отримує косметичні предмети, включаючи костюми для персонажа.

Гравець взаємодіє з неігровими персонажами через систему діалогового дерева і може залучати їх як компаньйонів для виконання квестів[7].



Рисунок 1.2 – Представлення гри Hogwarts Legacy

Hogwarts Legacy побудована на ігровому рушії Unreal Engine. Версія гри для PlayStation 5 пропонує два графічних режими — «Якість» та «Продуктивність». Вона також використовує можливості консолі та контролера DualSense, такі як технологія об'ємного звуку Tempest 3D AudioTech і оновлений тактильний зв'язок. Цей рушій дозволяє створювати високоякісні 3D-ігри та використовує мову програмування C++[8].

Для даної гри існують мінімальні системні вимоги, а рекомендовані системні вимоги включають:

- Операційна система: 64-bit Windows 10;
- Процесор: Intel Core i7-8700 (3.2Ghz) або AMD Ryzen 5 3600 (3.6Ghz);
- Пам'ять: 16 GB оперативна пам'ять, 85 GB доступного місця;
- Відеокарта: NVIDIA GeForce 1080 Ti, AMD Radeon RX 5700 XT або INTEL Arc A770;

Дана гра розповсюджується за допомогою сервісу Steam і її ціна становить 1599₴.

Syberia: The World Before (рис. 1.3) — це пригодницька гра в жанрі «вклади й клади», створена бельгійським художником коміксів Бенуа Сокалем. Гра розроблена та випущена компанією Microïds і є четвертою частиною серії *Syberia*.

Syberia: The World Before розгортається в двох часових лініях і розповідає історії двох головних героїнь: Дани Роуз у 1937 році та Кейт Вокер у 2004 році.

Дана Роуз — талановита молода піаністка, що живе у вигаданому європейському місті Ваген у період перед Другою світовою війною. Дана мріє про музичну кар'єру, але її життя різко змінюється через політичні події, які загрожують її родині та майбутньому. Гравець допомагає Дані долати труднощі та знайти своє місце в світі, що швидко змінюється.

Кейт Вокер, відома своїми пригодами в попередніх частинах серії *Syberia*, опиняється у важкій життєвій ситуації. Після подій попередніх ігор вона втрачає зв'язок із своїм минулим і намагається знайти новий сенс життя. Пошуки приводять її до відкриття таємниці, яка пов'язує її з Даною Роуз. Відтак, гравець допомагає Кейт розгадати цю загадку, досліджуючи нові локації та взаємодіючи з різними персонажами.

Протягом гри, гравці перемикаються між двома часовими лініями, досліджуючи різні місця, розв'язуючи головоломки та взаємодіючи з персонажами. Історії Дани та Кейт переплітаються, розкриваючи глибокі таємниці, які об'єднують їх долі[9].

Головоломки у грі *Syberia: The World Before* представлені у різних виглядах і складностях, від класичних логічних завдань до складних механізмів та комбінацій предметів. Головоломки розроблені таким чином, щоби заохочувати гравця використовувати логічне мислення, креативність та дослідницький підхід для розв'язання проблем та просування в історії гри.

					КН.КР.Б – 122 – 24 – ПЗ	Арк.
						18
Змн.	Арк.	№ докум.	Підпис	Дата		



Рисунок 1.3 – Представлення гри Syberia: The World Before

Рушієм гри Syberia: The World Before є Unity, основною мовою програмування якого є C#. Рекомендовані системні вимоги включають:

- Операційна система: Windows 10;
- Процесор: i7 8700 або вище;
- Пам'ять: 16 GB оперативної пам'яті та 22 GB вільного місця;
- Відеокарта: Nvidia RTX 2070 або AMD Radeon RX 6600 XT;

Гру можна придбати на платформі Steam або спробувати демо-версію.

Опісля детального огляду, виведено таблицю 1.1 результатів порівняння за визначеними критеріями для оцінки ігор. Для зручності розуміння обрано десятибальну систему оцінювання. Дані для порівняння ігор зібрані з відгуків користувачів платформи Steam та внесені у таблицю 1.1 нижче.

Таблиця 1.1 – Порівняння обраних ігор

Назва	Основні характеристики			
Bully	A	Графіка: 9/10	Музика: 7/10	Інтерфейс: 6/10
	B	Геймплей: 10/10	Режими: 8/10	Складність: 4/10
	C	Сюжет: 4/10	Персонажі: 10/10	Новизна: 2/10

Hogwarts Legacy	A	Графіка: 9/10	Музика: 9/10	Інтерфейс: 9/10
	B	Геймплей: 8/10	Режими: 9/10	Складність: 1/10
	C	Сюжет: 2/10	Персонажі: 7/10	Новизна: 8/10
Syberia: The World Before	A	Графіка: 8/10	Музика: 8/10	Інтерфейс: 8/10
	B	Геймплей: 9/10	Режими: 7/10	Складність: 9/10
	C	Сюжет: 10/10	Персонажі: 10/10	Новизна: 9/10

Після отримання результатів якості гри, виходячи з вище наведених оцінок за допомогою формули наведеної у таблиці 1.2, виводиться фінальна оцінка.

Таблиця 1.2 – Формула знаходження кінцевої оцінки гри

Оцінка = $(A+B+C)/3$	$A=(A1+A2+A3)/3$	A1	A2	A3
	$B=(B1+B2+B3)/3$	B1	B2	B3
	$C=(C1+C2+C3)/3$	C1	C2	C3

де A – оформлення;

B – механіка;

C – зміст;

Отримана оцінка базується на трьох основних критеріях та кількох складових, пов'язаних з жанром (у даному випадку - жанр пригодницький квест). Результативна оцінка представлена в таблиці 1.3.

Таблиця 1.3 – Фінальна оцінка для кожної з ігор

Назва	A	B	C	Оцінка (max 10)
Bully	7.33	7.33	5.33	6.7/10
Hogwarts Legacy	9	6	5.67	6.9/10
Syberia: The World Before	8	8.33	9.67	8.7/10

Таким чином, усі вищезазначені ігрові додатки належать до одного жанру – пригодницький квест, але вони значно відрізняються між собою. Ці відмінності

					КН.КР.Б – 122 – 24 – ПЗ	Арк.
						20
Змн.	Арк.	№ докум.	Підпис	Дата		

проявляються як у використаних технологіях та різних ігрових рушіях, так і у мові програмування. Крім того, є значні відмінності в якості графіки та обмеженнях, встановлених ігровими рушіями. У всіх трьох іграх відрізняються ступінь деталізації сюжету та глибина розкриття персонажів, їхніх дій та розвитку сюжету.

Висновки до першого розділу

Підсумовуючи, перший розділ став ключовим для розробки гри, оскільки надав важливі відомості та підґрунтя для подальших дій. Аналіз ігрової індустрії дозволив краще зрозуміти сучасні тенденції та вимоги гравців, що допоможе під час створення ігрового продукту. Це надало цінні уявлення про те, що сьогодні цікаво та актуально для аудиторії гри, а також допомогло зрозуміти, які елементи гри можуть викликати позитивні реакції від гравців.

Вивчення аналогів у схожих жанрах стало важливою частиною цього розділу. Це дало можливість виявити сильні та слабкі сторони інших ігор, що мають схожий концепт з грою "ZTURush". Зокрема, аналіз конкурентів дозволяє уникнути потенційних помилок та використати найкращі практики, що вже успішно застосовуються в інших іграх.

Загалом, цей розділ не лише став основою для подальшої розробки гри, а й допоміг зрозуміти контекст індустрії та визначити стратегію, яка дозволить створити ігровий продукт, що задовольнить потреби та очікування цільової аудиторії.

					КН.КР.Б – 122 – 24 – ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		21

РОЗДІЛ 2 ПРОЕКТУВАННЯ

2.1 Вибір середовища та інструментів реалізації

Ігровий рушій — це програмний компонент, який відповідає за технічну сторону будь-якої відеогри. Основні функції ігрового рушія включають рендеринг, фізику, звук, скрипти, анімацію, інтелект штучного інтелекту, мережевий код, керування пам'яттю та граф сцени. Це дозволяє розробникам створювати багатоплатформові ігри, спрощуючи процес розробки та забезпечуючи високу якість гри[10].

Unreal Engine 5

Unreal Engine – ігровий рушій (рис. 2.1), розроблений та підтримуваний компанією Epic Games. Спочатку він був призначений для створення шутерів від першої особи, проте його наступні версії успішно використовувалися в іграх різних жанрів, включаючи стелс-ігри, файтинги та масові багатокористувацькі рольові онлайн-ігри. Unreal Engine підтримує більшість існуючих платформ та операційних систем: Windows, Linux, MacOS, iOS, Android, Xbox One, Xbox 360, PlayStation 2, PlayStation 3, PlayStation 4, GameCube, Wii, Wii U, Nintendo Switch, PSP, Dreamcast[11].



Рисунок 2.1 – Ігровий двигун Unreal Engine

					КН.КР.Б – 122 – 24 – ПЗ	Арк.
						22
Змн.	Арк.	№ докум.	Підпис	Дата		

Рушій має зручний та простий інтерфейс, який покращується з кожною новою версією. Завдяки великій кількості налаштувань тіней, освітлення та рендеру, рушій дозволяє досягти фотореалістичної графіки. Інструментарій рушія значно спрощує роботу з елементами ландшафту та рослинності. Ігрову логіку можна розробляти без написання коду, використовуючи систему візуальних скриптів Blueprint, але більшість розробників надають перевагу мові програмування C++.

Вбудований редактор візуальних ефектів Cascade дозволяє налаштовувати систему частинок за допомогою різних модулів. Рушій також має редактор матеріалів, який використовує штучне затінення та дає повний контроль над зовнішнім виглядом об'єктів і персонажів. Широкий набір анімаційних інструментів допомагає редагувати сітки та анімації. Результати роботи можна відразу переглянути та за потреби редагувати.

Для створення відеороликів та різних анімаційних сцен вбудований Sequencer, інструмент, що дозволяє налаштовувати персонажів, камеру та освітлення. Unreal Engine підтримує режим віртуальної реальності з розширеними налаштуваннями управління рухом. Epic Games співпрацює з провідними розробниками програмного та апаратного забезпечення, забезпечуючи високу якість взаємодії з системами віртуальної та доповненої реальності. Розширена система штучного інтелекту дозволяє налаштовувати реалістичну поведінку персонажів гри[12].

Unreal Engine краще підходить для розробки глобальних 3D-ігор, хоча має широкий вибір платформ для готових проєктів та нових розробок.

Cry Engine 5

Остання версія ігрового рушія, розроблена та видана компанією Crytek, пропонує відмінну графіку. Cry Engine зосереджує увагу на візуальній складовій проєкту, тому в ньому є різноманітний функціонал та інструментарій для досягнення найвищої якості зображення (рис. 2.2). Рушій підтримує останню версію DirectX 12, набір API, що відповідає за візуальні аспекти гри[13].

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

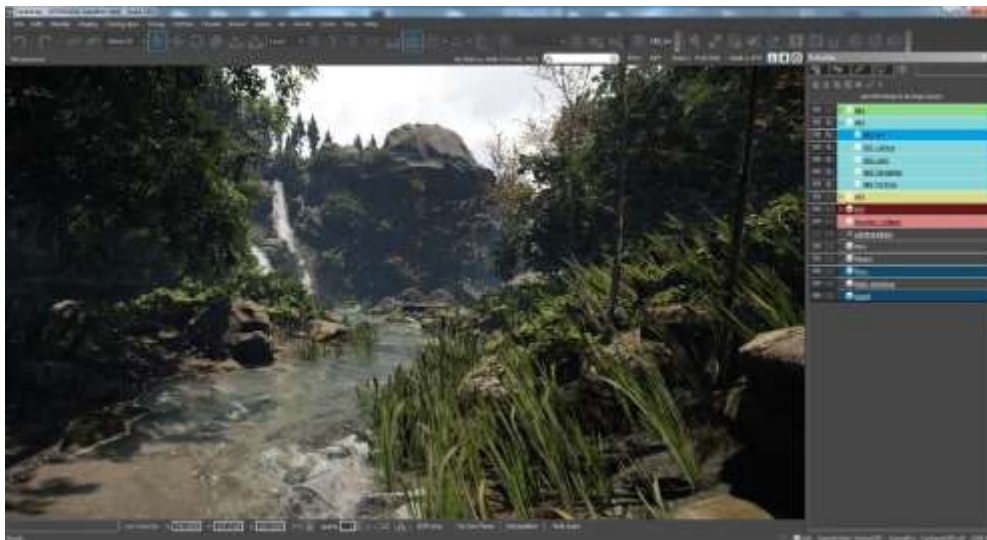


Рисунок 2.2 – Рушій Cry Engine

Physically Based Rendering – система в Cry Engine, що дозволяє імітувати взаємодію світла з матеріалами, використовуючи фізику реального світу, що надає об’єктам реалістичний вигляд. Завдяки динамічній обробці водних каустик досягається висока якість зображення води. Рушій також має інструменти для ефективного згладжування та зниження пікселізації зображення. Глобальне налаштування освітлення дозволяє використовувати більший тональний діапазон, ніж інші рушії, що покращує розпізнавання світлих, середніх та темних тонів без особливих зусиль.

Для роботи з матеріалами рушій має вбудований редактор Cry Engine SandBox, який пропонує великий інструментарій для створення рівнів та ігрових світів. За допомогою інструменту Designer Tool можна редагувати моделі та експортувати створені матеріали у зовнішні інструменти. Також існує редактор TrackView, що дозволяє створювати відеозаписи та інтерактивні сцени з об’єктами.

Як і в Unreal Engine 4, Cry Engine має систему візуальних скриптів FlowGraph, що дозволяє створювати та контролювати ігрову логіку без написання коду та скриптів вручну. Рушій має велику кількість налаштувань для анімації персонажів, включаючи параметричну кісткову анімацію. Вбудована

розширена система штучного інтелекту дозволяє налаштовувати реалістичну поведінку персонажів[14].

Cry Engine також може аналізувати продуктивність прямо під час гри, а вбудований інструмент Statoscope дозволяє у графічному вигляді слідкувати за витратою ресурсів комп'ютера для майбутньої оптимізації.

Unity

Unity — це кросплатформенний рушій для розробки ігор, створений компанією Unity Technologies. Unity підтримує майже всі існуючі платформи, що робить його надзвичайно універсальним і привабливим для розробників. Інтерфейс Unity простий та інтуїтивно зрозумілий, що дозволяє швидко освоїти роботу з ним як початківцям, так і досвідченим розробникам (рис. 2.3).

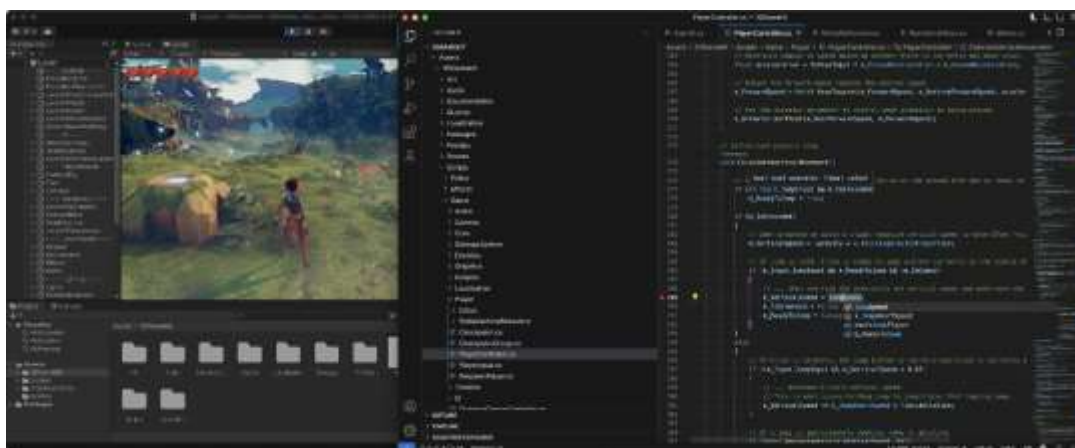


Рисунок 2.3 – Ігровий рушій Unity

Однією з основних переваг Unity є його багатofункціональність, яка робить цей рушій привабливим як для програмістів, так і для художників. Unity пропонує різні інструменти, серед яких:

- Timeline: інструмент для створення анімаційних сцен, який дозволяє легко анімувати об'єкти та персонажів.
- Cinemachine: набір динамічних камер, що дозволяють створювати складні кінематографічні ефекти.
- Progressive Lightmapper: інструмент для роботи з освітленням, який допомагає досягати реалістичного освітлення сцен.

- Autodesk Maya: інтеграція з програмою для моделювання та 3D-анімації, що дозволяє створювати високоякісні моделі та анімації.

Unity підтримує розробку проектів як у 2D, так і у 3D стилях, що додає йому ще більше універсальності. Для роботи з фізикою об'єктів Unity використовує рушії NVIDIA PhysX та Box2D, які забезпечують реалістичну фізику як у тривимірних, так і у двовимірних іграх.

Однією з найбільших переваг Unity є його відмінна оптимізація, яка значно покращує якість та швидкість роботи проектів. Це особливо важливо для мобільних та VR-ігор, де продуктивність має ключове значення. Unity підтримує мови програмування такі як C# та UnityScript, що дозволяє розробникам використовувати знайомі їм технології.

Таким чином, Unity є потужним та універсальним інструментом для розробки ігор, що пропонує широкий спектр функціональних можливостей, простий інтерфейс та велику кількість додаткових матеріалів для успішної реалізації проектів.

Проаналізувавши переваги та недоліки трьох найбільш популярних рушіїв, для розробки гри був обраний ігровий рушій Unity, який надає широкі можливості для створення ігор різного рівня складності та жанрів. Обираючи Unity, отримуємо доступ до багатьох готових рішень для розробки ігор, а також можливість налаштовувати гру з урахуванням специфічних потреб проекту. Оскільки розробка гри включатиме як 3D, так і 2D графіку, Unity є оптимальним вибором.

Для розробки 3D гри важливо обрати відповідне середовище для 3D моделювання, яке відіграє ключову роль у створенні високоякісних моделей та текстур, що використовуються в грі. Таке середовище має надавати інструменти для створення та редагування 3D об'єктів, анімації, текстурування та рендерингу. Вибір правильного середовища для 3D моделювання може значно вплинути на загальну якість гри, продуктивність процесу розробки та здатність команди втілювати свої ідеї.

					КН.КР.Б – 122 – 24 – ПЗ	Арк.
						26
Змн.	Арк.	№ докум.	Підпис	Дата		

Autodesk Maya

Autodesk Maya – професійне програмне забезпечення для 3D моделювання та анімації, яке використовується у кіноіндустрії, телебаченні та ігровій розробці (рис. 2.4). Середовище пропонує широкий набір інструментів для створення складних 3D моделей, включаючи полігональне моделювання, NURBS моделювання та скульптинг. Також містить потужні інструменти для створення анімацій, включаючи інверсну кінематику, деформації та ригінг персонажів, що дозволяє створювати реалістичні рухи та взаємодії.

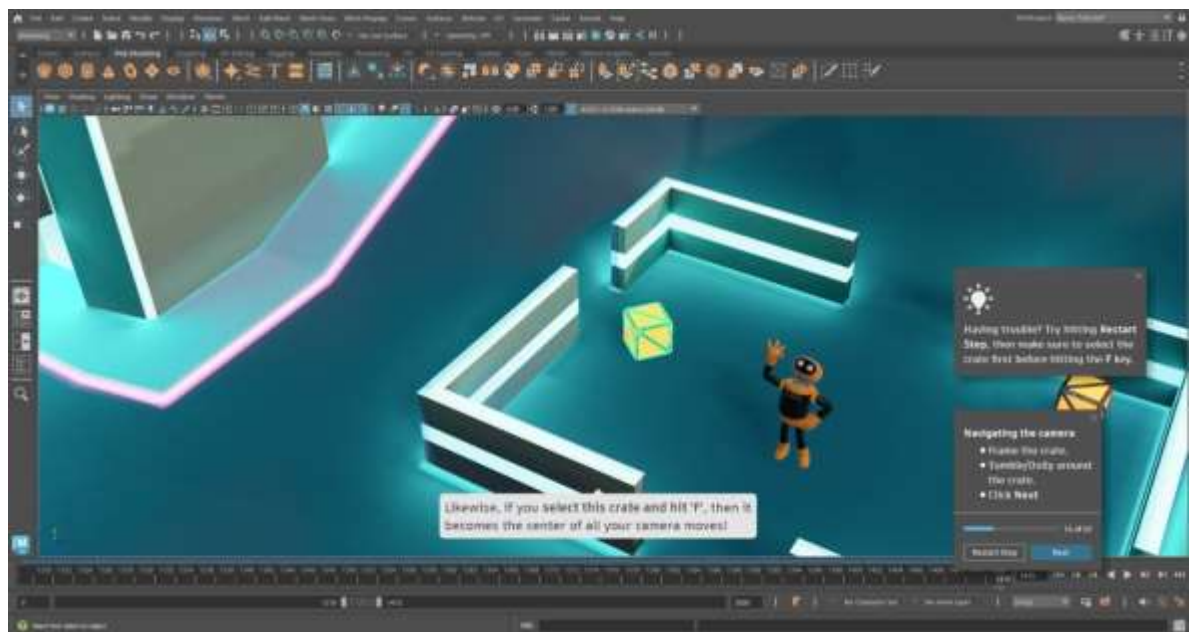


Рисунок 2.4 Середовище 3Д моделювання Maya

Maya використовує інтегровані інструменти для створення та застосування текстур, матеріалів та шейдерів, що забезпечує високу якість візуалізації об'єктів. Підтримка багатьох рендерингів дозволяє досягти фотореалістичної якості зображень.

Щодо інтерфейсу, варто відзначити, що структура інтерфейсу Maya сама по собі досить специфічна через велику кількість іконок і вкладок, які можуть збентежити не тільки початківців, але й досвідчених користувачів, які до цього працювали в інших програмах. Також середовище забезпечує зручну інтеграцію з різним програмним забезпеченням і спрощує робочий процес в цілому.

					КН.КР.Б – 122 – 24 – ПЗ	Арк.
						27
Змн.	Арк.	№ докум.	Підпис	Дата		

Проте, як і у всіх середовищах у даного також є недоліки. Основним недоліком є висока ціна, якщо для великих компаній купівля ліцензії Мауа може бути незначною витратою, проте для фрілансерів, особливо початківців 3D-художників, такі витрати є нерентабельними, тому їм варто розглянути більш доступні аналоги.

Autodesk 3ds Max

3Ds Max — це програмне забезпечення для 3D-моделювання, анімації та рендерингу, розроблене Autodesk, призначене для використання у сфері ігор та візуалізації дизайну (рис. 2.5). Воно входить до колекції мультимедійних та розважальних продуктів Autodesk, а також до колекції для архітектури, проектування та будівництва. Це програмне забезпечення є одним з інструментів для дизайну виробів та виробництва.

3Ds Max пропонує різні методи та потужні інструменти для створення та редагування анімацій. З його допомогою користувачі можуть створювати тривимірні комп'ютерні анімації та ефекти, які можна застосовувати у комп'ютерних іграх, фільмах, трансляціях, медичних ілюстраціях та судово-медичних презентаціях.



Рисунок 2.5 - Autodesk 3ds Max

Програмне забезпечення містить кілька контролерів анімації, що використовуються для зберігання ключових значень та процедурних

					КН.КР.Б – 122 – 24 – ПЗ	Арк.
						28
Змн.	Арк.	№ докум.	Підпис	Дата		

налаштувань, що дозволяє користувачам анімувати різні елементи. Крім того, 3Ds Max дозволяє зв'язувати об'єкти, формуючи ієрархії або ланцюжки для одночасної анімації наборів об'єктів, що спрощує процес.

Програма також має функції 3D-рендерінгу, включаючи можливість попереднього перегляду. Активуючи режим ActiveShade, користувачі можуть бачити ефекти змін освітлення та матеріалів у сцені в реальному часі.

3Ds Max також підтримує робочий процес, який дозволяє користувачам конвертувати сцени, змінюючи джерела світла, матеріали та об'єкти для використання з передовими технологіями рендерингу. Це дає змогу застосовувати різноманітні функції рендерингу, такі як нові фізичні матеріали та середовища, а також точні налаштування. Крім того, користувачі можуть гнучко перетворювати джерела світла, матеріали та об'єкти між різними рендеринг-двигунами, конвертуючи лише ті елементи, які їм потрібні[15].

Blender

Blender — це безкоштовне програмне забезпечення з відкритим вихідним кодом для 3D-моделювання, анімації, рендерингу та створення візуальних ефектів (рис. 2.6). Розроблене спільнотою Blender Foundation, ця програма є потужним інструментом для художників, дизайнерів та аніматорів.

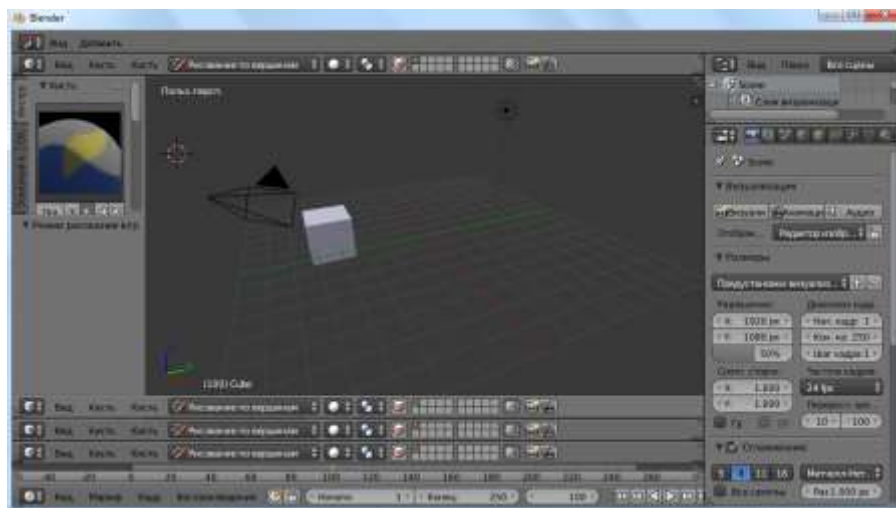


Рисунок 2.6 – Середовище моделювання Blender

Середовище підтримує багатий набір інструментів для моделювання, таких як скульптування, полігональне моделювання, ретопологія, текстурування

					КН.КР.Б – 122 – 24 – ПЗ	Арк.
						29
Змн.	Арк.	№ докум.	Підпис	Дата		

та UV-розгортання. Програма надає потужні інструменти для анімації, включаючи покадрову анімацію, анімацію з використанням кривих, рігінг (створення скелетних структур для моделей) та кінематику. Це дозволяє створювати реалістичні рухи персонажів та об'єктів.

Blender має вбудовані рендеринг-двигуни, такі як Eevee та Cycles, які підтримують трасування променів, глобальне освітлення та реалістичні тіні. Це дозволяє отримувати високоякісні зображення та анімації. Також підтримується композитинг та відеоредагування, що дозволяє створити візуальні ефекти та композити з використанням багатошарових зображень та відео[16].

Взявши до уваги функціональності та опис кожного з середовищ, для моделювання обрано програму Blender, яка є потужним інструментом для створення 3D моделей та анімацій.

Нижче наведено деякі ключові переваги Blender, які роблять його ідеальним вибором для проекту:

- Безкоштовний та відкритий доступ;
- Широкий набір інструментів;
- Потужний інструмент для скульптурингу;
- Інтегровані можливості рендерингу;
- Анімація та рігінг;
- Мультиплатформенність;
- Можливості інтеграції та експорту

Вибір Unity та Blender для розробки даної 3D гри забезпечує гнучкість, універсальність та потужні інструменти для створення високоякісного продукту.

2.2 Концепція та сюжет гри

Ідея гри "ZTURush" полягає в створенні захопливої та цікавої пригоди, де гравець уособлює студента, який опинився в складній ситуації: його хочуть відрахувати з університету через низку пропусків та неправильну поведінку.

					КН.КР.Б – 122 – 24 – ПЗ	Арк.
						30
Змн.	Арк.	№ докум.	Підпис	Дата		

Гравець має протистояти цьому вироку, збираючи зірочки після успішного виконання завдань від викладачів, збору предметів по кампусу та уникнення проректора, який стежить за ним.

Дія гри відбувається у Державному університеті «Житомирська політехніка», реальному університеті у місті Житомирі, а саме на першому поверсі навчального закладу. Студент, згадавши про початок сесії, прибуває до університету, проте, боргів уже й не злічити, йому світить відрахування, але випадає останній шанс усе виправити.

Перший поверх кампусу університету включає в себе різні локації, такі як аудиторії, кабінети викладачів, спортзали, копіцентр та кав'ярню на вході. Університетський кампус відображає атмосферу навчання, взаємодії з різними соціальними групами та конкуренцію в умовах високого навчального та студентського життя. Головний персонаж змушений боротися зі своїми проблемами, знаходячи баланс між навчанням, виконанням завдань від викладачів та уникненням відрахування від проректора.

Гравець починає гру зі студентом, який знаходиться в центрі університетського комплексу. Його завданнями є знаходження та взаємодія з NPC викладачами, які допомагають йому з кожним завданням, якісно підготувавши студента до іспитів та забезпечуючи його розвиток в академічній сфері. Кожне завдання вимагає від нього вміння збирати необхідні предмети по університетському майданчику, виконувати різноманітні завдання від викладачів та уникати зустрічі з проректором, який стежить за ним і намагається впіймати на "гарячому".

Головна мета гри полягає в тому, щоб студент успішно закінчив сесію, збравши всі необхідні предмети та виконавши всі завдання, тим самим уникнувши відрахування та вигравши гру.

У грі "ZTURush" особливість полягає в тому, що світ, в якому діє головний персонаж, створений на основі реального університету "Житомирська політехніка", він відображений у 3D проекції. Це дозволяє гравцям віртуально

					КН.КР.Б – 122 – 24 – ПЗ	Арк.
						31
Змн.	Арк.	№ докум.	Підпис	Дата		

побувати в даному навчальному закладі, навіть якщо вони там ще не були, або мають намір вступати в цей університет.

Гра виглядає у стилі low-poly, що означає, що всі моделі мають низьку полігонність і не дуже деталізовані. Діалоги у грі виконані у стилі коміксу з чіткими лініями та яскравими характерними рисами. Увесь візуальний стиль гри оснований на яскравих кольорах, які відтворюють основні кольори університету, що дозволяє передати атмосферу справжнього студентського життя. Виразні риси обличчя, зачіски та одягу персонажів роблять їх впізнаваними та дозволяють зрозуміти та ідентифікувати даного героя у реальному світі.

2.3 Ігрові механіки

Ігрові механіки визначають способи взаємодії гравця з віртуальним світом гри, включаючи рух, взаємодію з оточенням, виконання завдань, розв'язання головоломок та інше. Розглянемо основні ігрові механіки для гри "ZTURush":

- Збір предметів: Гравець має збирати різні предмети, які розкидані по університетському майданчику. Ці предмети необхідні для виконання завдання, при зборі усіх предметів зараховується як виконане одне завдання. Збір предметів включає і себе пошук конкретних предметів, що вимагає від гравця дослідження різних локацій.
- Взаємодія з NPC: Гравець має взаємодіяти з NPC викладачами, які надають йому завдання. Взаємодія включає в себе розмови та виконання завдань.
- Унакнення ворогів: Гравець має уникати зустрічі з проректором, який стежить за ним і намагається впіймати на «гарячому».
- Виконання завдань: Гравець повинен виконувати різноманітні завдання від викладачів, що включають в себе вирішення головоломок. Успішне виконання завдань надає гравцю зірочки, які наближають гравця до перемоги.

					КН.КР.Б – 122 – 24 – ПЗ	Арк.
						32
Змн.	Арк.	№ докум.	Підпис	Дата		

- Прогресія гри: Головна мета гри полягає в тому, щоб студент успішно закінчив сесію, зібравши всі необхідні предмети та виконавши всі завдання. Це вимагає від гравця стратегічного планування та ефективного використання ресурсів. Успішне досягнення цієї мети веде до виграшу.

Ці механіки разом створюють захоплюючий геймплей, де гравці мають балансувати між виконанням завдань, збором предметів та уникненням ворогів. Кожна дія має наслідки, що впливають на кінцевий результат гри.

2.4 Діаграми та алгоритми роботи

Для наочної демонстрації варіантів використання ігрової системи, були розроблені діаграми сценаріїв використання, представлені нижче (рис. 2.7). На цій діаграмі зображена проектована система, яка складається з безлічі сутностей або акторів, що взаємодіють із системою за допомогою так званих варіантів використання.

Дана гра має одну дійову особу – гравця. На діаграмі нижче зображено, що гравець може запускати гру, з'являється початкове меню з опціями, такими як почати гру, переглянути налаштування гучності, інформація про гру та вихід. У грі він зустрічає NPC викладачів і отримує від них завдання, а також знаходить та збирає предмети, необхідні для виконання завдання. При правильній відповіді гравець отримує зірочку, при неправильній втрачає життя, якщо отримано усі зірки, то гра автоматично завершується і перенесить до головного меню.

У геймплеї гравець також зустрічає проректора, котрий віднімає життя персонажа, якщо не залишилося життів, гра завершується із поразкою. Під час ігрової сесії користувач може натиснути на кнопку паузи, де має можливість змінити гучність, перейти до меню або відновити гру.

					КН.КР.Б – 122 – 24 – ПЗ	Арк.
						33
Змн.	Арк.	№ докум.	Підпис	Дата		

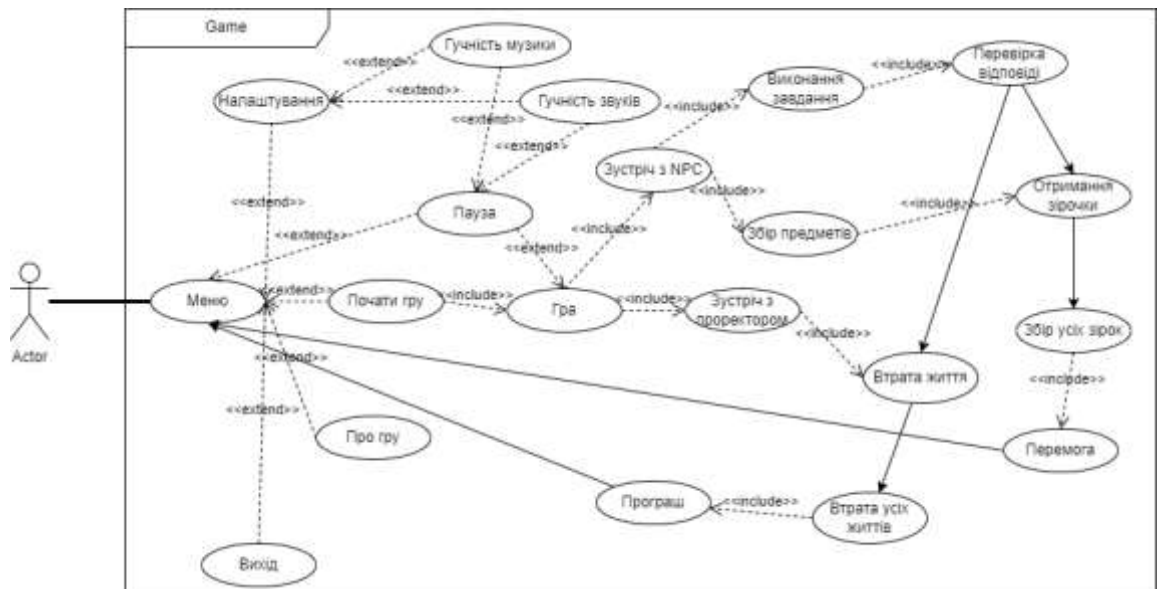


Рисунок 2.7 – Діаграма варіантів використання

Щоб зрозуміло відобразити функціональну структуру ігрової системи, можна скористатися IDEF0-діаграмою. Спочатку розробляється опис системи в цілому та її взаємодія з навколишнім світом (контекстна діаграма) (рис. 2.8). Після цього система декомпозується на підсистеми, кожна з яких детально описується в окремій діаграмі декомпозиції. Цей процес повторюється, доки не буде досягнута необхідна міра деталізації[17].



Рисунок 2.8 - IDEF0 ігрової системи

Згідно методології, на контекстній діаграмі присутня вхідна інформація, а саме «Зацікавленість проходження гри», відповідно на виході результатом процесу буде «Результат гри».

Керування процесом відбувається «Вимогами гравця» та «Сюжетом гри», а механізмами виконання виступатимуть «Програмне забезпечення» та «Користувач».

Наступний кроком розглядається декомпозиція контекстної діаграми для детального розгляду системи, що наведено нижче (рис. 2.9).

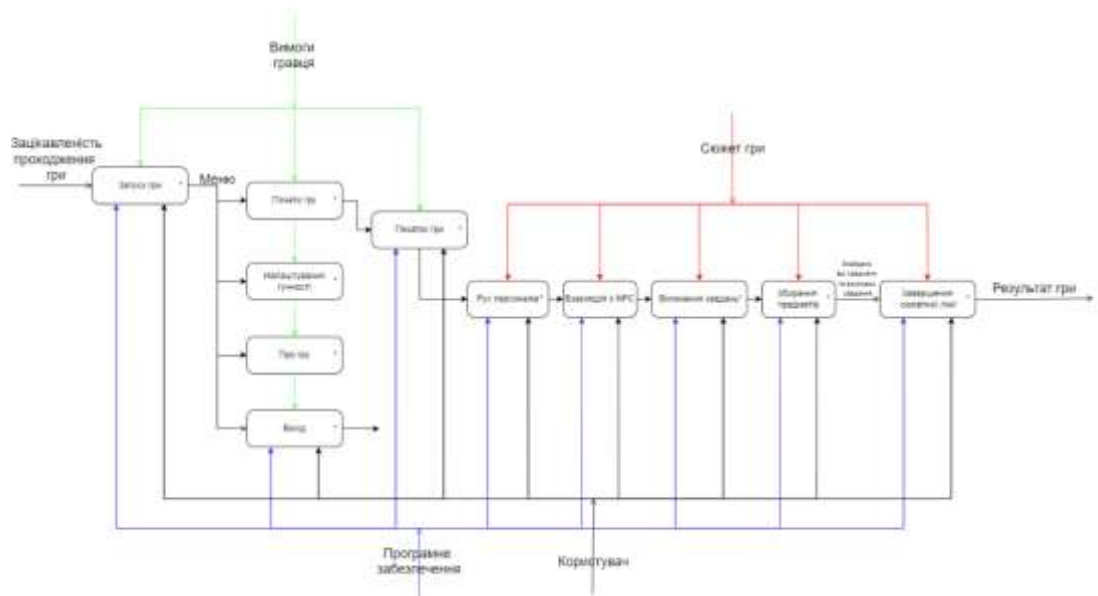


Рисунок 2.9 – Декомпозиція ігрової системи

Таким чином, в результаті проведеного структурно-функціонального аналізу на основі методології IDEF0 було визначено основні процеси, що проходять у додатку. Процес аналізу включав у себе такі етапи:

- Ідентифікація основних процесів;
- Визначення механізмів управління та контролю;
- Розпис вхідних даних;
- Отримувані результати;

Цей аналіз дозволив створити чітку та зрозумілу картину функціонування ігрової системи, що допомагає у подальшій розробці та оптимізації додатку.

Діаграма станів (state diagram) - це графічний інструмент для моделювання поведінки системи, що показує різні стани, в яких може перебувати система, та переходи між ними. Вона ілюструє, як система реагує на зміну вхідних даних, дії користувачів чи інші фактори.

Діаграми станів використовуються у проектуванні програмного забезпечення для моделювання поведінки програми в різних ситуаціях, таких як старт, пауза, завершення роботи, помилки тощо. Вони допомагають розуміти, як система функціонує в різних умовах, що полегшує розробку та тестування програмного забезпечення для програмістів та аналітиків[18].

Таким чином, побудова діаграм станів сприяє візуалізації та розумінню поведінки системи, а також виявленню можливих проблем у її функціонуванні. Щоб створити діаграму станів, потрібно спочатку визначити всі можливі стани системи, потім визначити події або вхідні дані, що спричиняють перехід між цими станами, і нарешті, зобразити стани та переходи на графіці.

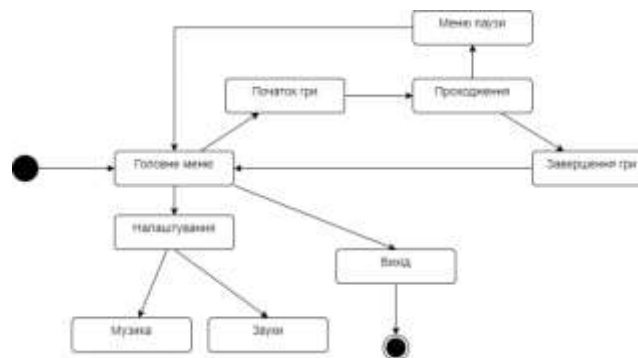


Рисунок 2.10 – Загальна діаграма станів та переходів гри

На загальній діаграмі станів (рис. 2.10) відображено стани переходів користувача по етапах гри. Відвідування головного меню, перехід до проходження гри, пауза під час гри та завершення гри. Після запуску гри, гравець проходить рівень, взаємодіє з NPC та виконує завдання. Під час проходження гри гравець може натиснути паузу. Якщо гравець втрачає всі життя або завершує всі завдання, гра завершується, і відображається результат.



Рисунок 2.11 – Діаграма станів та переходів елементу проходження гри

Діаграма станів для елемента "Проходження" гри "ZTURush" (рис. 2.11) відображає послідовність станів та переходів, які можуть відбуватися під час гри. Гравець починає гру, потрапляючи в початковий стан "Початок гри". Потім він переходить у стан "Проходження", де активно взаємодіє з грою, виконуючи різноманітні дії. У процесі проходження гравець може виконати завдання ("Виконання завдання") та зібрати всі необхідні предмети ("Зібрано всі предмети"). Якщо він успішно виконує завдання, він отримує зірку ("Отримано зірку"). Гравець може також викликати паузу гри ("Меню паузи"), відновити гру після паузи ("Відновлення гри") або завершити гру ("Завершення гри"), перейшовши до головного меню або вийшовши з гри.

Діаграма діяльностей є графічним засобом моделювання, що відображає послідовність дій у процесі виконання діяльності. Вона допомагає представити логіку процесу або алгоритму, розбиваючи складні процеси на прості кроки та ілюструючи їх послідовність. Діаграма містить елементи, такі як дії, рішення, розвилки, злиття та переходи між ними, що відображають кроки, умови, вибори та переходи у процесі виконання.

Щоб побудувати діаграму діяльностей, необхідно спочатку визначити всі можливі дії та їх послідовність. Потім ці дії зображуються на графіці та з'єднуються лініями для показу послідовності. Крім того, додаються умови та розгалуження, які вказують на можливі зміни шляху в процесі залежно від умов або виборів.

На даній діаграмі зображено процес геймплею (рис. 2.12). Пошук NPC викладачів, виконання завдань та збирання предметів, а у разі виявлення проректора втрата життя.

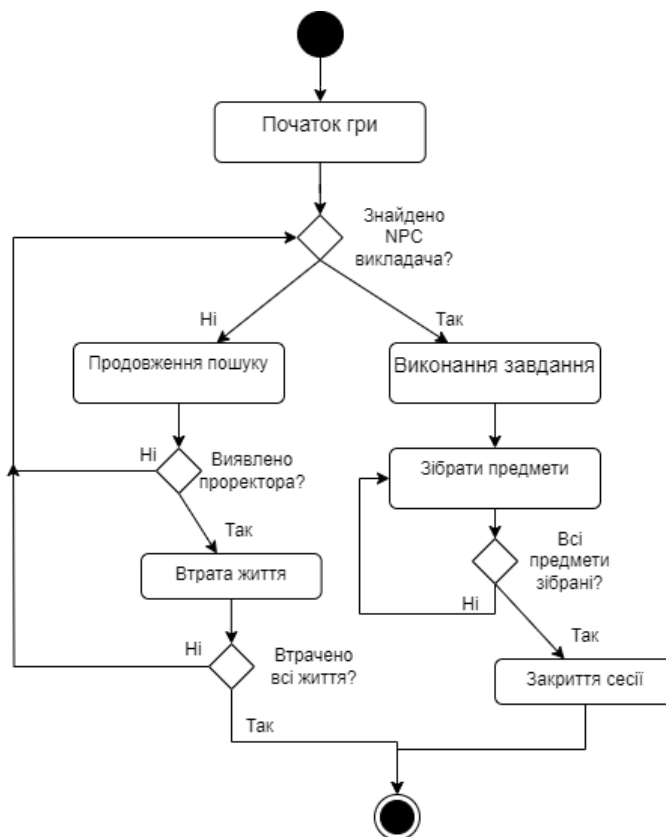


Рисунок 2.12 – Діаграма діяльності

Отже, діаграма діяльностей створює зрозуміле візуальне представлення послідовності дій та рішень у процесі виконання конкретної діяльності чи процесу.

Створення, огляд та аналіз діаграм грають ключову роль у визначенні алгоритмів, необхідних для ефективної роботи ігрового додатку. Перш за все, діаграми дозволяють визначити функціональність гри, включаючи можливості, функції та взаємодію з гравцем. Наприклад, діаграма активності може показати послідовність дій гравця, що допомагає визначити алгоритми для їх обробки та взаємодії з гравцем.

Далі, аналіз діаграм класів та об'єктів дозволяє визначити структуру персонажів, об'єктів та їх взаємодію. На основі цього можна розробити алгоритми для руху персонажів, взаємодії з об'єктами та інші логічні операції.

Також, діаграми станів або діаграми класів допомагають визначити логіку гри та управління. Вони вказують на різні стани гри та переходи між ними, що допомагає розробити алгоритми для керування цими станами та переходами.

Не останнє місце у важливості займає аналіз діаграм для оптимізації та підвищення ефективності гри. Виявлення повторюваних дій або неефективних процесів може вказати на потребу в удосконаленні алгоритмів для оптимізації роботи гри та покращення досвіду користувачів.

Отже, створення, аналіз та огляд діаграм грає важливу роль у визначенні необхідних алгоритмів для роботи ігрового додатку, що в свою чергу впливає на його ефективність, функціональність та загальний успіх серед користувачів.

На основі вище поданого можна визначити потрібні алгоритми.

Алгоритми керування персонажем

Алгоритми керування персонажем відповідають за переміщення головного героя по ігровому світу. Вони включають обробку введення з клавіатури або контролера та забезпечують плавність руху та анімації. Основні етапи алгоритму:

- Зчитування введення від гравця (натискання клавіш або кнопок контролера).
- Визначення напрямку та швидкості руху персонажа.
- Оновлення позиції персонажа на основі вхідних даних.
- Анімація руху відповідно до напрямку та швидкості.
- Обробка зіткнень з об'єктами та стінами.

Алгоритми взаємодії з NPC

Ці алгоритми забезпечують взаємодію головного героя з неігровими персонажами (NPC). Основні етапи алгоритму:

- Визначення положення персонажа відносно NPC.
- Виявлення близькості для початку взаємодії.
- Запуск діалогового інтерфейсу.
- Обробка вибору гравця в діалоговому вікні.

Алгоритми збирання предметів

Ці алгоритми відповідають за механіку збирання предметів, розкиданих по університету. Основні етапи алгоритму:

- Визначення положення персонажа відносно предметів.
- Виявлення близькості для початку взаємодії.
- Оновлення інвентаря гравця при успішному збиранні предмета.
- Видалення зібраного предмета з ігрового світу.

Алгоритми перевірки виконання завдань

Дані алгоритми забезпечують перевірку виконання гравцем отриманих завдань. Основні етапи алгоритму:

- Отримання завдання від NPC.
- Відстеження дій гравця, пов'язаних з виконанням завдання.
- Перевірка виконання умов завдання.
- Оновлення статусу завдання (виконано/невиконано).
- Нагородження гравця за успішне виконання завдання (зірочки, бонуси тощо).

Алгоритми втрати життів

Ці алгоритми відповідають за втрату життів головним персонажем. Основні етапи алгоритму:

- Відстеження дій гравця та проректора.
- Визначення ситуацій, коли проректор ловить гравця або гравець неправильно виконує завдання.
- Зменшення кількості життів при відповідних подіях.
- Відображення поточної кількості життів на інтерфейсі.
- Обробка завершення гри при втраті всіх життів (відрахування).

Структури даних для зберігання інформації про гравця, завдання, предмети та NPC

					КН.КР.Б – 122 – 24 – ПЗ	Арк.
						40
Змн.	Арк.	№ докум.	Підпис	Дата		

Для ефективного зберігання та обробки даних у грі використовуються структури даних, що включають:

- Гравець: містить інформацію про поточний статус гравця (кількість життів, інвентар, статус виконання завдань).
- Завдання: зберігає опис завдання, умови його виконання, статус виконання, нагороди.
- Предмети: містить інформацію про предмети, їхнє місцезнаходження та стан (зібраний/незібраний).
- NPC: включає дані про неігрових персонажів, їхнє місцезнаходження, діалоги та дії.

Ці структури даних дозволяють ефективно керувати ігровим процесом та забезпечувати динамічну взаємодію між гравцем та ігровим світом.

Висновки до другого розділу

Отже, були розглянуті ключові аспекти розробки гри "ZTURush". Було визначено, що для розробки гри використовуватиметься Unity як основне середовище розробки. Це рішення обґрунтоване його багатofункціональністю, широким спектром можливостей для створення 3D-ігор та підтримкою кросплатформеності. Для створення 3D-об'єктів обрано Blender, оскільки він є потужним та безкоштовним інструментом для 3D-моделювання. Також розроблено концепцію, основний сюжет гри "ZTURush" та описано основні ігрові механіки. Представлено діаграми та алгоритми, які описують основні процеси гри та логіку.

Підсумовуючи, у розділі було визначено ключові елементи проектування гри. Ці напрацювання створюють міцну основу для подальшого етапу — реалізації гри "ZTURush".

					КН.КР.Б – 122 – 24 – ПЗ	Арк.
						41
Змн.	Арк.	№ докум.	Підпис	Дата		

РОЗДІЛ 3 РЕАЛІЗАЦІЯ

3.1 Розробка ігрової сцени та моделей

У 3D-модельюванні об'єкти створюються з використанням різноманітних форм і геометричних елементів, таких як куби, сфери, циліндри тощо (рис. 3.1). Ці об'єкти можна змінювати за розміром, формою та положенням. Крім того, можна додавати різні текстури та кольори, щоб надати об'єктам реалістичного вигляду.

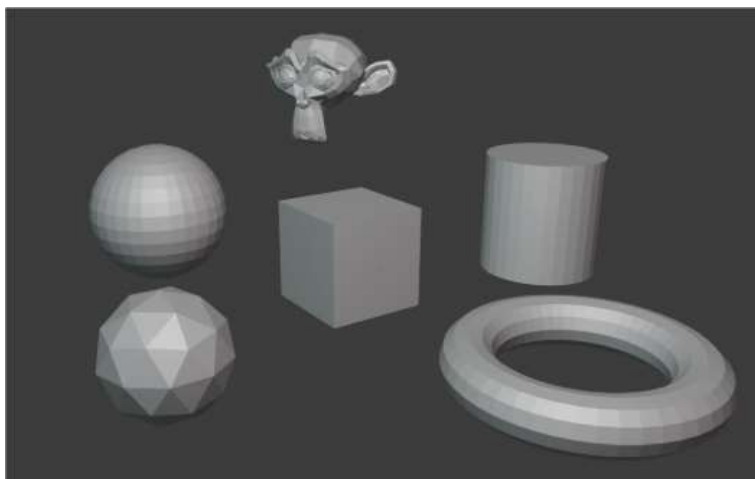


Рисунок 3.1 – Примітивні об'єкти Blender

Для пригодницької квест-гри було обрано тематику університету, на основі реального навчального закладу. Тому потрібно було створити такі елементи як: будівля, об'єкти інтер'єру в залежності від частини локації та елементи декору.

Для створення будівлі використовуються примітивні куби, для кожної частини стін. За допомогою модифікатора змінюємо розміри кубів відповідно до потрібних параметрів відносно схеми будівлі. З використанням модифікатора Boolean створюємо отвори у потрібних стінах для вікон. Для додавання вікон потрібно знайти в додатках засіб Archimesh, він спеціально розроблений для генерування елементів архітектури. Підбравши відповідні розміри до всіх елементів та додавши Plane в якості підлоги, додаємо матеріали до елементів. У результаті отримуємо основу для локації (рис. 3.2).

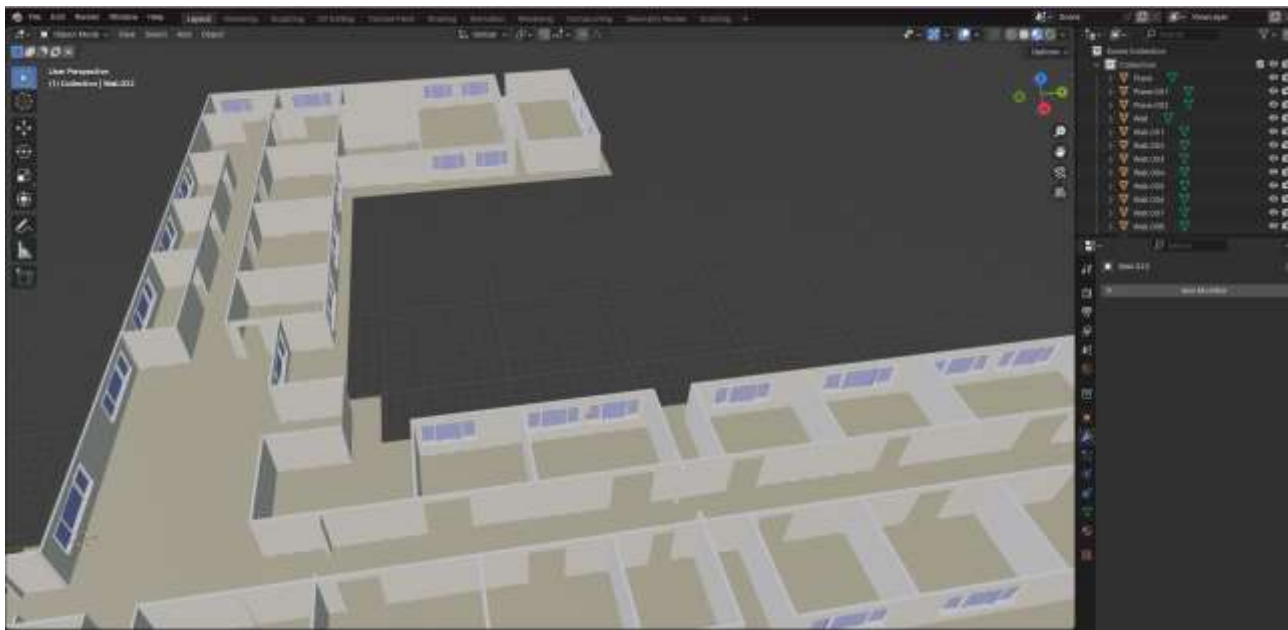


Рисунок 3.2 – Створена локація університету

З примітивів створюємо деякі елементи інтер'єру, яких не має у Unity Asset Store, до прикладу предмети кав'ярні (рис. 3.3 (а)), баскетбольні кільця у спортзалі (рис. 3.3 (б)), деякі шафи, столи та інші об'єкти.

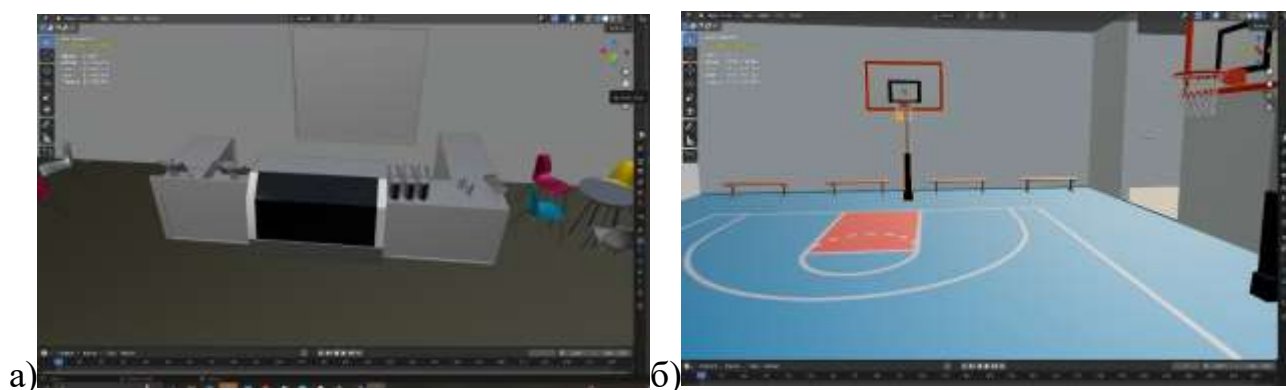


Рисунок 3.3 – Створені елементи інтер'єру: а) предмети кав'ярні; б) баскетбольні кільця

Для інших елементів інтер'єру та декору було проведено пошук оптимальних асетів у магазині Unity[29]. Розглянуто виключно безкоштовні варіанти, які підходили б по тематиці та стилю гри. Зокрема, завантажено асети Office Pack – Free, Low Poly Office Props – LITE, School assets, low poly WC та Melon's Low Poly Office, які включають в себе об'єкти меблів, елементи декору та деталі, такі як комп'ютери, ноутбуки телефони та книги (рис. 3.4).

					КН.КР.Б – 122 – 24 – ПЗ	Арк.
						43
Змн.	Арк.	№ докум.	Підпис	Дата		

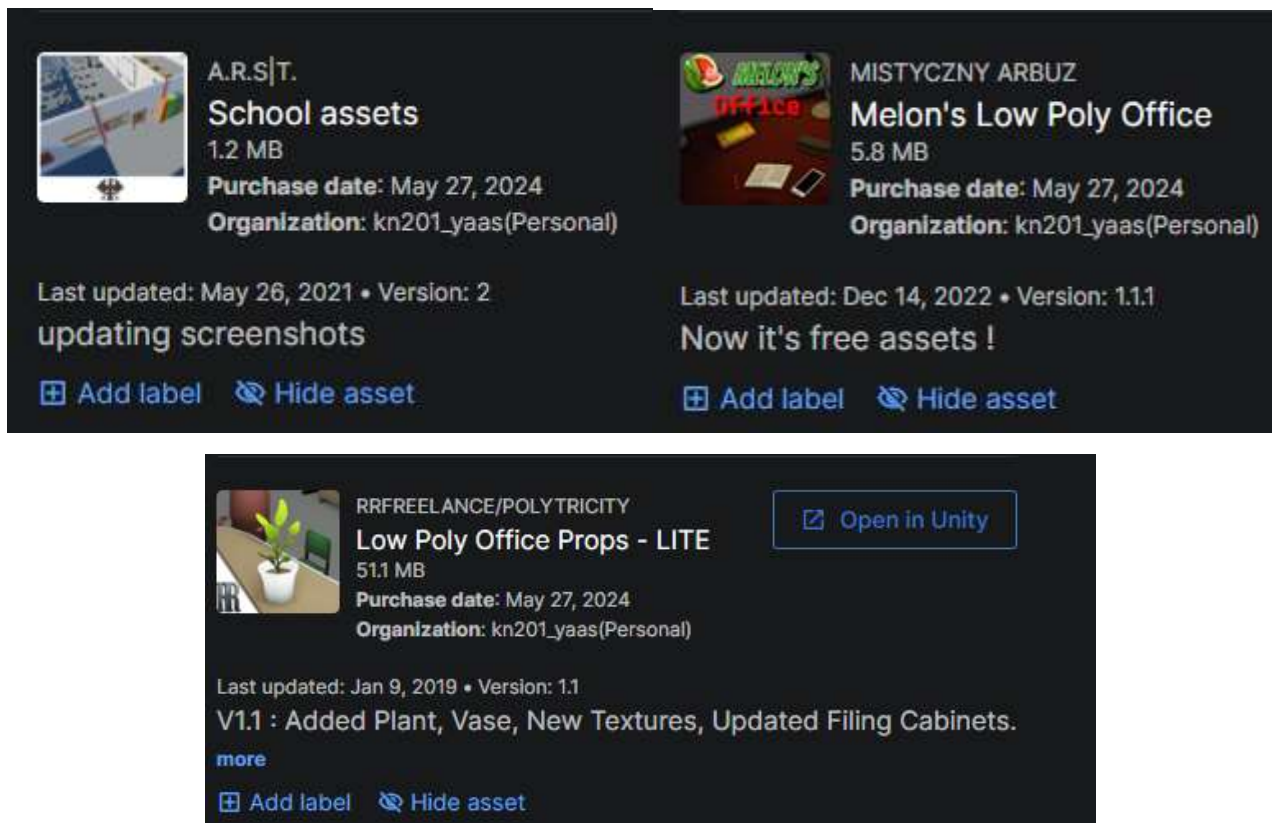


Рисунок 3.4 – Набори асетів

Після створення повноцінної локації, розпочато моделювання персонажів гри, а саме: студент, проректор університету, а також NPC викладачів.

Для прискорення моделювання було створено основу голови, рук і ніг (рис. 3.5). Ці частини модифікувалися за допомогою кистей для скульптингу під кожного персонажа, щоб надати їм унікальних рис та деталей.

Також було розроблено загальну основу тіла, яка використовувалася для створення одягу для героїв. Після моделювання одяг з'єднувався полігонами з відповідними частинами тіла персонажів, щоб забезпечити правильну посадку і природний вигляд.

Після завершення моделювання було проведено перевірку нормалей в програмі Autodesk 3ds Max. У разі необхідності, нормалі були перевернуті для правильного відображення і забезпечення коректної роботи з освітленням та тінями.

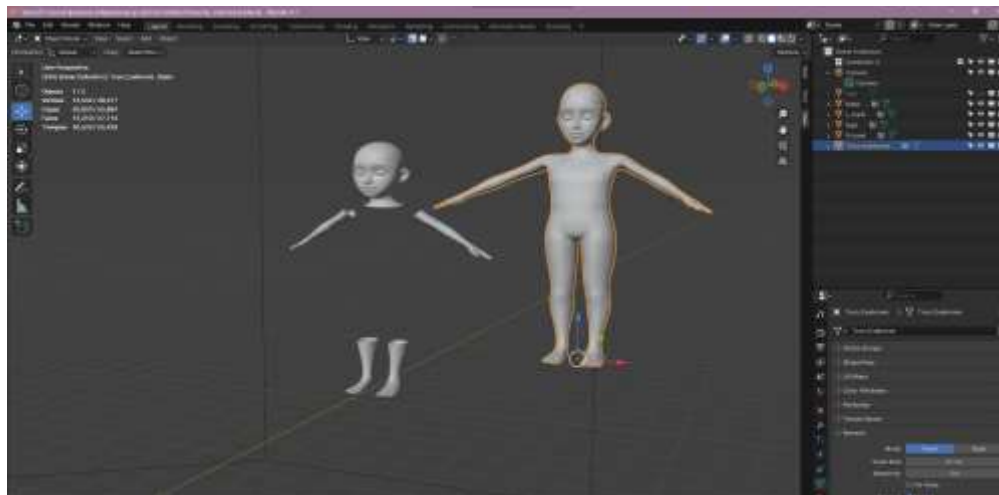


Рисунок 3.5 – Основа для моделювання персонажів

Після завершення моделювання та перевірки було створено базову текстуру та UV-розгортку. За допомогою функцій у Blender було виконано розгортку, зручну для розмальовування текстури (рис. 3.6). Після завершення цього етапу текстура автоматично накладалася на персонажа.



Рисунок 3.6 – Розгортка та текстура персонажа

Коли модель була повністю завершена, її було експортовано у форматі FBX для подальшого використання в інших програмах або ігрових рушіях.

Для всіх персонажів була проведена аналогічна робота, що дозволило створити п'ять моделей героїв, кожна з яких готова до подальшого використання (рис. 3.7-3.11).



Рисунок 3.7-3.11 – Готові моделі персонажів

Для подальшої роботи з моделями необхідно додати їм базові анімації. Для цього можна скористатися сайтом Mixamo. Цей сервіс дозволяє завантажити власну модель, додати їй кістки (рис. 3.12), вибрати анімації та налаштувати їх відповідно до потреб (рис. 3.13).

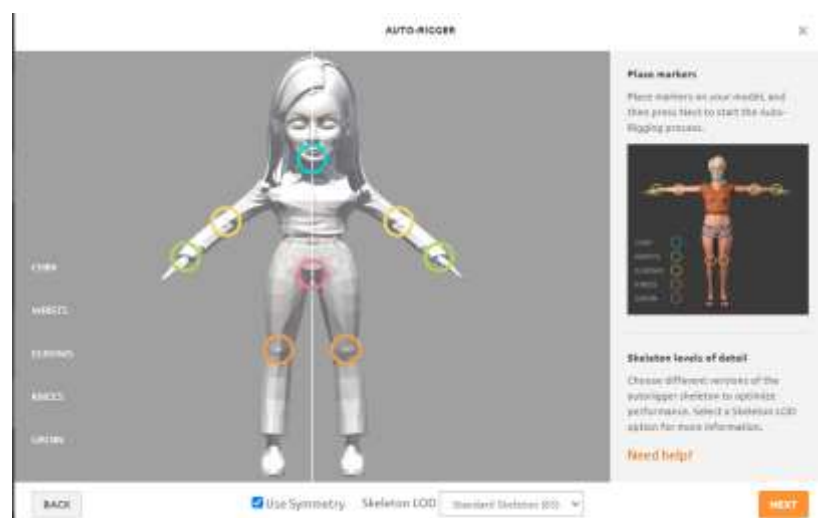


Рисунок 3.12 – Автоматичне встроювання кісток

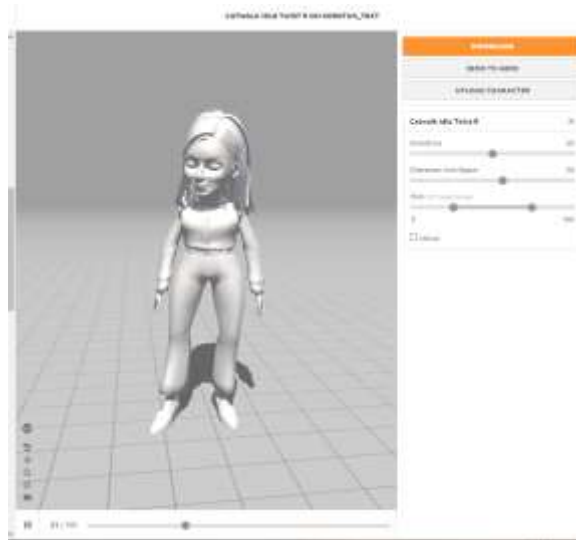


Рисунок 3.13 – Анімація персонажа

Після налаштування і завантаження анімацій, ми імпортуємо їх у Unity, встановлюючи для моделі тип анімації Humanoid і створюючи один аватар для всіх анімацій. Аніматор для NPC персонажів має простий вигляд без параметрів і складних структур (рис. 3.14).



Рисунок 3.14 – Зразок Аніматора для NPC

3.2 Розробка інтерфейсу гри

Проектування інтерфейсу користувача для гри "ZTURush" - це критичний етап розробки, що визначає зручність, ефективність та привабливість взаємодії гравця з грою. Успішний дизайн інтерфейсу не лише полегшує користування грою, а й створює позитивний враження та підвищує загальний рівень задоволення від геймплею.

При запуску гри, першим екраном, на який потрапляє користувач, є меню (рис. 3.15). В цьому меню був використаний фон, який не відволікає увагу від

					КН.КР.Б – 122 – 24 – ПЗ	Арк.
						47
Змн.	Арк.	№ докум.	Підпис	Дата		

основних компонентів, таких як кнопки та інші елементи. Фон спрямований на збереження уваги гравця на важливих функціях меню, не відволікаючи його від них.



Рисунок 3.15 – Інтерфейс основного меню

Основні кнопки у цьому меню забезпечують зручний доступ до основних функцій гри:

- Налаштування: Ця кнопка дозволяє користувачу налаштувати параметри гучності звуку та музики, забезпечуючи індивідуалізований досвід гри (рис. 3.16).



Рисунок 3.16 – Налаштування гучності

- Інформація про гру: Натискання на цю кнопку дозволяє гравцеві дізнатися детальнішу інформацію про гру (рис. 3.17).



Рисунок 3.17 – Інформація про гру

- Розпочати гру (рис. 3.18): Ця кнопка веде до початку гри (рис. 3.20).

```
0 references
public void PlayGame()
{
    SceneManager.LoadScene(1);
}
```

Рисунок 3.18 – Код завантаження сцени гри

- Вихід: Кнопка "EXIT" дозволяє користувачу вийти з гри та повернутися до робочого столу або іншого місця, де відкрито гру (рис. 3.19).

```
0 references
public void QuitGame()
{
    Debug.Log("Вихід");
    Application.Quit();
}
```

Рисунок 3.19 – Код виходу з гри



Рисунок 3.20 – Інтерфейс гри



Рисунок 3.21 – Меню паузи

Основними елементами Меню паузи (рис. 3.21) є регулювання гучності звуків та музики, що буде розглянуто пізніше, а також кнопка «Відновлення гри» та кнопка «Меню».

При натисненні на кнопку паузи, час гри встановлюється на нуль, що заморожує її та відображається Меню паузи (рис. 3.22).

```
0 references
public void Pause()
{
    pauseMenu.SetActive(true);
    Time.timeScale = 0;
}
```

Рисунок 3.22 – Виклик Меню паузи

Кнопка «Меню» завантажує сцену Головного меню та встановлює час на одиницю, тим самим відновлює гру (рис. 3.23).

```
0 references
public void Home()
{
    SceneManager.LoadScene(0);
    Time.timeScale = 1;
}
```

Рисунок 3.23 – Завантаження сцени Головного меню

Кнопка «Відновлення гри» приховує Меню паузи та встановлює час рівний одиниці, що відновлює хід гри з того місця, де було перервано (рис. 3.24).

					КН.КР.Б – 122 – 24 – ПЗ	Арк.
						50
Змн.	Арк.	№ докум.	Підпис	Дата		

```

0 references
public void Resume()
{
    pauseMenu.SetActive(false);
    Time.timeScale = 1;
}

```

Рисунок 3.24 – Відновлення гри

3.3 Реалізація геймплею

3.3.1 Керування персонажем

Першим кроком перед розробкою алгоритму є створення Blend Tree в Аніматорі. Це дозволяє плавно змінювати анімацію персонажа від стану бездіяльності (Idle) до бігу (Run) в залежності від його швидкості.

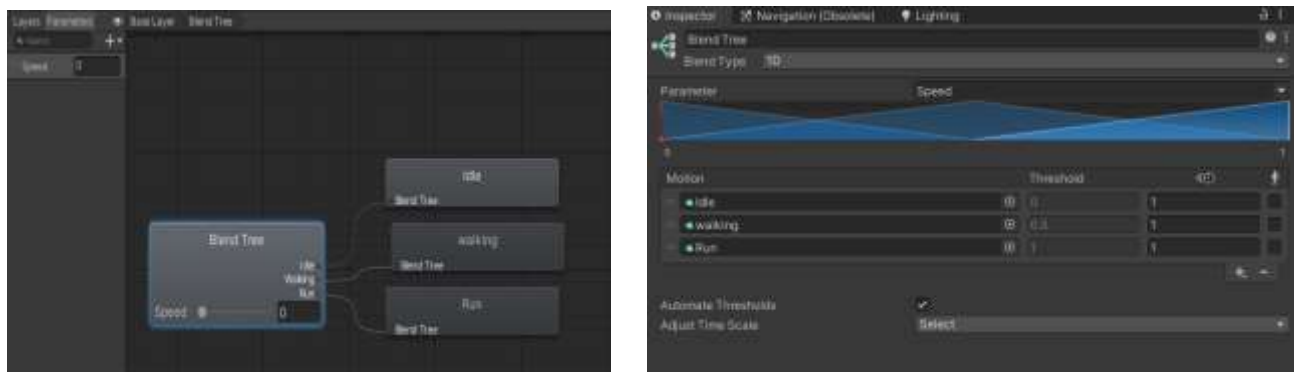


Рисунок 3.25 – Анімація Студента

Для оптимізації процесу зіткнень ми прикріплюємо до персонажа компонент Capsule Collider. Цей компонент визначає невидимі "кордони" об'єкта. Для базової фізики гравця додаємо компонент Rigidbody. Після виконання цих кроків ми отримуємо гравця з мінімальним набором необхідних фізичних компонентів, готового до програмування.

Основою для управління персонажем став Kinematic Character Controller. Це високоефективне рішення для контролю за персонажами, яке не обмежується конкретним ігровим жанром і легко інтегрується в будь-який проект або архітектуру. За допомогою Kinematic Character Controller було створено Player Controller, що відповідає за реалізацію управління персонажем в грі.

На початку визначаємо структуру для вхідних даних персонажа. У контролері підключаємо `KinematicCharacterMotor`, для керування рухом персонажа. Встановлюються параметри руху, такі як максимальна стабільна швидкість, швидкість прискорення до максимальної швидкості, швидкість повороту та метод орієнтації персонажа. Підключаємо аніматор для керування анімаціями героя.

У методі “`Awake`” даний контролер подається в `KinematicCharacterMotor`.

Метод “`SetInputs`” встановлює вхідні дані для керування персонажем, а саме приймає структуру “`PlayerCharacterInputs`”, що містить осі руху та орієнтацію камери. Створюється вектор руху на основі вхідних осей “`MoveAxisRight`” та “`MoveAxisForward`”, де “`ClampMagnitude`” обмежує довжину вектора до 1, щоб запобігти надто швидким або неконтрольованим рухам. Далі проєктується напрямок вперед від камери на площину по горизонталі, а отриманий вектор нормалізується. Якщо камера дивиться прямо вгору або вниз, то використовується напрямок вгору від камери. Також створюємо змінну повороту для обертання персонажа у бік камери по горизонталі.

Вектор руху повертається у напрямку, заданому ротацією камери. В залежності від метода орієнтації, персонаж буде дивитися в напрямку камери або в напрямку руху.

```

reference
public void SetInputs(ref PlayerCharacterInputs inputs)
{
    // Clamp input
    Vector3 moveInputVector = Vector3.ClampMagnitude(new Vector3(inputs.MoveAxisRight, 0f, inputs.MoveAxisForward), 1f);

    // Calculate camera direction and rotation on the character plane
    Vector3 cameraPlanarDirection = Vector3.ProjectOnPlane(inputs.CameraRotation * Vector3.forward, Motor.CharacterUp).normalized;
    if (cameraPlanarDirection.sqrMagnitude == 0f)
    {
        cameraPlanarDirection = Vector3.ProjectOnPlane(inputs.CameraRotation * Vector3.up, Motor.CharacterUp).normalized;
    }
    Quaternion cameraPlanarRotation = Quaternion.LookRotation(cameraPlanarDirection, Motor.CharacterUp);

    // Move and look inputs
    _moveInputVector = cameraPlanarRotation * moveInputVector;

    switch (OrientationMethod)
    {
        case OrientationMethod.TowardsCamera:
            _lookInputVector = cameraPlanarDirection;
            break;
        case OrientationMethod.TowardsMovement:
            _lookInputVector = _moveInputVector.normalized;
            break;
    }
}

```

Рисунок 3.26 – Зчитування введення

Метод “UpdateRotation” відповідає за обчислення та встановлення обертання персонажа в кожен кадр. Він визначає, в який бік повинен бути повернутий персонаж, на основі вхідних даних.

```

// References
public void UpdateRotation(ref Quaternion currentRotation, float deltaTime)
{
    if (_lookInputVector.sqrMagnitude > 0f && OrientationSharpness > 0f)
    {
        // Smoothly interpolate from current to target look direction
        Vector3 smoothedLookInputDirection = Vector3.Slerp(Motor.CharacterForward, _lookInputVector, 1 - Mathf.Exp(-OrientationSharpness * deltaTime)).normalized;

        // Set the current rotation (which will be used by the KinematicCharacterMotor)
        currentRotation = Quaternion.LookRotation(smoothedLookInputDirection, Motor.CharacterUp);
    }
}

```

Рисунок 3.27 – Встановлення повороту персонажа

Метод “UpdateVelocity” відповідає за обчислення та встановлення швидкості персонажа в кожен кадр. Він визначає, як персонаж повинен рухатися, враховуючи вхідні дані та стан землі. Перевіряється поточна швидкість та натискання клавіші Shift, якщо не натиснута клавіша, то поточна швидкість зменшується вдвічі та програється анімація ходьби, якщо натиснута клавіша, то програється анімація бігу, у разі нульової швидкості виконується анімація Idle.

```

// References
public void UpdateVelocity(ref Vector3 currentVelocity, float deltaTime)
{
    // Speed movement
    if (Motor.GroundingStatus.IsStableGround)
    {
        float currentVelocityMagnitude = currentVelocity.magnitude;

        Vector3 effectiveGroundNormal = Motor.GroundingStatus.GroundNormal;

        // Project velocity on plane
        currentVelocity = Motor.GetDirectionTangentToSurface(currentVelocity, effectiveGroundNormal) * currentVelocityMagnitude;

        // Calculate target velocity
        Vector3 InputRight = Vector3.Cross(MoveInputVector, Motor.CharacterUp);
        Vector3 normalizedInput = Vector3.Cross(effectiveGroundNormal, InputRight).normalized * _moveInputVector.magnitude;
        Vector3 targetMovementVelocity = normalizedInput * _movementSpeed;

        // Update movement velocity
        currentVelocity = Vector3.Lerp(currentVelocity, targetMovementVelocity, 1f - Mathf.Exp(-StableMovementSharpness * deltaTime));

        // Update animation
        if (currentVelocity.sqrMagnitude > 0.1f && Input.GetKey(KeyCode.LeftShift))
        {
            currentVelocity *= 0.5f;
            Walk();
        }
        else if (currentVelocity.sqrMagnitude > 0.1f && Input.GetKey(KeyCode.LeftShift))
        {
            Run();
        }
        else
        {
            Idle();
        }
    }

    // Two sets account additive velocity
    if (_internalVelocityAdd.sqrMagnitude > 0f)
    {
        currentVelocity += _internalVelocityAdd;
        _internalVelocityAdd = Vector3.zero;
    }
}

```

Рисунок 3.28 – Оновлення швидкості та встановлення анімації персонажа

Взаємодію персонажа з камерою та основні дії у грі визначаються у скрипті ExamplePlayer.cs, який об’єднує PlayerController та камеру (Додаток А).

Константам присвоюються назви осей введення, які визначені у налаштуваннях вводу. Камері задається точка слідування за персонажем, яка знаходиться на персонажі. Також додаються колайдери персонажа до списку колайдерів, які камера ігнорує при перевірці перекриття виду камери на персонажа.

Метод “SwitchState” включає або вимикає мотор персонажа та сам компонент. Використання даного методу буде розглянуто пізніше.

```
4 references
public void SwitchState(bool enable)
{
    Character.Motor.enabled = enable;
    enabled = enable;
}
```

Рисунок 3.29 – Зміна стану мотора

У методі “Update” викликається метод для обробки введення від користувача для керування персонажем.

Метод “LateUpdate” обробляє обертання камери разом із фізичними об’єктами. А також викликає метод для обробки введення від користувача для керування камерою.

Метод “HandleCameraInput” обробляє введення від користувача для керування камерою. Створюється вектор вводу для керування оглядом камери. Отримується введення для зміни масштабу камери та викликається метод для оновлення стану камери на основі вводу.

```
1 reference
private void HandleCameraInput()
{
    // Create the look input vector for the camera
    float mouseLookAxisUp = Input.GetAxisRaw(MouseYInput);
    float mouseLookAxisRight = Input.GetAxisRaw(MouseXInput);
    Vector3 lookInputVector = new Vector3(mouseLookAxisRight, mouseLookAxisUp, 0f);

    // Input for zooming the camera (disabled in WebGL because it can cause problems)
    float scrollInput = -Input.GetAxis(MouseScrollInput);

    if (UNITY_WEBGL)
        scrollInput = 0f;

    // Apply inputs to the camera
    CharacterCamera.UpdateWithInput(Time.deltaTime, scrollInput, lookInputVector);

    // Handle toggling zoom level
    if (Input.GetMouseButtonDown(1))
    {
        CharacterCamera.TargetDistance = (CharacterCamera.TargetDistance == 0f) ? CharacterCamera.DefaultDistance : 0f;
    }
}
```

Рисунок 3.30 – Обробка введення для керування камерою

					КН.КР.Б – 122 – 24 – ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		54

Метод “HandleCharacterInput” створює структуру “PlayerCharacterInputs” та заповнює її даними введення від користувача для осей руху та обертання камери. Викликається метод “SetInputs” у персонажа для застосування введення.

```
1 reference
private void HandleCharacterInput()
{
    PlayerCharacterInputs characterInputs = new PlayerCharacterInputs();

    // Build the CharacterInputs struct
    characterInputs.MoveAxisForward = Input.GetAxisRaw(VerticalInput);
    characterInputs.MoveAxisRight = Input.GetAxisRaw(HorizontalInput);
    characterInputs.CameraRotation = CharacterCamera.Transform.rotation;

    // Apply inputs to character
    Character.SetInputs(ref characterInputs);
}
```

Рисунок 3.31 - Обробка введення для керування персонажем

3.3.2 Взаємодія з NPC

Як і для головного персонажа, для оптимізації процесу зіткнень ми прикріплюємо компонент Capsule Collider, проте позначаємо як Is Trigger, щоб опрацювати зіткнення. Для базової фізики додаємо компонент Rigidbody.

Для обробки взаємодії створюємо скрипт “TeacherController” для усіх викладачів (Додаток Б).

На початку створюємо перерахування “t” teacher, яке визначає тип викладача (g, k1, m, k). Відповідно перерахування призначається кожному викладачу певним типом, для обробки подій щодо кожного викладача.

Метод “OnTriggerEnter(Collider other)” викликається коли інший колайдер потрапляє в колайдер цього об’єкта. Відбувається перевірка, чи інший колайдер є колайдером гравця. Якщо це гравець, то викликається метод “SwitchState” для вимкнення його руху і встановлюється анімація руху в режимі спокою. Також зупиняється слідування проректора за студентом.

Залежно від визначеного раніше типу викладача завантажується асинхронно відповідна сцена діалогу та квесту (Додаток В), а гравець переміщується на певну позицію.

					КН.КР.Б – 122 – 24 – ПЗ	Арк.
						55
Змн.	Арк.	№ докум.	Підпис	Дата		

Відображення деяких елементів інтерфейсу вимикається, задля запобігання зайвої інформації.

3.3.3 Діалоги та квести

Для розробки діалогових сцен та квестів було вирішено скористатися бібліотекою Fungus. Ця бібліотека дозволяє розробникам легко створювати складні сюжети, розгалужені дії та інші інтерактивні елементи. Однією з ключових переваг Fungus є його інтеграція з ігровим движком Unity, що дозволяє розробникам ефективно використовувати всі можливості Unity разом із зручним інтерфейсом Fungus.

Інтеграція Fungus з Unity спрощує процес створення інтерактивних ігор, оскільки вона надає розробникам доступ до широкого спектру інструментів і можливостей Unity для втілення складних ідей та концепцій гри. Розробники можуть легко створювати та управляти сценами, діалогами, персонажами та подіями у грі, використовуючи готові рішення та скрипти, які надає Fungus[20].

При використанні бібліотеки Fungus, діалоги та квести будуються у вигляді блок-схеми (рис. 3.32).



Рисунок 3.32 – Flowchart сцени

Кожен блок містить набір власних команд, та виконує свою роль у схемі (рис. 3.33). Змінні у flowchart (рис. 3.34) використовуються у квестах, для перевірки правильності відповіді.

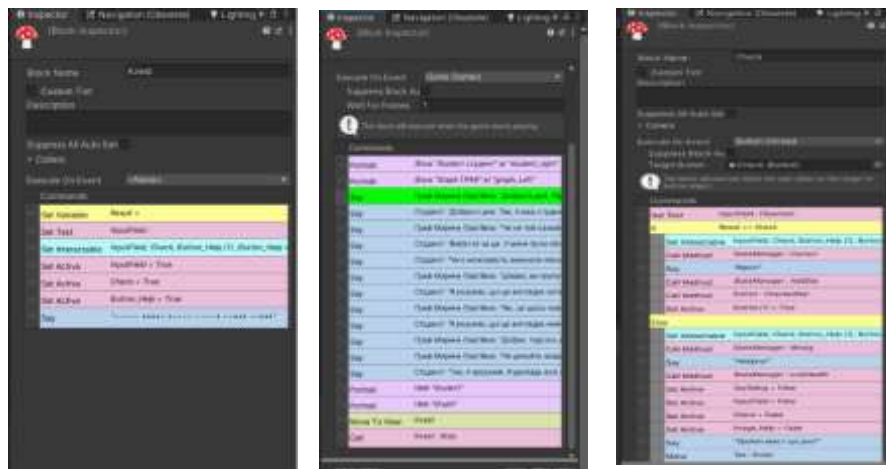


Рисунок 3.33 – Блоки з командами

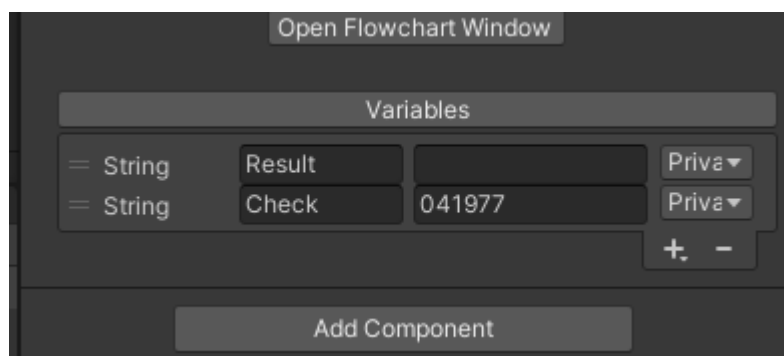


Рисунок 3.34 – Змінні Flowchart

Запуск сцени розпочинається із блоку, котрий позначений як виконуваний від початку сцени. За допомогою інтерактивних елементів, котрі містять у собі уже коди для роботи, встановлюються портрети персонажів, відображаються діалоги та меню вибору.

Відображення діалогів та реплік виконується за допомогою елементу Say (рис. 3.35). При огляді функціоналу елементу можна побачити вибір персонажа, де автоматично підтягується скрипт до нього, текст репліки, озвучка тексту, та інші чекбоксы, що служать для відображення та роботи елементу.

Для елементів вибору відповіді є блоки Menu, після вибору якого перенаправляють до наступного блоку, де відбувається дія (рис. 3.36).

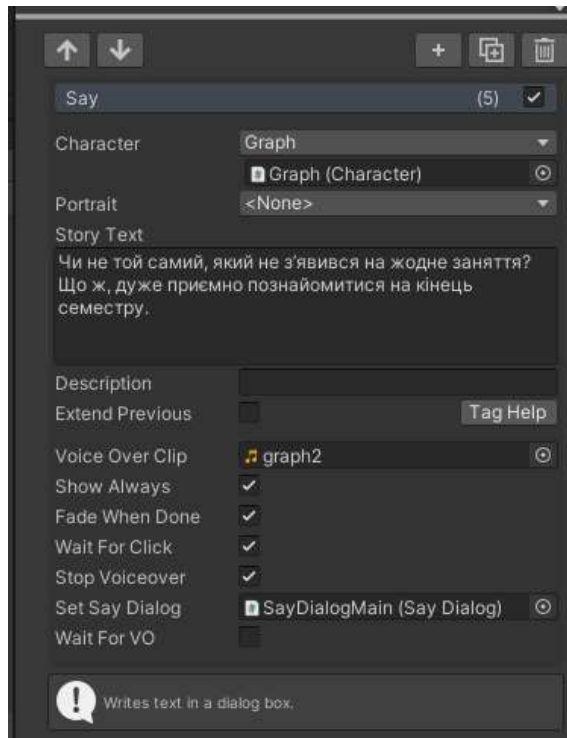


Рисунок 3.35 – Елемент Say

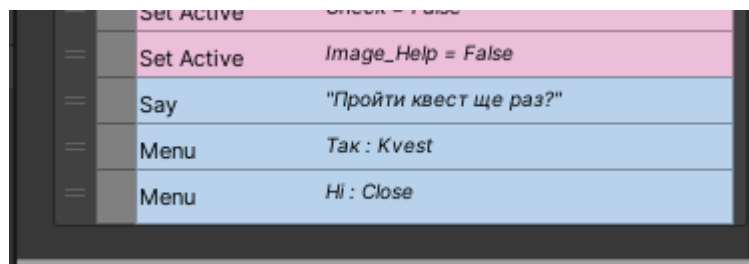


Рисунок 3.36 – Елемент Menu

До прикладу у даному блоці (див. рис. 3.36) потрібно обрати, чи проходити квест ще раз після невдалої спроби чи ні. Якщо обрано Так, то гавця переносить на початок квесту, якщо ні, то є можливість вийти.

Коли ми встановлюємо зв'язки між блоками у flowchart, ми визначаємо логіку, яка визначає, який блок буде відтворюватися після виконання попереднього. Fungus автоматично визначає, який блок показувати на сцені в поточний момент часу, відповідно до послідовності дій, встановленої у flowchart. Коли гравець грає гру, Fungus автоматично перемикається між блоками згідно з вказаними у flowchart правилами, що дозволяє відтворювати історію та взаємодіяти з гравцем у реальному часі.

Для усіх сцен взаємодії з NPC діалог однаковий (рис. 3.37).



Рисунок 3.37 – Діалог студента з викладачем

Оскільки квести є різними по завданням, проте вони схожі за функціоналом.

У квест входить завдання у вигляді зображення або репліки, поле для введення, кнопка перевірки та кнопка підказки.

На початку блоку Квест встановлюється порожнє значення для змінної Result та для поля введення. Встановлюється активність та видимість для поля введення, кнопки перевірки та кнопки підказки; виводиться завдання (рис. 3.38).

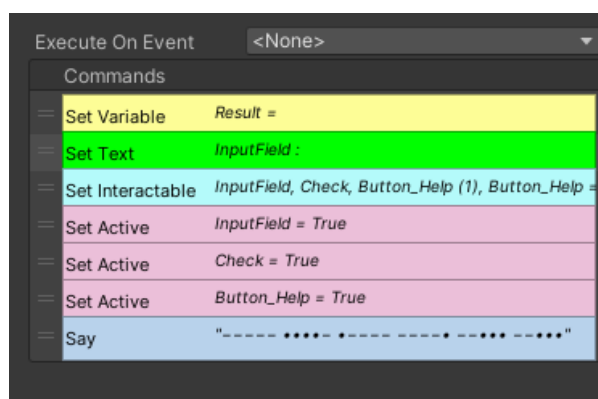


Рисунок 3.38 – Блок квест

Блок перевірки запускається при натисненні на кнопку перевірки (рис. 3.39). У ньому проходить перевірка правильності відповіді та додавання зірки

або віднімання життя, а також якщо відповідь правильна то об’єкт NPC знищується.

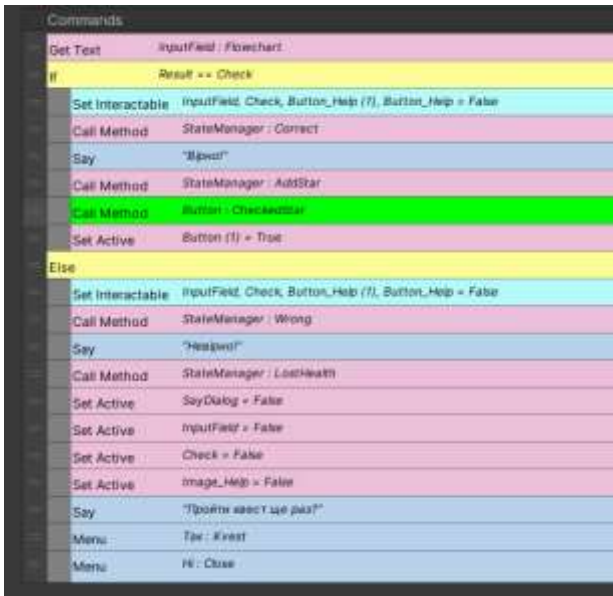


Рисунок 3.39 – Блок перевірки

На сцені присутній об’єкт зі скриптом “StateManager”, у якому відбувається пошук головного контролера життя і квестів та віднімається життя при поразці або віднімається квест від загальної кількості, що додає зірку (рис. 3.40).

```
0 references
private void AddStar()
{
    GameObject.FindGameObjectWithTag("Player").GetComponent<StateController>().kvest -= 1;
    AudioManager.PlaySFX(audioManager.addStar);
    Debug.Log("-1 kvest");
}

0 references
private void LostHealth()
{
    GameObject.FindGameObjectWithTag("Player").GetComponent<StateController>().health -= 1;
    AudioManager.PlaySFX(audioManager.minusHealth);
    Debug.Log("-1 health");
}
```

Рисунок 3.40 – Фрагмент коду скрипта “StateManager”

Закриття діалогової сцени та перехід до головної відбувається у скрипті “ButtonTemp” (Додаток Г). При правильній відповіді викликається метод “CheckedStar”, де булева змінна приймає значення true і при закритті викликається метод “GoBack”, у якому знаходиться відповідний об’єкт

викладача, котрий знищується. Викликається метод “State”, де стан неактивних елементів міняється на активний (рис. 3.41). Після цього асинхронна сцена закривається, у разі не виконання квесту, об’єкт викладача не знищується (рис. 3.42).

```

4 references
private void State()
{
    GameObject child = GameObject.Find("ItemsFind");
    itemText = child.GetComponentsInChildren<Text>();
    img = child.GetComponentInChildren<Image>();

    Image pause = GameObject.Find("Pause").GetComponentInChildren<Image>();
    pause.enabled = true;

    for (int i = 0; i < itemText.Length; i++)
    {
        itemText[i].enabled = true;
    }
    img.enabled = true;

    var player = FindObjectOfType<ExamplePlayer>();
    player.SwitchState(true);

    GameObject morozov = GameObject.Find("Morozov");
    morozov.GetComponent<Morozov_Controller>().SetShouldFollow(true);
    AudioManager.PlaySFX(audioManager.dialog);
}

```

Рисунок 3.41 – Метод State

```

0 references
public void CheckedStar()
{
    checkStar = true;
}

0 references
public void GoBackG()
{
    if (checkStar == true)
    {
        GameObject Graph = GameObject.FindGameObjectWithTag("Graph");
        Destroy(Graph);
        checkStar = false;
    }

    State();
    SceneManager.UnloadSceneAsync(2);
}

```

Рисунок 3.42 – Фрагмент коду закриття сцени

3.3.4 Життя та нагороди

Відображення та інформація про кількість життів та квестів зберігається та змінюється у скрипті “StateController”. Ключовими змінними є health – змінна, що відповідає за рівень здоров’я та kvest – змінна, що відповідає за кількість виконаних завдань гравцем. Також скрипт містить масив сердець та зірок для відображення на екрані (рис. 3.43).

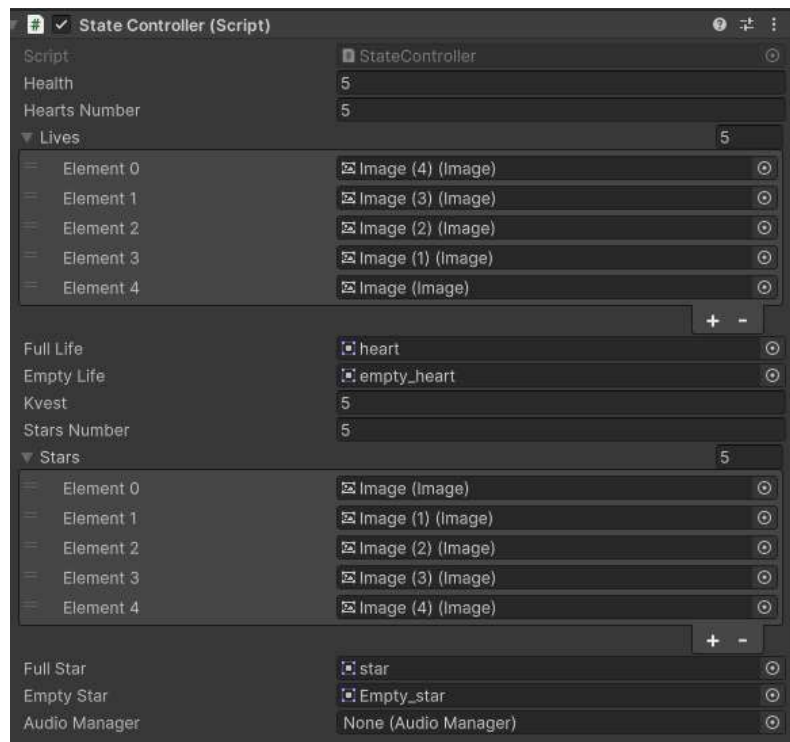


Рисунок 3.43 – Значення змінних контролера

Метод “Update” викликає методи “Hearts()” і “Stars()” при кожному кадрі гри для оновлення відображення "життів" і зірок.

Метод “Hearts()” (рис. 3.44) перевіряє, чи не перевищено максимальну кількість "життів"; оновлює відображення "життів" гравця на основі поточного рівня здоров'я; перевіряє, чи гравець має ще "життя". Якщо ні, відтворює звук та завантажує сцену для завершення гри.

Метод “ Stars()” (рис. 3.44) також перевіряє кількість зірок та оновлює дані на основі кількості виконаних завдань.

```

private void Hearts()
{
    if (health > heartsNumber)
    {
        health = heartsNumber;
    }
    for (int i = 0; i < lives.Length; i++)
    {
        if (i < health)
        {
            lives[i].sprite = fullLife;
        }
        else
        {
            lives[i].sprite = emptyLife;
        }
        if (i < heartsNumber)
        {
            lives[i].enabled = true;
        }
        else
        {
            lives[i].enabled = false;
        }
    }
    if (health <= 0)
    {
        AudioManager.PlaySFX(audioManager.dialog);
        SceneManager.LoadScene(6);
    }
}

private void Stars()
{
    if (kvest > starsNumber)
    {
        kvest = starsNumber;
    }
    for (int i = 0; i < stars.Length; i++)
    {
        if (i < kvest)
        {
            stars[i].sprite = emptyStar;
        }
        else
        {
            stars[i].sprite = fullStar;
        }
        if (i < starsNumber)
        {
            stars[i].enabled = true;
        }
        else
        {
            stars[i].enabled = false;
        }
    }
    if (kvest <= 0)
    {
        AudioManager.PlaySFX(audioManager.dialog);
        SceneManager.LoadScene(7);
    }
}

```

Рисунок 3.44 – Фрагмент коду скрипта

3.3.5 Алгоритм збирання предметів

На кожен предмет, котрий повинен бути знайденим, для оптимізації процесу зіткнень ми прикріплюємо компонент Box Collider, та позначаємо як Is Trigger, щоб опрацювати зіткнення, а також додаємо скрипт “Item” (рис.3.45).

```

Unity Message | 0 references
private void OnTriggerEnter(Collider other)
{
    PlayerInventory playerInventory = other.GetComponent<PlayerInventory>();

    if (playerInventory != null)
    {
        playerInventory.ItemCollected();

        gameObject.SetActive(false);
    }
}

```

Рисунок 3.45 – Скрипт “Item”

Цей код у Unity відповідає за взаємодію гравця з предметами в грі. Коли гравець зіштовхується з об'єктом, який має колайдер і він є тригером, викликається метод “OnTriggerEnter(Collider other)”. В цьому методі спершу отримується посилання на компонент “PlayerInventory” у гравця, який викликав зіткнення. Після чого перевіряється наявність цього компонента: якщо він існує, викликається метод “ItemCollected()” (рис.3.46) у гравця для обробки зібраного предмета, а також вимикається активність самого предмета у грі, щоб приховати його.

```
1 reference
public void ItemCollected()
{
    NumberOfItems++;
    OnItemCollected.Invoke(this);
    AudioManager.PlaySFX(audioManager.collect);

    if (NumberOfItems == 10)
    {
        GameObject.FindGameObjectWithTag("Player").GetComponent<StateController>().kvest -= 1;
        AudioManager.PlaySFX(audioManager.addStar);
    }
}
```

Рисунок 3.46 – Фрагмент скрипта PlayerInventory

Метод “ItemCollected()” у класі “PlayerInventory” викликається при зібранні гравцем нового предмета. Даний метод збільшує лічильник предметів у інвентарі на одиницю і викликає подію “OnItemCollected”, що дозволяє іншим об'єктам відреагувати на збір предмета гравцем. Параметр this передає об'єкт інвентаря гравця, який зібрав предмет. Також метод перевіряє, чи гравець зібрав 10 предметів, якщо так, то зменшує лічильник завдань (змінна kvest) у гравця на одиницю, знаходячи гравця за допомогою пошуку по тегу "Player".

Подія “OnItemCollected” дозволяє оновлювати текст кількості зібраних предметів у скрипті “InventoryUI” (рис. 3.47).

```
private Text itemText;

// Start is called before the first frame update
@ Unity Message | 0 references
void Start()
{
    GameObject child = GameObject.Find("ItemsFind");
    itemText = child.GetComponentInChildren<Text>();
}

// Update is called once per frame
0 references
public void UpdateItemText(PlayerInventory playerInventory)
{
    itemText.text = playerInventory.NumberOfItems.ToString();
}
```

Рисунок 3.47 – Оновлення тексту

3.3.6 Керування проректором

Першим кроком перед розробкою алгоритму є налагодження роботи анімацій в Аніматорі. Створюємо булеві змінні, які будуть визначати перехід між анімаціями (рис. 3.48).

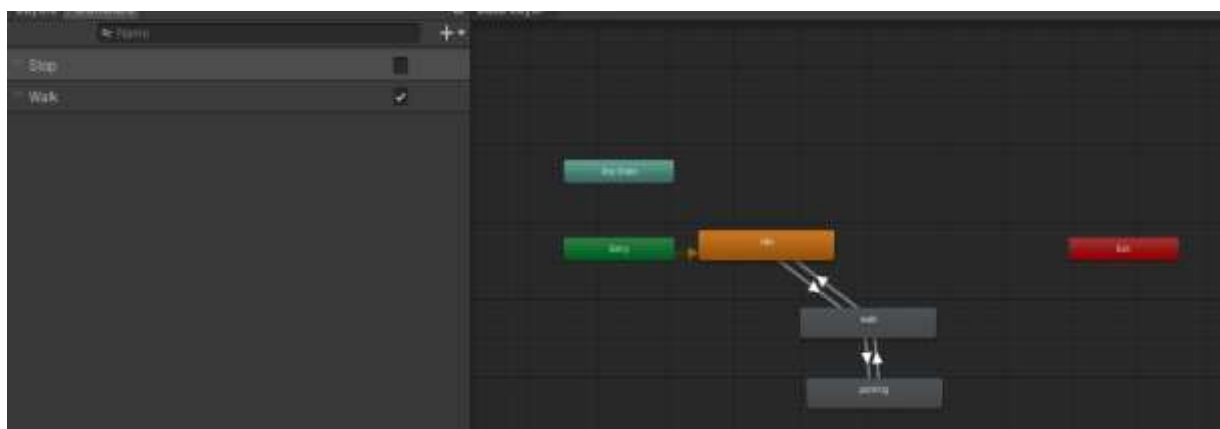


Рисунок 3.48 – Аніматор проректора

Наступним важливим кроком є додавання компоненту Capsule Collider для оптимізації процесу зіткнень та Rigidbody для базової фізики. Оскільки проректор повинен переслідувати студента ми додаємо йому компонент Nav Mesh Agent (рис. 3.49) та створюємо Nav Mesh для сцени.

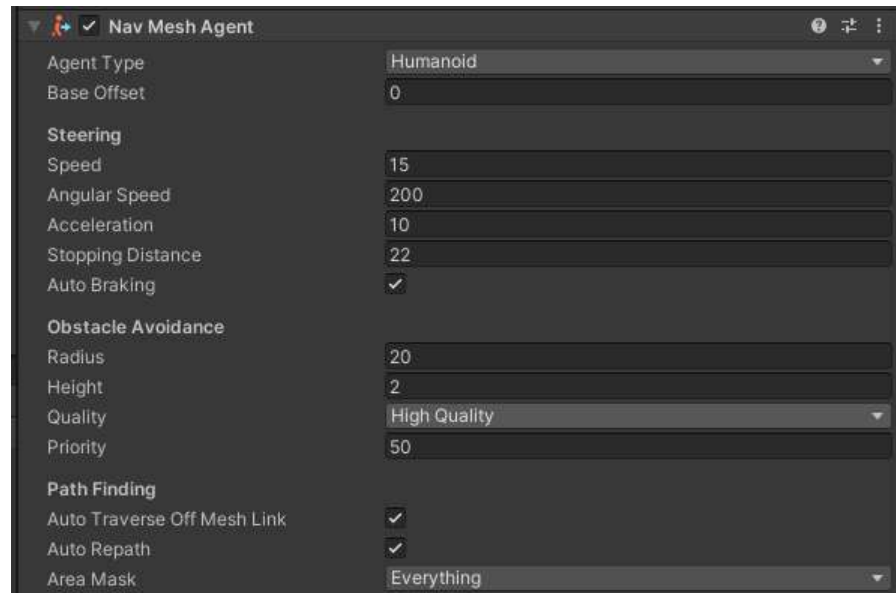


Рисунок 3.49 – Компонент Nav Mesh Agent для проректора

Після цих кроків можна перейти до розробки скрипта контроллера. Скрипт включає поля класу, які необхідно заповнити для подальшої розробки (рис. 3.50).

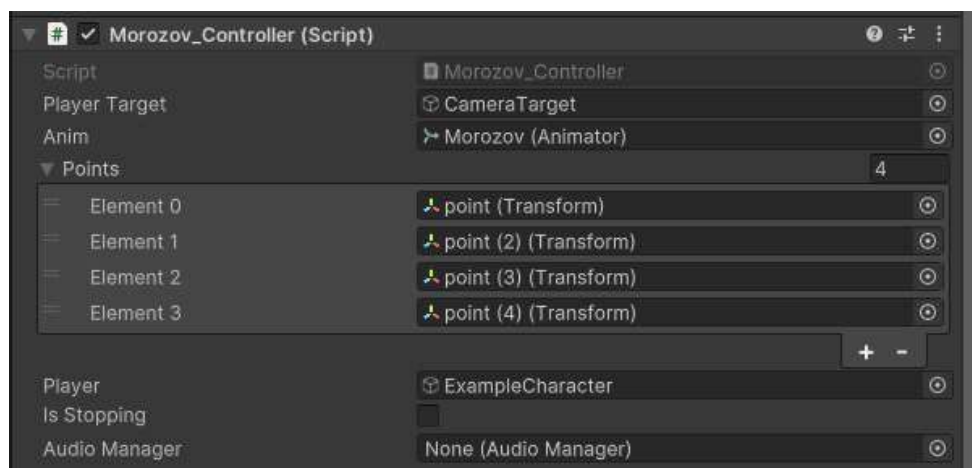


Рисунок 3.50 – Заповнені поля класу

Метод “Update” керує оновленням стану проректора кожен кадр, перевіряє відстань до цілі гравця. Якщо ці відстань у заданому діапазоні, то запускається метод “Control” через корутину. Якщо відстань не в допустимому діапазоні, зупиняє анімацію спокою і відключає статус зупинки (рис. 3.51).

```

void Update()
{
    updateTime += Time.deltaTime;

    if(nav.remainingDistance <= 20 && nav.remainingDistance >= 5)
    {
        StartCoroutine(Control());
    }
    else
    {
        anim.SetBool("Stop", false);
        isStopping = false;
        hasDealtDamage = false;
    }
}

```

Рисунок 3.51 – Фрагмент коду контролера

Метод “Control” (корутина)(рис. 3.52) керує роботою персонажа при зупинці біля гравця. Зупиняє рух гравця та відтворює анімацію спокою, перевіряє, чи ще не було завдано шкоду гравцеві (щоб не завдавати шкоду кілька разів при стоянні поруч). Здійснює переміщення персонажа на випадкову точку з масиву points, а гравця переміщає на початкову точку гри і повертає рух.

```

private IEnumerator Control()
{
    if (!isStopping)
    {
        isStopping = true;

        var player = FindObjectOfType<ExamplePlayer>();
        player.SwitchState(false);

        Animator playerAnim = GameObject.Find("ExampleCharacter").GetComponent<Animator>();
        playerAnim.SetFloat("Speed", 0);

        anim.SetBool("Stop", true);

        yield return new WaitForSeconds(3f);

        if (hasDealtDamage == false) // Перевіряє, чи ще віддали шкоду
        {
            Player.GetComponent<StateController>().health -= 1;
            AudioManager.PlaySFX(audioManager.minusHealth);
            Debug.Log("-1 health");
            hasDealtDamage = true;
        }

        transform.position = points[Random.Range(0, points.Length-1)].position;

        Player.GetComponent<PlayerController>().Motor.SetPosition(startPointPos);

        isStopping = false;

        player.SwitchState(true);

        yield return new WaitForSeconds(2f);
        //hasDealtDamage = false;
    }
}

```

Рисунок 3.52 – Фрагмент коду з зупинкою

Метод `SetShouldFollow(bool shouldFollow)` встановлює статус слідування персонажа за гравцем на основі параметра `shouldFollow`.

Метод “LateUpdate” викликається після оновлення всіх інших оновлень. Налаштовує орієнтацію персонажа так, щоб він дивився у напрямок гравця. Оновлює пункт призначення (destination) для `NavMeshAgent`, який керує рухом персонажа на основі розташування гравця (рис. 3.53).

```
2 references
public bool StoppedFollowing { get; set; }

2 references
public void SetShouldFollow(bool shouldFollow)
{
    StoppedFollowing = !shouldFollow;
    nav.destination = nav.transform.position;
    anim.SetBool("Walk", shouldFollow);
}

@ Unity Message | 0 references
private void LateUpdate()
{
    if (StoppedFollowing)
        return;

    Vector3 targetPosition = PlayerTarget.transform.position;
    targetPosition.y = transform.position.y;
    this.transform.LookAt(targetPosition);

    if (updateTime >= 1)
    {
        nav.destination = PlayerTarget.transform.position;
        Debug.Log(nav.remainingDistance);
        updateTime = 0;
    }
}
```

Рисунок 3.53 – Фрагмент коду пізнього оновлення

Цей код відтворює складний механізм керування рухом персонажа з використанням навмисних маршрутів, взаємодіє з анімацією та оброблює події при зіткненні з гравцем.

3.4 Аудіосупровід та налаштування гучності звуку

Для аудіосупроводу та налаштування гучності на сцені голоіного меню потрібно створити компонент, котрий міститиме скрипт “AudioManager” та два дочірні компоненти `Audio Source` для музики та звуків окремо.

Скрипт повинен містити посилання на компонент `AudioSource`, який використовується для відтворення фоновної музики та посилання на компонент

					КН.КР.Б – 122 – 24 – ПЗ	Арк.
						68
Змн.	Арк.	№ докум.	Підпис	Дата		

AudioSource, який використовується для відтворення звукових ефектів (SFX). Також публічні поля типу AudioClip, які зберігають різні аудіокліпи для різних звуків у грі.

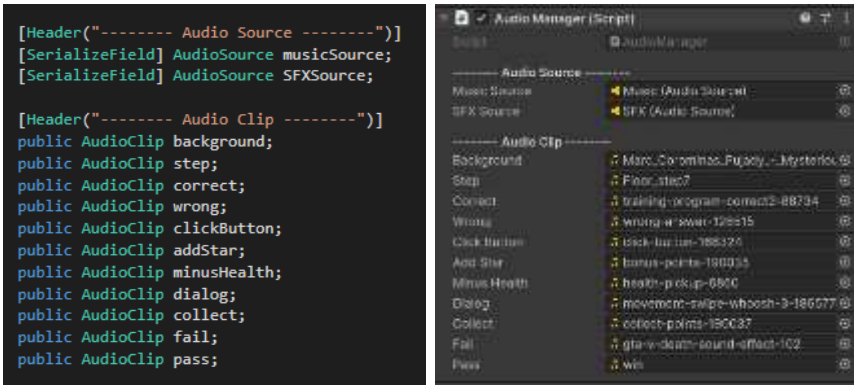


Рисунок 3.54 – Створені поля скрипту

Метод “Awake” в класі AudioManager перевіряє, чи вже існує інстанція цього класу. Якщо ні, він встановлює поточний об’єкт як єдиний екземпляр і робить його не знищуваним при завантаженні нової сцени за допомогою DontDestroyOnLoad. Якщо інший екземпляр уже існує, метод знищує поточний об’єкт, щоб забезпечити, що є лише одна інстанція AudioManager в грі.

Метод “Start” встановлює фонову музику і починає її відтворення.

Метод “PlaySFX” відтворює переданий звуковий ефект (clip) за допомогою компонента SFXSource методом PlayOneShot, який відтворює аудіокліп один раз.



Рисунок 3.55 – Методи класу AudioManager

Звуки викликаються методом PlaySFX (рис. 3.56).

```

@ Unity Message | 0 references
public class PlayerSound : MonoBehaviour
{
    public AudioManager audioManager;

    @ Unity Message | 0 references
    private void Awake()
    {
        audioManager = GameObject.FindWithTag("Audio").GetComponent<AudioManager>();
    }

    @ references
    public void StepSound()
    {
        audioManager.PlaySFX(audioManager.step);
    }
}

```

Рисунок 3.56 – Код присвоєння звуку ходіння у класі PlayerSound

Для керування гучності звуків та музики створюємо компонент Audio Mixer Controller та створюємо у групі Master групу Music та SFX. До цих груп створюємо параметри гучності та додаємо групи до відповідних компонентів Audio Source.

До елементу Canvas додаємо новий скрипт VolumeSettings, клас якого використовується для керування налаштуваннями гучності музики та звукових ефектів (SFX) у грі Unity. Серед змінних є посилання на об'єкт AudioManager, який використовується для керування аудіо в грі, посилання на слайдер, який використовується для регулювання гучності музики та посилання на слайдер, який використовується для регулювання гучності звукових ефектів.

Метод “Start” викликається при запуску скрипта. Він перевіряє, чи збережені налаштування гучності музики в PlayerPrefs. Якщо збережені налаштування існують, завантажує їх за допомогою методу LoadVolume(). Якщо ні, встановлює початкові значення гучності для музики та звукових ефектів, викликаючи методи SetMusicVolume() та SetSFXVolume() (рис. 3.57).

```

@ Unity Message | 0 references
private void Start()
{
    if (PlayerPrefs.HasKey("musicVolume"))
    {
        LoadVolume();
    }
    else
    {
        SetMusicVolume();
        SetSFXVolume();
    }
}

```

Рисунок 3.57 – Фрагмент класу VolumeSettings

					КН.КР.Б – 122 – 24 – ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		70

Метод “SetMusicVolume” зчитує значення гучності з слайдера musicSlider і встановлює відповідне значення гучності музики в AudioManager, перетворюючи його на логарифмічну шкалу ($\text{Mathf.Log10}(\text{volume}) * 20$) для більш природного сприйняття змін гучності. Після цього зберігає поточне значення гучності в PlayerPrefs під ключем "musicVolume" (рис. 3.58).

```
2 references
public void SetMusicVolume()
{
    float volume = musicSlider.value;
    myMixer.SetFloat("music", Mathf.Log10(volume)*20);
    PlayerPrefs.SetFloat("musicVolume", volume);
}
```

Рисунок 3.58 – Фрагмент класу VolumeSettings

Метод “SetSFXVolume” зчитує значення гучності з слайдера SFXSlider і встановлює відповідне значення гучності звукових ефектів в AudioManager, використовуючи логарифмічну шкалу ($\text{Mathf.Log10}(\text{volume}) * 20$). Після цього зберігає поточне значення гучності в PlayerPrefs під ключем «SFXVolume» (рис. 3.59).

```
2 references
public void SetSFXVolume()
{
    float volume = SFXSlider.value;
    myMixer.SetFloat("SFX", Mathf.Log10(volume) * 20);
    PlayerPrefs.SetFloat("SFXVolume", volume);
}
```

Рисунок 3.59 – Фрагмент класу VolumeSettings

Метод “LoadVolume” завантажує збережені значення гучності музики і звукових ефектів з PlayerPrefs, встановлює відповідні значення на слайдерах musicSlider та SFXSlider, а потім викликає методи SetMusicVolume() та SetSFXVolume(), щоб застосувати завантажені значення до AudioManager (рис. 3.60) .

```
1 reference
private void LoadVolume()
{
    musicSlider.value = PlayerPrefs.GetFloat("musicVolume");
    SFXSlider.value = PlayerPrefs.GetFloat("SFXVolume");

    SetMusicVolume();
    SetSFXVolume();
}
```

Рисунок 3.60 – Фрагмент класу VolumeSettings

Висновки до третього розділу

У цьому розділі було докладно розглянуто процес реалізації комп'ютерної гри "ZTURush". Спершу було створено ігрову сцену та моделі, що включало розробку різних ігрових об'єктів, персонажів і навколишнього середовища. Далі було розроблено інтерфейс гри, що дозволяє гравцям зручно взаємодіяти з грою та отримувати необхідну інформацію під час гри.

Важливим етапом стало впровадження геймплею, де було розглянуто керування персонажем, взаємодію з NPC, діалоги та квести. Детально розглянуто алгоритми управління життям персонажа та отримання нагород, а також розроблено систему збирання предметів і керування проректором, що додає грі динамічності та інтерактивності.

Особливу увагу приділено аудіосупроводу гри та налаштуванню гучності звуку, що забезпечує гравцям більш глибоке занурення в ігровий процес. Завдяки цьому, гра стала не лише візуально привабливою, але й звуково насиченою, що підвищує загальне враження від гри.

					КН.КР.Б – 122 – 24 – ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		72

ВИСНОВКИ

У кваліфікаційній роботі проведено дослідження можливостей використання інформаційних технологій для створення комп'ютерної гри у жанрі пригодницький квест. Аналіз охоплював різні аспекти, включаючи графічний двигун Unity, мову програмування C#, інструмент Blender для моделювання 3D об'єктів та анімацій. Це дозволило обрати оптимальний технологічний стек для реалізації гри "ZTURuah" у жанрі пригодницького квесту.

Проектування програмного забезпечення гри включало розробку архітектури з різними модулями для керування геймплеєм, взаємодії з об'єктами, обробки введення гравця та інші аспекти. Це дозволило зручно втілити логіку гри, управляти інтерактивними елементами, та забезпечити зручність у додаванні нових функцій та модифікації гри. Особлива увага була приділена модулю керування діалогами та квестами, що забезпечував розвиток сюжету та іммерсивний досвід для гравців. Кожен з модулів мав свою функціональність, що сприяло створенню цілісної та динамічної гри.

Особлива увага була приділена дизайну та інтерфейсу гри "ZTURush". Було розроблено ефективне та легко відтворюване меню налаштувань, що дозволяє гравцям налаштовувати параметри гри за своїм власним вибором. Також було створено послідовність подій та систему анімацій для забезпечення плавності та реалістичності геймплею.

У підсумку, результати дослідження підтверджують успішну реалізацію комп'ютерної гри "ZTURush" у жанрі платформера за допомогою сучасних інформаційних технологій. Розроблена гра має ефективну архітектуру, зручний інтерфейс та задовольняє вимоги сучасних гравців.

					КН.КР.Б – 122 – 24 – ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		73

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Нагорна А. Що чекає на ігрову індустрію цього року?. Speka - онлайн медіа про технології та підприємництво | SPEKA.media | SPEKA.media. URL: <https://speka.media/10-peredbacen-dlya-geimdev-industriyi-na-2024-rik-982j7m>.
2. Що таке VR? Поняття віртуальної реальності. Adobe. URL: <https://www.adobe.com/ua/products/substance3d/discover/what-is-vr.html>.
3. Штучний інтелект в ігровій індустрії - нові можливості для розробників та гравців. Mediacom. URL: <https://mediacom.com.ua/shtuchnij-intelekt-v-igrovij-industrii-novi-mozhливosti-dlya-rozrobnikiv-ta-gravtsiv/>.
4. Cloud-геймінг – чи зміниться розважальна індустрія?. Домени – перевірка та реєстрація доменів в Україні | Imena.ua. URL: <https://www.imena.ua/blog/cloud-gaming/>.
5. Rockstar New England. Bully: Scholarship Edition. Rockstar Games, 2008. URL: https://store.steampowered.com/app/12200/Bully_Scholarship_Edition/.
6. RenderWare engine. ModDB. URL: <https://www.moddb.com/engines/renderware>.
7. Додому. Hogwarts Legacy. URL: <https://www.hogwartslegacy.com/uk-ua>.
8. Hogwarts Legacy Walkthrough Team. What Game Engine Does Hogwarts Legacy Use? | Hogwarts Legacy | Game8. Game8 | The Top Gaming and App Walkthroughs Straight from Japan!. URL: <https://game8.co/games/Hogwarts-Legacy/archives/402966>.
9. Syberia: The World Before | Microids. Microids. URL: <https://www.microids.com/game-syberia-the-world-before/>.
10. What is a Gaming Engine?. Arm | The Architecture for the Digital World. URL: <https://www.arm.com/glossary/gaming-engines#:~:text=What%20is%20a%20Gaming%20or,a%20variety%20of%20programming%20languages>.

11. Unreal Engine 5.4 Release Notes. URL: <https://dev.epicgames.com/documentation/en-us/unreal-engine/unreal-engine-5.4-release-notes>.
12. We make the engine. You make it Unreal. Unreal Engine. URL: <https://www.unrealengine.com/en-US>.
13. CRYENGINE | CRYENGINE 5.5 Major Release. CRYENGINE. URL: <https://www.cryengine.com/news/view/cryengine-55-major-release>.
14. CRYENGINE | Enterprise. CRYENGINE. URL: <https://www.cryengine.com/enterprise>.
15. Key features of 3ds Max. URL: <https://www.autodesk.com/products/3ds-max/features>.
16. Blender 4.1 Manual. Blender Documentation - blender.org. URL: <https://docs.blender.org/manual/en/latest/index.html>.
17. Створення схем IDEF0 - Підтримка від Microsoft. Microsoft Support. URL: <https://support.microsoft.com/uk-ua/topic/створення-схем-idef0-ea7a9289-96e0-4df8-bb26-a62ea86417fc>.
18. Діаграма станів. StudFiles. URL: <https://studfile.net/preview/5010027/page:5/>.
19. Unity Asset Store - The Best Assets for Game Making. Unity Asset Store - The Best Assets for Game Making. URL: <https://assetstore.unity.com/>.
20. Home. GitHub. URL: <https://github.com/snozbot/fungus/wiki>.

ДОДАТКИ

					КН.КР.Б – 122 – 24 – ПЗ	Арк.
						76
Змн.	Арк.	№ докум.	Підпис	Дата		

Лістинг контролера камери:

```

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

namespace Scripts
{
    public class ExampleCharacterCamera : MonoBehaviour
    {
        [Header("Framing")]
        public Camera Camera;
        public Vector2 FollowPointFraming = new Vector2(0f, 0f);
        public float FollowingSharpness = 10000f;
        [Header("Distance")]
        public float DefaultDistance = 6f;
        public float MinDistance = 0f;
        public float MaxDistance = 10f;
        public float DistanceMovementSpeed = 5f;
        public float DistanceMovementSharpness = 10f;
        [Header("Rotation")]
        public bool InvertX = false;
        public bool InvertY = false;
        [Range(-90f, 90f)]
        public float DefaultVerticalAngle = 20f;
        [Range(-90f, 90f)]
        public float MinVerticalAngle = -90f;
        [Range(-90f, 90f)]
        public float MaxVerticalAngle = 90f;
        public float RotationSpeed = 1f;
        public float RotationSharpness = 10000f;
        public bool RotateWithPhysicsMover = false;

        [Header("Obstruction")]
        public float ObstructionCheckRadius = 0.2f;
        public LayerMask ObstructionLayers = -1;
        public float ObstructionSharpness = 10000f;
        public List<Collider> IgnoredColliders = new List<Collider>();

        public Transform Transform { get; private set; }
        public Transform FollowTransform { get; private set; }

        public Vector3 PlanarDirection { get; set; }
        public float TargetDistance { get; set; }

        private bool _distanceIsObstructed;
        private float _currentDistance;
        private float _targetVerticalAngle;
        private RaycastHit _obstructionHit;
        private int _obstructionCount;
        private RaycastHit[] _obstructions = new RaycastHit[MaxObstructions];
        private float _obstructionTime;
        private Vector3 _currentFollowPosition;

        private const int MaxObstructions = 32;

        void OnValidate()
        {
            DefaultDistance = Mathf.Clamp(DefaultDistance, MinDistance, MaxDistance);
            DefaultVerticalAngle = Mathf.Clamp(DefaultVerticalAngle, MinVerticalAngle,
            MaxVerticalAngle);
        }
    }
}

```

					КН.КР.Б – 122 – 24 – ПЗ	Арк.
						77
Змн.	Арк.	№ докум.	Підпис	Дата		

```

void Awake()
{
    Transform = this.transform;

    _currentDistance = DefaultDistance;
    TargetDistance = _currentDistance;

    _targetVerticalAngle = 0f;

    PlanarDirection = Vector3.forward;
}

// Set the transform that the camera will orbit around
public void SetFollowTransform(Transform t)
{
    FollowTransform = t;
    PlanarDirection = FollowTransform.forward;
    _currentFollowPosition = FollowTransform.position;
}

public void UpdateWithInput(float deltaTime, float zoomInput, Vector3 rotationInput)
{
    if (FollowTransform)
    {
        if (InvertX)
        {
            rotationInput.x *= -1f;
        }
        if (InvertY)
        {
            rotationInput.y *= -1f;
        }

        // Process rotation input
        Quaternion rotationFromInput = Quaternion.Euler(FollowTransform.up *
(rotationInput.x * RotationSpeed));
        PlanarDirection = rotationFromInput * PlanarDirection;
        PlanarDirection = Vector3.Cross(FollowTransform.up,
Vector3.Cross(PlanarDirection, FollowTransform.up));
        Quaternion planarRot = Quaternion.LookRotation(PlanarDirection,
FollowTransform.up);

        _targetVerticalAngle -= (rotationInput.y * RotationSpeed);
        _targetVerticalAngle = Mathf.Clamp(_targetVerticalAngle, MinVerticalAngle,
MaxVerticalAngle);
        Quaternion verticalRot = Quaternion.Euler(_targetVerticalAngle, 0, 0);
        Quaternion targetRotation = Quaternion.Slerp(Transform.rotation, planarRot *
verticalRot, 1f - Mathf.Exp(-RotationSharpness * deltaTime));

        // Apply rotation
        Transform.rotation = targetRotation;

        // Process distance input
        if (_distanceIsObstructed && Mathf.Abs(zoomInput) > 0f)
        {
            TargetDistance = _currentDistance;
        }
        TargetDistance += zoomInput * DistanceMovementSpeed;
        TargetDistance = Mathf.Clamp(TargetDistance, MinDistance, MaxDistance);

        // Find the smoothed follow position
        _currentFollowPosition = Vector3.Lerp(_currentFollowPosition,
FollowTransform.position, 1f - Mathf.Exp(-FollowingSharpness * deltaTime));
    }
}

```

```

        // Handle obstructions
        {
            RaycastHit closestHit = new RaycastHit();
            closestHit.distance = Mathf.Infinity;
            _obstructionCount = Physics.SphereCastNonAlloc(_currentFollowPosition,
ObstructionCheckRadius, -Transform.forward, _obstructions, TargetDistance,
ObstructionLayers, QueryTriggerInteraction.Ignore);
            for (int i = 0; i < _obstructionCount; i++)
            {
                bool isIgnored = false;
                for (int j = 0; j < IgnoredColliders.Count; j++)
                {
                    if (IgnoredColliders[j] == _obstructions[i].collider)
                    {
                        isIgnored = true;
                        break;
                    }
                }
                for (int j = 0; j < IgnoredColliders.Count; j++)
                {
                    if (IgnoredColliders[j] == _obstructions[i].collider)
                    {
                        isIgnored = true;
                        break;
                    }
                }

                if (!isIgnored && _obstructions[i].distance < closestHit.distance &&
_obstructions[i].distance > 0)
                {
                    closestHit = _obstructions[i];
                }
            }

            // If obstructions detector
            if (closestHit.distance < Mathf.Infinity)
            {
                _distanceIsObstructed = true;
                _currentDistance = Mathf.Lerp(_currentDistance, closestHit.distance,
1 - Mathf.Exp(-ObstructionSharpness * deltaTime));
            }
            // If no obstruction
            else
            {
                _distanceIsObstructed = false;
                _currentDistance = Mathf.Lerp(_currentDistance, TargetDistance, 1 -
Mathf.Exp(-DistanceMovementSharpness * deltaTime));
            }
        }

        // Find the smoothed camera orbit position
        Vector3 targetPosition = _currentFollowPosition - ((targetRotation *
Vector3.forward) * _currentDistance);

        // Handle framing
        targetPosition += Transform.right * FollowPointFraming.x;
        targetPosition += Transform.up * FollowPointFraming.y;

        // Apply position
        Transform.position = targetPosition;
    }
}
}
}

```

Лістинг контролера викладачів:

```

using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;
using KinematicCharacterController;
using Scripts;
using UnityEngine.UI;
using UnityEngine.AI;

public class TeacherController : MonoBehaviour
{
    public t teacher;
    public enum t { g, k1, m, k }

    private Text[] itemText;
    private Image img;

    public AudioManager audioManager;

    private void Awake()
    {
        audioManager =
        GameObject.FindGameObjectWithTag("Audio").GetComponent<AudioManager>();
    }

    private IEnumerator OnTriggerEnter(Collider other)
    {
        if (other.gameObject.tag == "Player")
        {
            var player = FindObjectOfType<ExamplePlayer>();
            player.SwitchState(false);

            Animator playerAnim =
            GameObject.Find("ExampleCharacter").GetComponent<Animator>();
            playerAnim.SetFloat("Speed", 0);

            GameObject child = GameObject.Find("ItemsFind");
            itemText = child.GetComponentsInChildren<Text>();
            img = child.GetComponentInChildren<Image>();

            Image pause = GameObject.Find("Pause").GetComponentInChildren<Image>();
            pause.enabled = false;

            for (int i = 0; i < itemText.Length; i++)
            {
                itemText[i].enabled = false;
            }
            img.enabled = false;

            GameObject morozov = GameObject.Find("Morozov");
            morozov.GetComponent<Morozov_Controller>().SetShouldFollow(false);

            yield return new WaitForSeconds(2f);

            if (this.teacher == t.m)
            {
                audioManager.PlaySFX(audioManager.dialog);
                SceneManager.LoadScene(3, LoadSceneMode.Additive);
            }
        }
    }
}

```

					КН.КР.Б – 122 – 24 – ПЗ	Арк.
						80
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        other.gameObject.GetComponent<PlayerController>().Motor.SetPosition(new
Vector3(-91f, 0.09f, -295f));

    }
    else if (this.teacher == t.g)
    {
        audioManager.PlaySFX(audioManager.dialog);
        SceneManager.LoadScene(2, LoadSceneMode.Additive);
        other.gameObject.GetComponent<PlayerController>().Motor.SetPosition(new
Vector3(-91f, 0.09f, -574f));
    }
    else if (this.teacher == t.k)
    {
        audioManager.PlaySFX(audioManager.dialog);
        SceneManager.LoadScene(4, LoadSceneMode.Additive);
        other.gameObject.GetComponent<PlayerController>().Motor.SetPosition(new
Vector3(222f, 0.09f, -17.5f));
    }
    else if (this.teacher == t.k1)
    {
        audioManager.PlaySFX(audioManager.dialog);
        SceneManager.LoadScene(5, LoadSceneMode.Additive);
        other.gameObject.GetComponent<PlayerController>().Motor.SetPosition(new
Vector3(-17.5f, 0.09f, -635f));
    }

    }

}
}
}

```



Рисунок В.1 – Зображення знаходження предметів квесту



Рисунок В.2 – Взаємодія з NPC

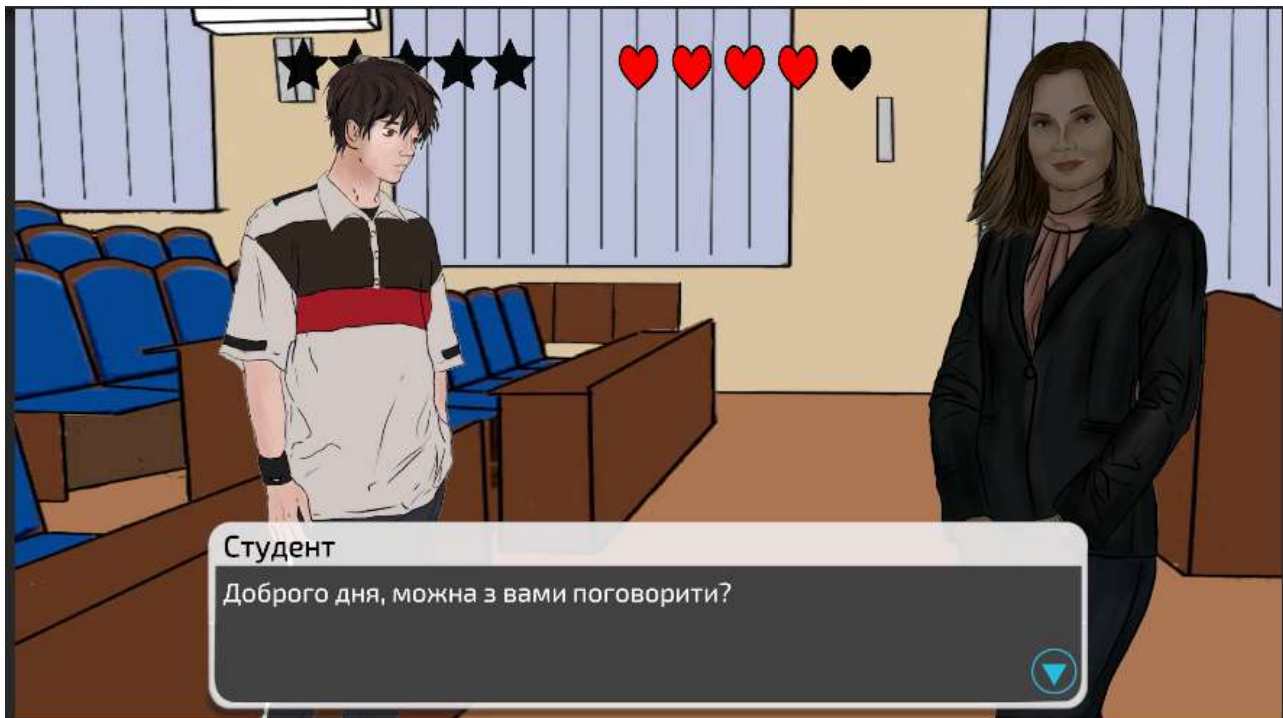


Рисунок В.3 – Діалог з викладачем

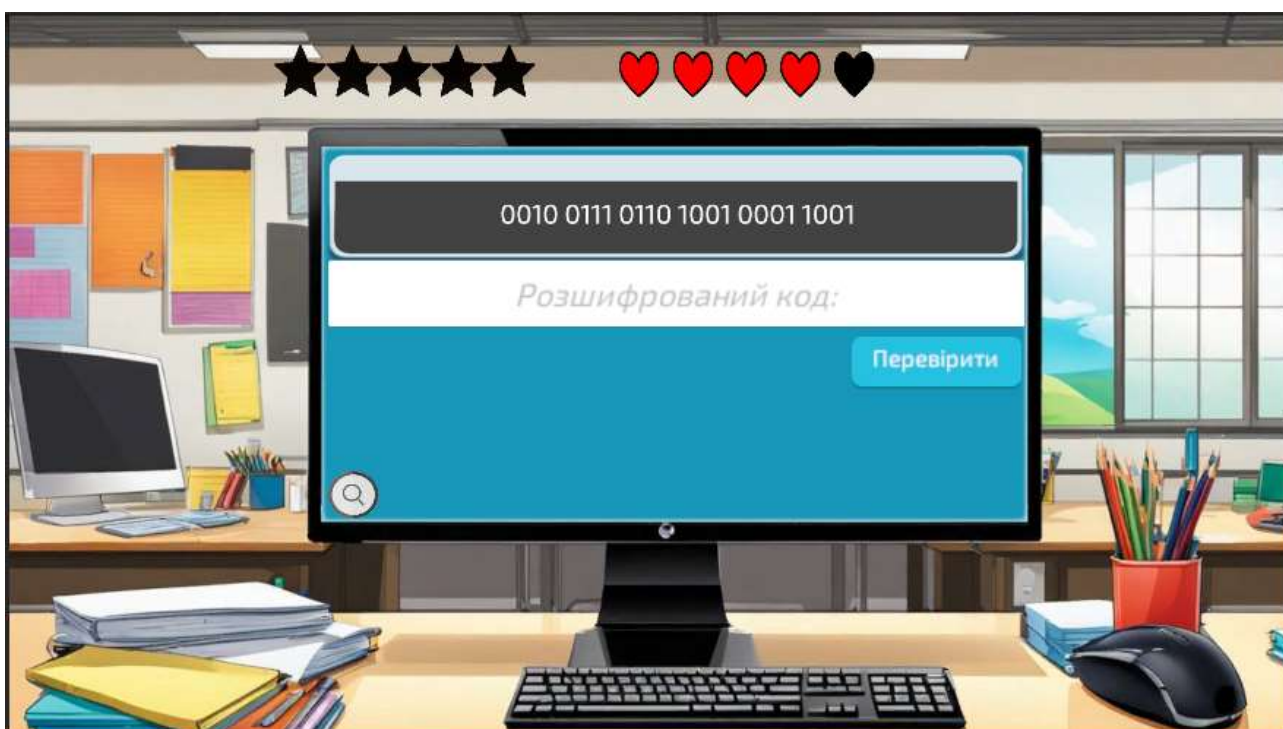


Рисунок В.4 – Квест-завдання

Змн.	Арк.	№ докум.	Підпис	Дата

КН.КР.Б – 122 – 24 – ПЗ

Арк.

83

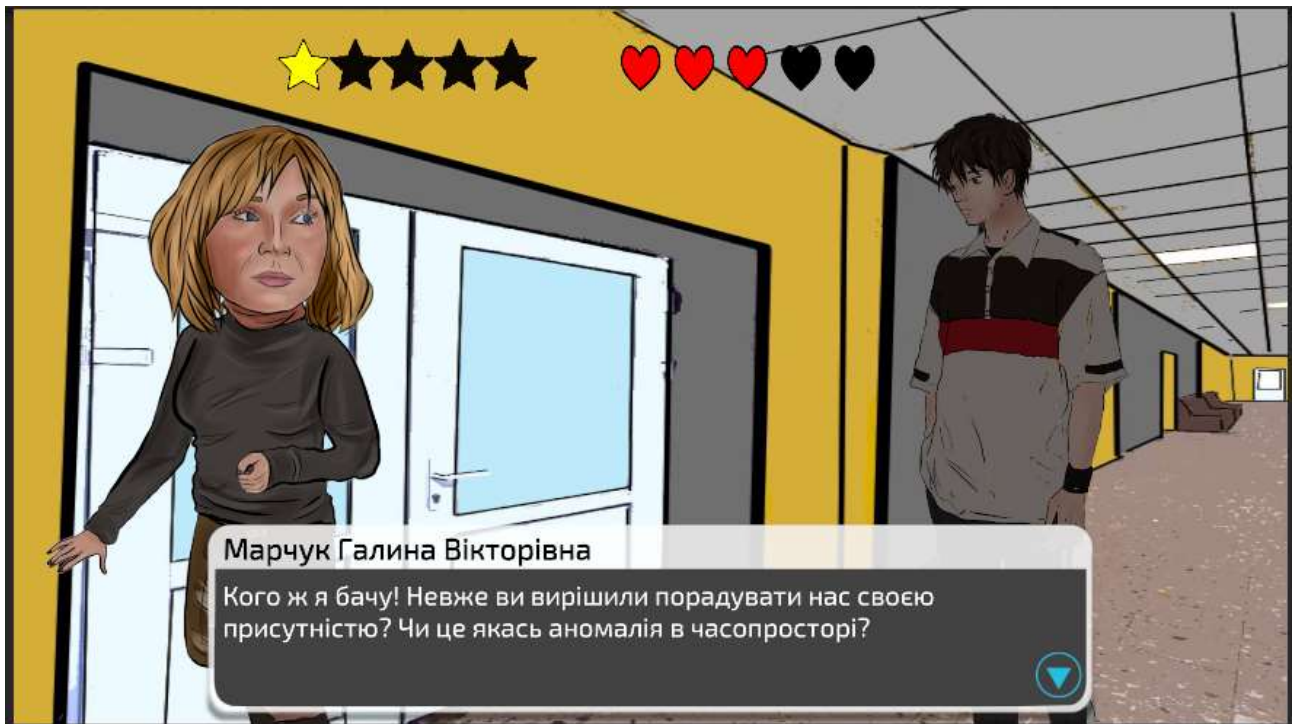


Рисунок В.5 – Діалог з викладачем

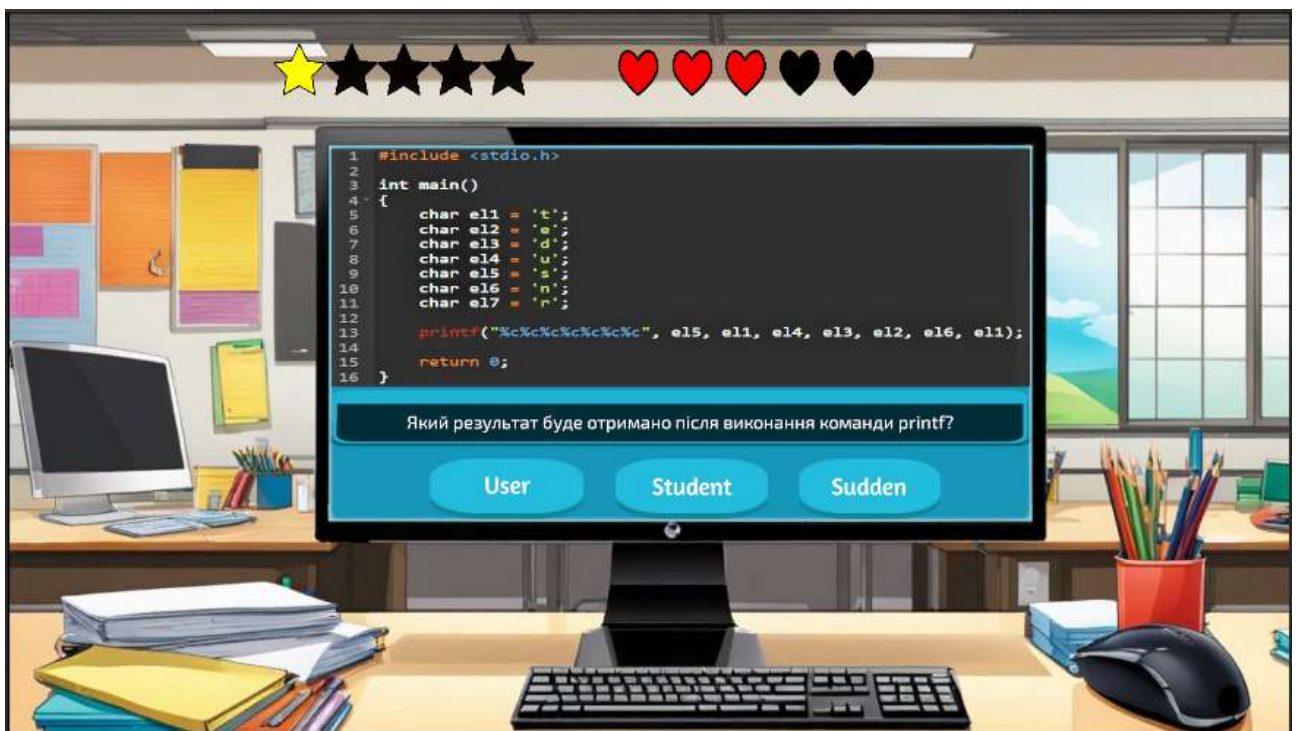


Рисунок В.6 – Квест-завдання

Змн.	Арк.	№ докум.	Підпис	Дата

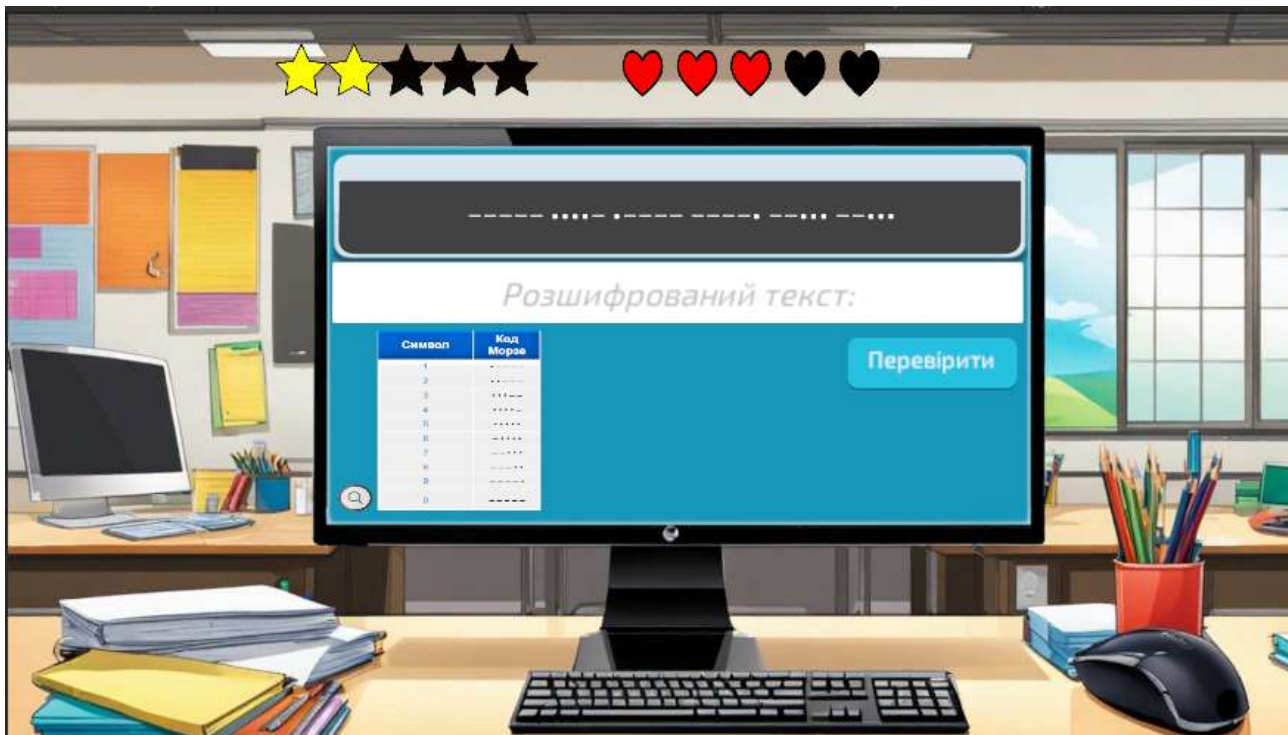


Рисунок В.7 – Квест-завдання

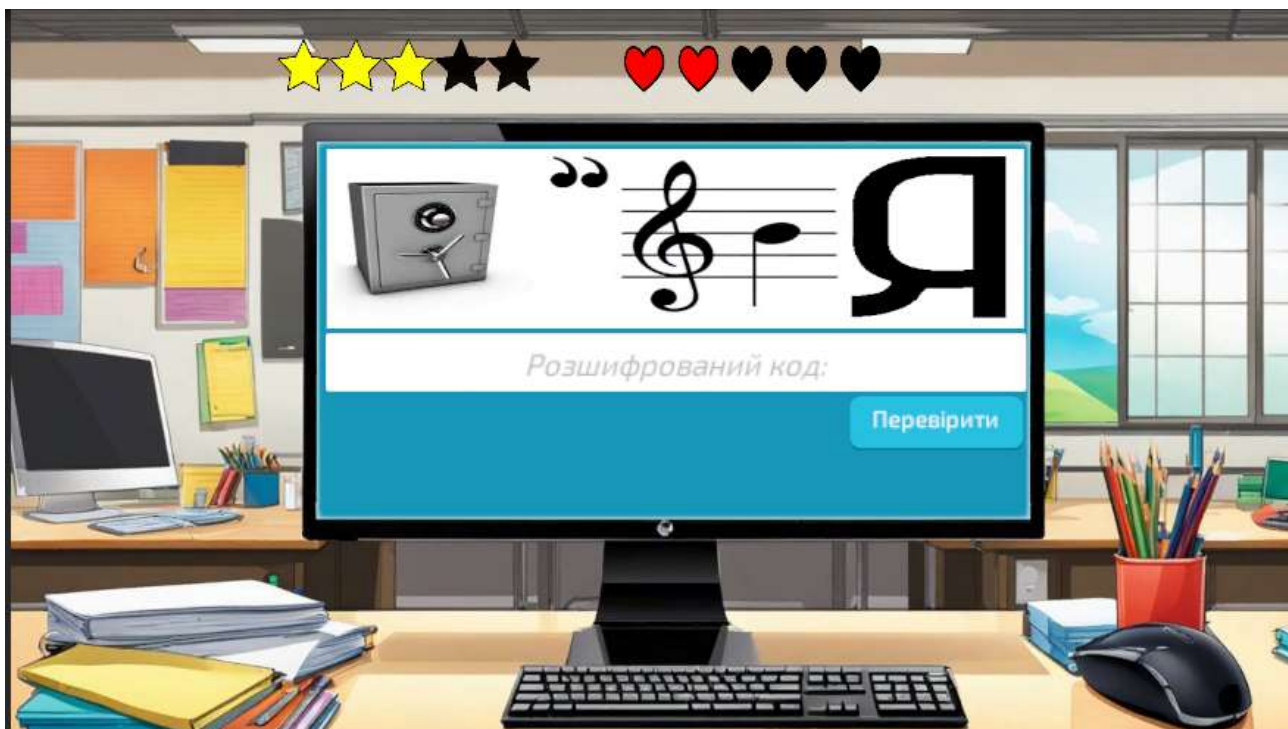


Рисунок В.8 – Квест-завдання



Рисунок В.9 – Діалог з проректором

Лістинг скрипта ButtonTemp.cs:

```

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

public class ButtonTemp : MonoBehaviour
{
    private Text[] itemText;
    private Image img;
    public AudioManager audioManager;
    private bool checkStar = false;

    private void Awake()
    {
        audioManager =
        GameObject.FindGameObjectWithTag("Audio").GetComponent<AudioManager>();
    }

    public void CheckedStar()
    {
        checkStar = true;
    }

    public void GoBackG()
    {
        if (checkStar == true)
        {
            GameObject Graph = GameObject.FindGameObjectWithTag("Graph");
            Destroy(Graph);
            checkStar = false;
        }

        State();
        SceneManager.UnloadSceneAsync(2);
    }

    public void GoBackM()
    {
        if (checkStar == true)
        {
            GameObject Marchuk = GameObject.FindGameObjectWithTag("Marchuk");
            Destroy(Marchuk);
            checkStar = false;
        }
        State();
        SceneManager.UnloadSceneAsync(3);
    }

    public void GoBackK()
    {
        if (checkStar == true)
        {
            GameObject Korotun = GameObject.FindGameObjectWithTag("Korotun");
            Destroy(Korotun);
            checkStar = false;
        }
        State();
        SceneManager.UnloadSceneAsync(4);
    }
}

```

					КН.КР.Б – 122 – 24 – ПЗ	Арк.
						87
Змн.	Арк.	№ докум.	Підпис	Дата		

```

    }
    public void GoBackK1()
    {
        if (checkStar == true)
        {
            GameObject Korotun1 = GameObject.FindGameObjectWithTag("Korotun1");
            Destroy(Korotun1);
            checkStar = false;
        }
        State();
        SceneManager.UnloadSceneAsync(5);
    }
    public void Menu()
    {
        audioManager.PlaySFX(audioManager.dialog);
        SceneManager.LoadScene(0);
    }

    private void State()
    {
        GameObject child = GameObject.Find("ItemsFind");
        itemText = child.GetComponentsInChildren<Text>();
        img = child.GetComponentInChildren<Image>();

        Image pause = GameObject.Find("Pause").GetComponentInChildren<Image>();
        pause.enabled = true;

        for (int i = 0; i < itemText.Length; i++)
        {
            itemText[i].enabled = true;
        }
        img.enabled = true;

        var player = FindObjectOfType<ExamplePlayer>();
        player.SwitchState(true);

        GameObject morozov = GameObject.Find("Morozov");
        morozov.GetComponent<Morozov_Controller>().SetShouldFollow(true);
        audioManager.PlaySFX(audioManager.dialog);
    }
}

```