Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно–інформаційних систем і програмної інженерії
(повна назва факультету)
Кафедра комп'ютерних наук
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

бакалавр
(назва освітнього ступеня)

на тему: Розробка демоверсії сюжетно–орієнтованої мобільної гри «Механічне серце» засобами ігрового рушія Unity

Виконав: студент IV курсу, групи СН–41
спеціальності 122 Комп’ютерні науки
(шифр і назва спеціальності)

(підпис) Філіпович Т.І. (прізвище та ініціали)

Керівник (підпис) Шимчук Г.В. (прізвище та ініціали)

Нормоконтроль (підпис) Литвиненко Я.В. (прізвище та ініціали)

Завідувач кафедри (підпис) Боднарчук І.О. (прізвище та ініціали)

Рецензент (підпис) Дідич І.С. (прізвище та ініціали)

Тернопіль
2023

Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно–інформаційних систем і програмної інженерії
(повна назва факультету)

Кафедра комп'ютерних наук
(повна назва кафедри)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Боднарчук І.О.
(підпис) (прізвище та ініціали)
«__» _____ 2023 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня Бакалавр
(назва освітнього ступеня)

за спеціальністю 122 Комп'ютерні науки
(шифр і назва спеціальності)

Студенту Філіпович Тетяні Іванівні
(прізвище, ім'я, по батькові)

1. Тема роботи Розробка демоверсії сюжетно–рольової мобільної гри засобами ігрового рушія Unity.

Керівник роботи Шимчук Григорій Валерійович, ст. викл. кафедри КН
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом ректора від « 07 » лютого 2023 року № 4/7–133.

2. Термін подання студентом завершеної роботи 20.06.2023р.

3. Вихідні дані до роботи Літературні та інтернет джерела.

4. Зміст роботи

Вступ. 1.Поняття «Мобільна гра». Підбір програмних застосунків. 1.1 Поняття мобільної гри. Візуальна новела. Етапи розробки. 1.2 Огляд конкурентів. 1.3 Підбір ігрового рушія. 1.4 Візуальна складова гри. 1.5 Adobe Photoshop. 1.6 Adobe Illustrator. 1.7 Leonardo.ai. 1.8 Висновок до першого розділу. 2. Реалізація сюжетно–рольової гри «Механічне серце». 2.1 Розробка концепту та вмісту прологу історії. 2.2 Скачування та налаштування Unity. 2.3 Розробка сюжетно–орієнтованої гри. 2.4 Висновок до другого розділу. 3. Безпека життєдіяльності, основи охорони праці. 3.1 Вплив електромагнітного випромінювання від мобільних телефонів. 3.2. Психофізіологічне розвантаження гравців мобільної гри «Механічне серце». Висновки. Перелік джерел. Додатки.

5. Перелік графічного матеріалу

1. Титульна сторінка. 2. Мета, об'єкт та предмет дослідження. 3. Актуальність обраної теми. 4. Огляд конкурентів. 5. Огляд ігрових рушіїв. 6. Етапи розробки. 7. Створення візуальної. 8. Налаштування ігрового рушія. 9. Організація робочого простору. 10. Сцени гри. 11. Розробка механік гри. 12. Висновок.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Безпека життєдіяльності, основи охорони праці			

7. Дата видачі завдання 23 січня 2023 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1.	Ознайомлення з завданням до кваліфікаційної роботи	23.01.2023	*Виконано*
2.	Підбір джерел про розробку мобільних ігор	24.01.2023–26.01.2023	*Виконано*
3.	Опрацювання джерел по темі кваліфікаційної роботи	27.01.2023–31.01.2023	*Виконано*
4.	Виконання дослідження щодо розробки мобільної гри	01.02.2023–07.02.2023	*Виконано*
5.	Оформлення розділу «Поняття «Мобільна гра». Підбір	08.02.2023–09.02.2023	*Виконано*
	програмних застосунків»		
6.	Оформлення розділу «Реалізація сюжетно–рольової	10.02.2023–12.02.2023	*Виконано*
	гри «Механічне серце»»		
7.	Виконання завдання до підрозділу «Безпека життєдіяльності»	05.06.2023–06.06.2023	*Виконано*
8.	Виконання завдання до підрозділу «Основи охорони	07.06.2023–08.06.2023	*Виконано*
	праці»		
9.	Оформлення кваліфікаційної роботи	09.06.2023–11.06.2023	*Виконано*
10.	Нормоконтроль	12.06.2023–13.06.2023	*Виконано*
11.	Перевірка на плагіат	14.06.2023	*Виконано*
12.	Попередній захист кваліфікаційної роботи	15.06.2023	*Виконано*
13.	Захист кваліфікаційної роботи	20.06.2023	

Студент _____ Філіпович Т.І.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Шимчук Г.В.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Розробка демоверсії сюжетно–орієнтованої мобільної гри «Механічне серце» засобами ігрового рушія Unity // Кваліфікаційна робота освітнього рівня «Бакалавр» // Філіпович Тетяна Іванівна // Тернопільський національний технічний університет імені Івана Пулюя, факультет комп'ютерно–інформаційних систем і програмної інженерії, кафедра комп'ютерних наук, група СН–41 // Тернопіль, 2023 // С. – 51 , рис. – 14, табл. – 2, слайди – 12, додат. – 3, бібліогр. – 31.

Ключові слова: мобільна гра, ігровий рушій, візуальна новела, демоверсія, Unity, Adobe Photoshop, штучний інтелект, розробка.

Кваліфікаційна робота присвячена розробці демоверсії сюжетно–рольової мобільної гри засобами ігрового рушія Unity.

Мета роботи розробити демоверсію мобільної сюжетно–рольової мобільної гри «Механічне серце», відшукати рішення та способи реалізації проекту за допомогою ігрового рушія Unity.

В першому розділі кваліфікаційної роботи розглянуто перспективи розробки мобільних ігор, їх жанрові різновидності, окремо було дано визначення візуальній новелі, особливості її створення та основні аспекти. Описано план реалізації інтерактивної історії, проведено аналіз програмного забезпечення, підбір найкращого ігрового рушія, стилістики та середовища розробки.

В другому розділі кваліфікаційної роботи описано основні етапи розробки візуальної новели «Механічне серце», процес створення: сюжету гри, візуальну частину, спрайтів персонажів, фонів та діалогів, розробку користувацького інтерфейсу, написання програмного коду та його реалізація у ігровому рушії.

В третьому розділі розглянуто питання щодо впливу електромагнітного випромінювання від мобільних телефонів та психофізіологічного розвантаження при застосуванні мобільного додатка «Механічне серце».

ANNOTATION

Development of demo–version story–oriented mobile game "Mechanical heart" by means of Unity Engine // Qualification work of the education level “Bachelor” // Filipovych Tetiana Ivanivna // Ternopil Ivan Pului National Technical University, Faculty of Computer Information Systems and Software Engineering, Computer Science Department, group CS–41 // Ternopil, 2023 // P. – 51, pic. – 14 , tables – 2, draw. – 12 , annexes – 3, ref – 31.

Keywords: mobile game, game engine, visual novel, demo version, Unity, Adobe Photoshop, artificial intelligence, development.

The qualification work is devoted to development of demo–version story–oriented mobile game «Mechanical Heart» by means of Unity Engine.

The goal of the work of this qualification work is to develop a demo version of a story–oriented mobile game «Mechanical heart», find solutions and ways to implement the project using the Unity game engine.

The first section of the qualification work contains prospects for developing mobile games, their genre varieties, in addition, the basic information about visual novels, features of its creation and main aspects. The plan for the implementation of the interactive story is described, and an analysis of software is conducted to select the best game engine, style, and development environment.

The second section of the qualification work describes main stages of development of a story–oriented mobile game «Mechanical heart», such as: writing the plot, development of the visual part (sprites characters, backgrounds and dialogue boxes), UI/UX Kit, writing scripts and its implementation in the game engine.

The third section of the qualification work discusses issues related to the impact of electromagnetic radiation from mobile phones and psychophysiological relaxation when using the mobile application «Mechanical Heart» are discussed.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

Ассет – цифровий об'єкт, який переважно складається з однотипних даних, неподільна сутність, яка представляє частину ігрового контенту. Це може бути зображення, текст, звук, тощо.

Анімація (з лат. anima – душа і похідного фр. animation – оживлення) – вид кіномистецтва, засноване на штучному уявленні про рух об’єкта. Анімація створюється за рахунок знімання послідовних фаз руху намальованих або об'ємних об'єктів.

Арт (від англ. art – мистецтво) – англіцизм, являє собою словом професійної комунікації серед художників.

Баг – це неочікувана помилка або недолік у програмному забезпеченні.

Білдінг (від анг. build – будувати) – це процес створення виконуваних файлів гри, які готові до розповсюдження або тестування.

Візуальна новела – це жанр комп'ютерних ігор, який поєднує у собі елементи інтерактивної книги та ігрового процесу.

Гейм–дизайнер – режисер у розробці ігор, відповідальний за весь процес створення продукту, від зародження ідеї до реалізації.

Геймплей – термін, що описує взаємодію гравця з грою. Він включає в себе правила, механіку, завдання, виклики та весь досвід, який гравець отримує під час гри.

Гейм–розробник – програміст, який за допомогою мови програмування та ігрового рушія, розробляє ігрові механіки, інтерфейс тощо. Відповідає за технічну частину розробки.

Дедлайн (від англ. deadline – мертва лінія) — англіцизм, який має таке ж значення, як і українське слово реченéць: крайній термін (дата або/чи час), до якого має бути виконано певне завдання.

Жанр – вид творів у галузі якого–небудь мистецтва, який характеризується певними сюжетними та стилістичними ознаками.

Ігровий рушій – це програмний фреймворк, призначений для розробки відеоігор, і зазвичай містить відповідні бібліотеки та програми підтримки.

Інді(від англ. independent – незалежний) – програмне забезпечення або відеогра, які створені незалежно від великих видавництв та фінансової підтримки.

Інтерактивна історія – різновид художнього твору, де сюжет не є лінійним, або фіксованим. Всі події залежать безпосередньо від вибору читача.

Контролер – у даному контексті сукупність елементів, яка відповідає за ту чи іншу механіку гри.

Корутина – це спеціальний вид функції в Unity, який дозволяє виконувати довгі або важкі заходи без блокування основного потоку виконання програми.

ОС – операційна система.

Пайплайн – послідовність дій у процесі розробки.

ПЗ – програмне забезпечення.

Прототип – це пробний або експериментальний зразок гри, який створюється для перевірки ідей, механік та концепцій до повноцінного розроблення гри.

Спрайт (англ. sprite – «фея, ельф») – графічний об’єкт у комп’ютерній графіці (зазвичай растрове зображення, іноді анімація).

ШІ – штучний інтелект.

ЗМІСТ

ВСТУП

Актуальність теми. Мобільні технології дозволяють нам завжди бути на зв'язку, отримувати доступ до інформації та проводити дозвілля у будь–якому зручному місці. Одним з найпопулярніших способів проведення даного дозвілля є мобільні ігри, які займають велику частину ринку цифрових розваг.

Серед різноманіття жанрів мобільних ігор, варто виділити відносно молоду вітку сюжетно–рольових ігор – візуальні новели. Даний жанр, який надає можливість гравцю ототожнювати себе із головним героєм та прямо впливати на ігровий світ та події у ньому, тільки набуває популярності в Україні, але є швидко розвиваючою індустрією із великими прибутками за кордоном, зокрема в азійських країнах.

Крім розваг, мобільні ігри є важливою галуззю розвитку сучасної індустрії. Розробка мобільних ігор відкриває безліч можливостей для компаній у сфері маркетингу, реклами та брендингу, а також важливим інструментом для популяризації культури та історії різних країн та народів.

Дана кваліфікаційна робота буде присвячена дослідженню розробки гри жанру візуальна новела, дослідженню особливостей та втілення їх у кінцевий продукт. Буде висвітлено та описано процес розробки ігрового процесу, їх візуальну та аудіо частину, а також оцінено перспективи розробки мобільних ігор у майбутньому.

Мета і задачі дослідження. Метою даної кваліфікаційної роботи освітнього рівня «Бакалавр» є :

– Проаналізувати програмне забезпечення для розробки сюжетно–орієнтованої мобільної гри на ігровому рушії Unity.

– Сформувати вимоги відповідно до поставленого завдання.

– Розробити візуальні частину гри (середовище, персонажі, дизайн інтерфейсу).

– Розробити ігрову архітектуру.

– Розробити аудіо–оформлення та музичний супровід.

– Провести тестування демоверсії гри.

Практичне значення одержаних результатів. Розроблена у ході даної кваліфікаційної роботи сюжетно–рольову гру «Механічне серце» можна використовувати, як робочий прототип для подальшої розробки та роботи над цією та багатьма іншими інтерактивними історіями. Розроблена ігрова архітектура дозволить в подальшому завантажувати та створювати повноцінну сюжетно–рольову гру. Також слід зазначити позитивний вплив на психоемоційне розвантаження гравця.

РОЗДІЛ 1. ПОНЯТТЯ «МОБІЛЬНА ГРА». ПІДБІР ПРОГРАМНИХ ЗАСТОСУНКІВ

1.1 Поняття мобільної гри. Візуальна новела. Етапи розробки

Мобільні ігри – це ігри розроблені для мобільних пристроїв, смартфонів, кишенькових ПК, планшетів та портативних медіа плеєрів. Мобільні ігри можуть бути як самостійними (створені тільки для малоформатних пристроїв), так і пристосованою копією вже наявної відеогри.

З кожним роком ринок мобільних ігор стає все різноманітнішим і прибутковішим. Так, наприклад, за даними веб–журналу GamesIndustry.biz [12], за весь період існування (на даний момент це 2.5 роки) сюжетно–рольової гри «Genshin Impact» із елементами візуальної–новели та Action/RPG китайська компанія «Hoyovers» заробила 3.7 мільярда доларів.

На даний момент мобільні ігри є найбільш популярними серед ігор на різних пристроях, таких як: ноутбуки, ПК та ігрові консолі. Це відбувається за рахунок того, що смартфони та їх аналоги є більш доступними по ціні, користувач може відволіктися на гру у будь–який зручний для нього час та місці, у себе вдома, дорозі тощо. Розробляються такі моделі смартфонів, які не поступаються потужністю та витривалістю користувацьким ПК, а середовища розробки та емулятори, дозволяють адаптувати гру до великої кількості версій операційних систем та моделей[26]. Також немало важливим плюсом є вартість їхньої розробки. На відміну від широкомасштабних проектів, які вимагають іноді сотні тисяч доларів, та цілу команду професіоналів, розробкою мобільних ігор може зайнятися пересічна людина з великим ентузіазмом і бажанням створити щось нове. Також це забезпечує неймовірну різноманітність жанрів, механік та сюжетів.

За жанрами мобільні ігри можна поділити на категорії [23]:

– аркади;
– головоломки;
– настільні;
– перегони;
– стратегії;
– інтерактивні історії.

У різних жанрах зазвичай ставлять різні пріоритети на аспектах гри. Так наприклад у перегонах та пригодницьких іграх, особливу увагу приділяють ігровому процесу та наративному дизайну (різноманітністю квестів і задач), а у інтерактивних історіях надію покладають на сюжет та візуальне оформлення.

Але спільним залишається одне це основні етапи розробки. Для того щоб створити мобільну гру необхідно[24]:

1. Придумати ідею гри, вибрати жанр, прописати механіки, та за потреби сюжет.

2. Коротко описати проект, створити план дій для зручності, вказавши всі часові рамки.

3. Підібрати ігровий рушій та мову програмування відповідно.

4. За допомогою ігрових рушіїв розробити робочий прототип гри.

5. Вибрати стилістику графіки, створити список необхідних спрайтів, відмалювати їх, та відредагувати відповідно до вимог розробки .

6. Розробити інтерфейс, логотип тощо.

7. Реліз

Серед різноманіття стилів та жанрів гри, особливо привернув увагу жанр сюжетно–рольових ігор – візуальна новела, де вся увага зосереджується на сюжеті, розвитку фентезійних світів та взаємодією між вигаданими персонажами. Також завдяки такій особливості жанру, багато іноземних країн розглядають його як метод популяризації своєї культури та розголосу про таланти їх митців у світі загалом. Оскільки у подальшому проект буде

вдосконалюватися, і планується як збірник було прийнято рішення розробити сюжетно–рольову гру – візуальну новелу.

Візуальна новела – це жанр відеоігор, у якому історія розказується гравцеві за допомогою зображень, коротких текстових повідомлень та звуків[29].

На рисунку 1.1 зображено стандартний вигляд інтерфейсу та ігрову механіку на прикладі візуальної новели «Ace Attorney» [15].

Рисунок 1.1 – Геймплей візуальної новели «Ace Attorney»

Досить часто її ототожнюють із новою віткою сучасної драматургії, як новий спосіб та погляд на розповідь історії. Ступінь інтерактивності у таких іграх зазвичай дуже низький. Користувачу лише зрідка необхідно робити певний вибір у діалоговому вікні, результатом який може призвести до того чи іншого розвитку подій. Зазвичай основну увагу у розробці таких ігор є сюжет, взаємодія між персонажами та візуальний аспект. У інтерактивних історіях зазвичай фабула не лінійна, а швидше нагадує дерево рішень, де кожен прийняти вибір гравцем, впливає на подальші події в історії, і приводить до одного з декількох фіналів.

1.2 Огляд конкурентів

Перед розробкою концепції гри слід провести детальний аналіз потенційних компаній та продуктів конкурентів, чим саме їхні візуальні новели зацікавили читачів, які проблеми виникали та на яких платформах випускали демоверсії ігор.

Для огляду конкурентів було створену таблицю 1.1, у якій висвітлено всі необхідні дані про ігри, автора (розробник компанія чи інді), перша платформа випуску/платформа випуску на сьогоднішній день, прибутки за весь час існування, переваги(особливості) та недоліки. Оглядитимемо п’ять різних, але дуже цікавих візуальних новел, які стали своєрідною класикою жанру, а саме: «Mystic Messenger», «Ace Attorney», «The Arcana: A Mystic Romance», «Doki Doki Literature Club!», та «To the Moon».

Таблиця 1.1 – Огляд потенційних конкурентів гри.

	Mystic Messenger	**Ace Attorney**	**The Arcana: A Mystic Romance**	**Doki Doki Literature Club!**	**To the Moon**
1	**2**	**3**	**4**	**5**	**6**
Видавець	Cheritz компанія	Capcom компанія	Nix Hydra Компанія (раніше команда розробників)	Team Salvato команда розробників	Freebird Games компанія
Платформа –дебют	GooglePlay	Nintendo	Itch.io	Itch.io	Steam
Платформа випуску	GooglePlay та AppleStore	Windows, iOS, Nintendo 3DS, Android	GooglePlay та AppleStore	Microsoft Windows, macOS і Linux	Microsoft Windows, macOS, Linux, iOS, Android, Nintendo Switc

Продовження таблиці 1.1

1	2	3	4	5	6
Прибутки	Невідомо	≈ 4 млн. грн	Невідомо	Умовно безкоштовна ≈ 16 млн. грн	≈ 12 млн. грн
Переваги	Особливий гемплей віртуальної переписки, захоплюючий сюжет	Змішання різних геймплейних елементів, таких як дослідження місця злочину	Цікавий інтерфейс, додаткові міні–ігри	Інноваційний підхід, музичний супровід	Заплутані головоломки та моменти розгадування загадок
Недоліки	Величезна вага додатку, низька оптимізація та старіших пристроях	Залежність від текстового контенту та надто розгалужені розмови	Погана оптимізація гри на різних форматах пристрою	Дещо за складні загадки та головоломки	Обмежена інтерактивність, надто короткі серії

Кожна із даних візуальних новел є особливою і унікальною у своєму роді, за що і отримують любов фанатів та великі показники скачувань. Проаналізувавши даний огляд можемо зробити висновок що майже кожна новела розроблена компанією яка спеціалізується у даному жанрі, або командою розробників (5–6 чоловік).

Також варто зауважити, що ігри які спершу випускалися командами, на початках завантажувалися не на стрімінгових сервісах або інтернет магазинах, а на сайті інді–розробників, де отримували реакції пересічних гравців вдосконалювали творіння і тільки тоді намагалися дати розголос про проєкт.

В наслідок чого було прийнято рішення розробити якісний фундамент проєкту, якісну текстову архітектуру, зміну персонажів тощо, і вже в подальшому публікувати роботу на таких сайтах як itch.io, та удосконалювати продукт до тих пір поки він не буде готовий приносити плоди.

1.3 Підбір ігрового рушія

Після того, як визначено вид гри, її жанрові особливості та вимоги, наступним кроком є вибір основи майбутнього проекту – ігрового рушія.

Ігровий рушій – це програмна основа розробки будь–якої відеогри, яка контролює технічну сторону реалізації та працездатність геймплейних, графічних та інших елементів гри. Він спрощує процес розробки гри шляхом уніфікації та систематизації її внутрішньої структури.

Ігрові рушії дозволяють додавати рівні, елементи та персонажів, прописувати події, які відбуваються залежно від дій головного героя, фізику предметів та характер поведінки об’єктів тощо. По–своїй природі ігрові рушії це те саме що й фреймворк у загальному розумінні. Рушії це програмні платформи які спрощують написання та створення ігор, та надають змогу у швидші терміни створювати багатогранні проєкти з цікавими механіками та складною побудовою.

При виборі необхідного ігрового рушія для мобільної візуальної новели, у інді розробника відкриваються чотири варіанти: Godot, Ren'Py, Unity і Unreal Engine. Кожен із них по–своєму особливий та унікальний, тому розглянемо їх переваги та недоліки, та проведемо порівняльний аналіз для виявлення найкращого та найзручнішого для даної гри.

Перш за все розглянемо порівняльну таблицю усіх чотирьох потенційних рушія за основними критеріями, такими як: кросплатформеність, актуальність на ринку праці, зручність у роботі, важкість вивчення, мова програмування на якій пишуться ігри, та вартість користування. Результати огляду висвітлено у таблиці 1.2.

Таблиця 1.2 – Порівняльна таблиця актуальних ігрових рушіїв

Рушій	**Godot**	**Ren'Py**	**Unity**	**Unreal Engine**
Кросплатфо–рменість	Так	Так	Так	Так
Актуальність	Висока	Низька	Дуже висока	Дуже висока
Зручність у роботі	Дуже зручний	Зручний	Зручний	Зручний
Важкість освоєння	Середня	Легкий	Важкий	Важкий
Мова програмування	GDScript	Python	C#	C++
Вартість ліцензії	Безкоштовний	Безкоштовний	Безкоштовний до $100k/рік	5% від доходу

За результатами аналізу важливо зазначити що кожен із даних рушіїв має усі важливі та потрібні механіки для розв’язання поставленої задачі, і вибір між даними рушіями залежать від вимог до проєкту поставлених компанією або самим розробником, і, якщо ми говоримо про інді розробку, від особистого досвіду та знань програміста. Говорячи про створення візуальної новели «Механічне серце», основними вимогами до рушія є наступні аспекти:

– Присутня можливість розробки на різних пристроях, безпосередньо на ОС Android,

– Актуальність на ринку праці та популярність рушія загалом.

– Можливість перевірки роботи гри у реальному часі.

– Достатня кількість навчального контенту та легкість використання.

– Наявність документації та служби підтримки.

– Мова програмування C#.

Зваживши усі за та проти, ігровим рушієм на даний кваліфікаційний проєкт був обраний – Unity, адже саме він найбільше задовольняє всі висунуті вимоги,

для комфортної та перспективної роботи у цій галузі. Розглянемо даний рушій більш детально.

Unity — один з найпопулярніших ігрових рушіїв у світі, який використовується для розробки ігор, віртуальної реальності, доповненої реальності та інтерактивних додатків. Він відомий своєю широкою популярністю серед розробників завдяки ряду переваг, які забезпечують його успіх на ринку[11]. Unity дозволяє розробникам створювати ігри та програми, які можуть працювати на кількох платформах, включаючи Windows, MacOS, Linux, iOS, Android тощо.

Unity дозволяє створювати широкий спектр типів ігор і є чудовим інструментом для створення високоякісних ігор і програм. Також варто відзначити досить легкий та зрозумілий у засвоєнні користувацький інтерфейс, який однозначно прискорить процес навчання та знайомство з програмний забезпеченням, і дасть можливість швидко приступити до роботи.

На рисунку 1.2 зображено робоче поле та інтерфейс ігрового рушія Unity.

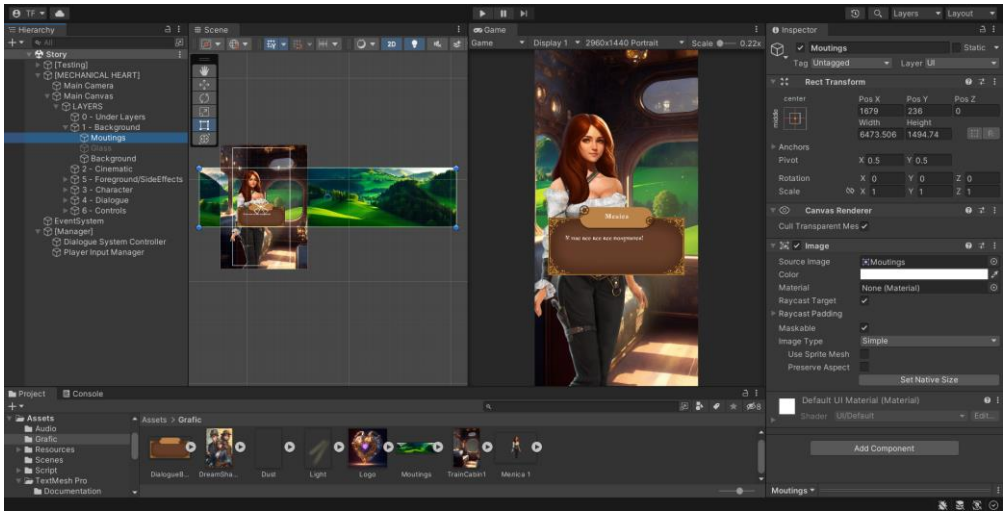

Рисунок 1.2 – Інтерфейс рушія Unity

Unity є популярним ігровим рушієм завдяки своїй кросплатформеності [13], легкості використання, багатому функціоналу, великій спільноті та можливостям для розширення. Він надає розробникам гнучкість та потужність

для створення високоякісних ігор та інтерактивних додатків для різних платформ. Тому саме на ньому буде виконана дана кваліфікаційна робота.

1.4 Візуальна складова гри

Загалом коли перед гейм–розробником постає питання графіки у грі, у нього є декілька варіантів для роздумів, такі як: піксельна графіка, вокселька, 3д і так далі. Але у даному випадку ці види графіки використовуються частіше як експеримент, спроба принести щось нове у жанр, або той чи інший вид графіки підходить по сюжету та стилістиці історії.

У даному випадку візуальний аспект відіграє ключову роль у розповіді історії, і вимагає найбільшого розуміння читача, графіка повинна бути зрозумілою і також приємною і атмосферною. Для цього завдання ідеально підійде «класична» растрова та векторна графіка.

Растрова графіка – це тип графіки, який використовує матрицю пікселів для зображення зображень. Кожен піксель містить інформацію про колір і яскравість в даній частині зображення[6]. Даний стиль дуже популярний в комп'ютерних іграх, графічних редакторах та фоторедакторах.

Растрова графіка є базою для мобільних ігор, через те що вона проста в використанні і має низьке навантаження на процесор. Вона використовується для створення зображень, спрайтів і текстур, які зустрічаються в мобільних іграх. Растрова графіка також може бути компенсована для зменшення розміру файлу і оптимізації завантаження і відображення на мобільних пристроях.

На рисунку 1.3 зображено приклад роботи виконаної у растровій графіці, авторства художників Le Vuiong [3] та damaoye233 [18].

Рисунок 1.3 – Растрова графіка

Векторна графіка – це спосіб зображення, який використовує математичні моделі для побудови графічних об'єктів. На відміну від растрової графіки, яка заснована на фіксованій сітці пікселів, векторна графіка використовує математичні формули, щоб побудувати геометричні форми, такі як прямокутники, кола та криві[25].

Незважаючи на зростання прогресивніших графічних технологій, векторна графіка продовжує мати відданих прихильників і залишається впізнаваним і улюбленим стилем в ігровій індустрії. Але в силу того, що даного типу графіка менш універсальна ніж растрова, її рідше використовують, як фундамент візуального аспекту. Векторна графіка ідеальна для дещо іншого, а саме для інтерфейсу користувача. В силу того, що інтерфейс сам по собі не вимагає високої деталізації, але залежний від якості масштабування, робить даного виду графіку, чудовим інструментом у руках художника.

На рисунку 1.4 зображено приклад використання векторної графіки на прикладі робіт українського UI/UX художника John Mevis [10] та корейського Shining Melon [9].

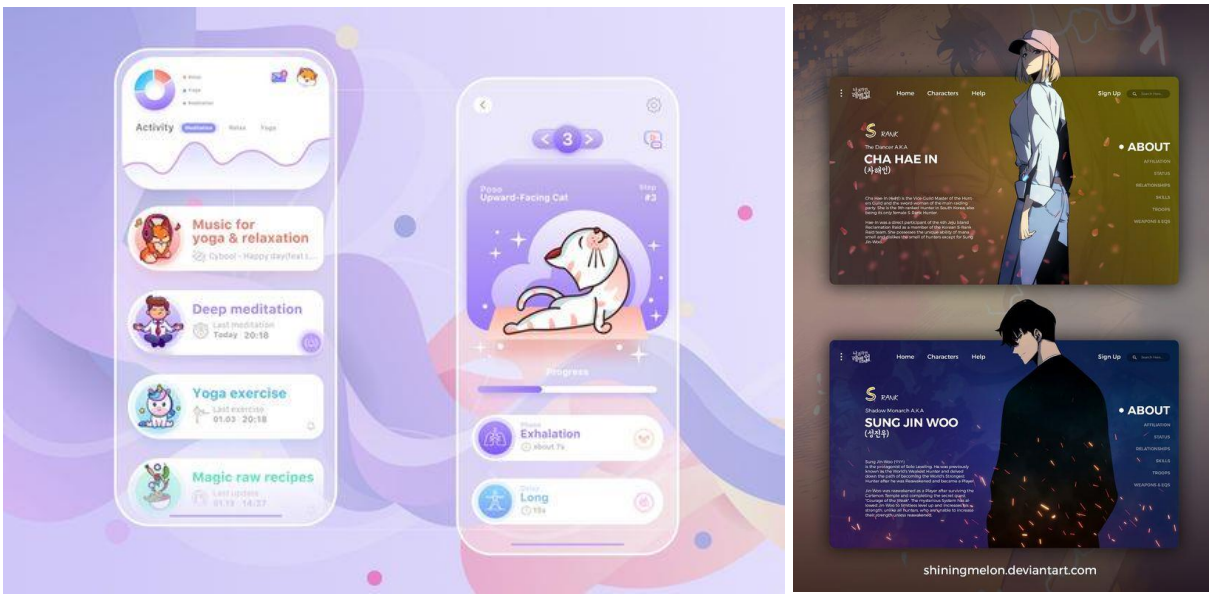

Рисунок 1.4 – Векторна графіка

Отже, стилістика даної візуальної новели буде виконана, за допомогою растрової графіки (спрайти персонажів, фонів, інструментів, логотип, тощо) та векторної (інтерфейс користувача, діалогове вікно), із використанням технологій штучного інтелекту для пришвидшення роботи над грою.

1.5 Adobe Photoshop

Adobe Photoshop — це професійний багатофункціональний растровий редактор, розроблений та виданий компанією Adobe Systems.

При розробці даного проекту Adobe Photoshop використовуватиметься як основний інструмент та фундамент візуальної частини роману. Завдяки тому, що редактор дозволяє художнику імпортувати додаткові пензлі та текстури, встановлювати плагіни, а також деформовувати інтерфейс як заманеться користувачу, робота стає не тільки швидшою, а ще й приємною.

На рисунку 1.5 зображено налаштований інтерфейс Adobe Photoshop.

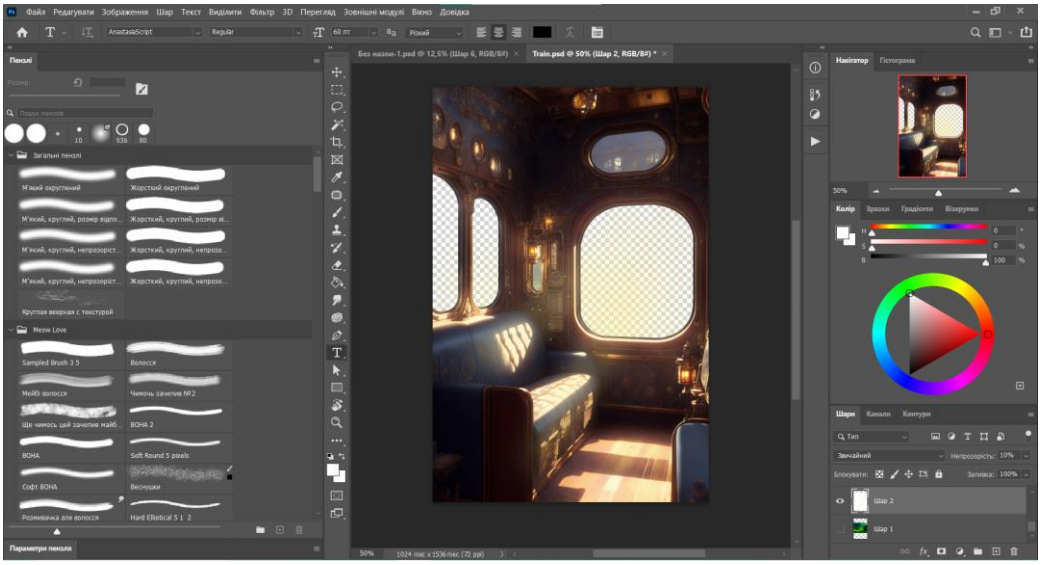

Рисунок 1.5 – Інтерфейс Adobe Photoshop

При малюванні всіх спрайтів та ассетів застосовуватиметься практично весь функціонал програми, але ось основний перелік функцій та інструментів[2]:

– Інструменти виділення – прямолінійне та звичайне ласо.

– Інструмент шари – дозволяють робити зміни на окремих шарах і зберігати їх окремо від інших елементів зображення, що дає більшу контроль і гнучкість при роботі.

– Інструменти рисування – набір пензлів, окрім стандартних, також використовуватимуться текстурні та авторські кисті.

– Інструмент трансформації – а точніше Ctrl+T, дана комбінація клавіш надає змогу змінювати та деформувати виділений об’єкт, обертати, перетягувати, змінювати розмір та пропорції.

– Інструмент фільтрів – за допомогою гарячих клавіш Ctrl+L та Ctrl+M викликається меню редагування вмісту чорного на білого (шкала рівнів), та кольорів (шкала кривої). Для більш детальної корекції роботи використовується вкладка корекція.

Попри те що Adobe Photoshop недешеве програмне забезпечення, його освоєння в подальшому гарантуватиме якість та успішність продукту. Також варто зазначити, що редактор підтримує використання графічного планшету, при розробці ассетів використовувався графічний планшет – 10moons 8192.

1.6 Adobe Illustrator

Для розробки інтерфейсу гри буде використано відомий редактор векторної графіки – Adobe Illustrator.

На рисунку 1.6 зображено процес розробки інтерфейсу користувача за допомогою Adobe Illustrator.

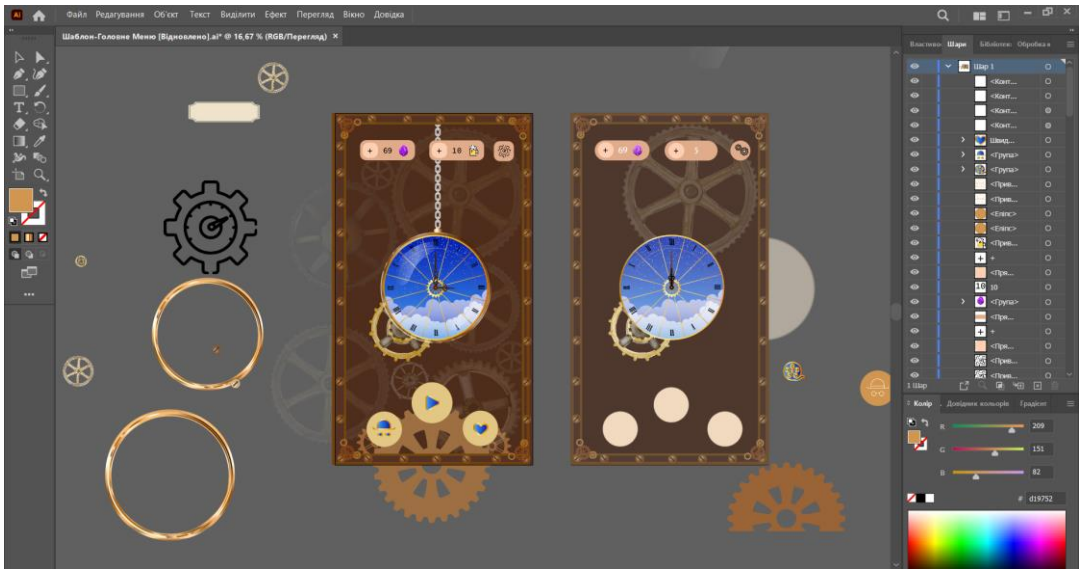

Рисунок 1.6 – Процес роботи у Adobe Illustrator

Adobe Illustrator пропонує набір різноманітних інструментів, які можна використовувати для розробки інтерфейсу гри. У даному проєкті найчастіше використовувалися такі інструменти як:

– інструмент «Shape», який дозволяє швидко створювати геометричні форми, такі як прямокутники, кола, трикутники та багатокутники.

– «Pen Tool» – дозволяє створювати складні векторні шляхи та форми.

– Інструменти для форматування тексту, вирівнювання, заливки та виділення.

– «Gradient» (градієнт), що дозволяє створювати плавні переходи між кольорами та створювати реалістичні тіні та відблиски

– «Layers» (шари), який дозволяє організовувати графічні елементи на різних шарах. Це дозволяє дизайнерам легко керувати розташуванням та

взаємодією елементів інтерфейсу, а також робити зміни безпосередньо на потрібному шарі, не впливаючи на інші елементи[1].

Adobe Illustrator це потужний інструмент для розробки інтерфейсу гри, завдяки своїм різноманітним інструментам та можливостям. Він надає художникам можливість створювати естетично привабливі, функціональні та інтуїтивно зрозумілі інтерфейси, що сприяють гарному геймплею та задоволенню користувачів

1.7 Leonardo.ai

Однією із особливостей даної кваліфікаційної роботи є використання нової технології у цифровому світі, а саме використання генерації картинок за допомогою штучного інтелекту.

Штучний інтелект (ШІ) - це науковий напрямок, що досліджує розробку комп'ютерних систем, здатних виконувати завдання, які вимагають інтелектуальних здібностей людини. ШІ дозволяє створювати системи, здатні розпізнавати образи, розуміти мову, вирішувати складні проблеми та передбачати наслідки дій [17].

Хоча на цю тему у суспільстві ходять суперечки, важко заперечувати той факт, що штучний інтелект заощаджує надзвичайно велику кількість сил та часу на виконання певного завдання. Для даного проєкту було подано заявку на доступ до бета–тесту штучного інтелекту під назвою Leonardo.ai.

Leonardo AI - це інноваційна платформа, що відкриває безліч можливостей для створення різноманітних застосувань штучного інтелекту. Платформа розробляється і підтримується компанією OpenAI, яка є одним з провідних лідерів у сфері розробки інтелектуальних систем.

На рисунку 1.7 зображено процес роботи із штучним інтелектом Leonardo.ai, на прикладі генерації сцени спальні головної героїні.

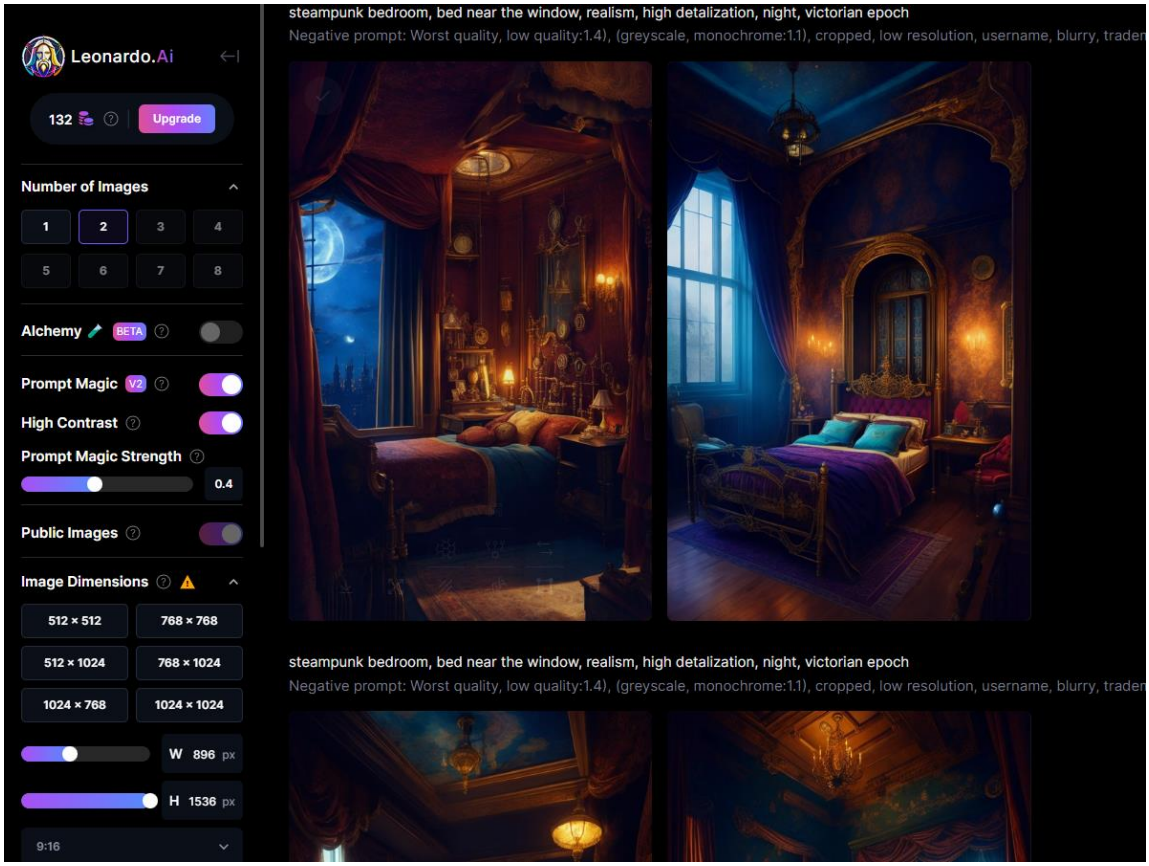

Рисунок 1.7 – Інтерфейс сайту Leonardo.ai

Штучний інтелект використовує навчальні алгоритми та нейронні мережі, щоб створювати вражаючі візуальні шедеври лише за допомогою текстового опису бажаної роботи.

Особливістю Leonardo.ai є можливість взаємодіяти зі штучним інтелектом та налаштовувати його параметри. Користувачі можуть вказати вхідні дані, такі як кольорову палітру, контур чи композицію, і Leonardo.ai врахує ці параметри при генерації картинки[27].

1.8 Висновок до першого розділу

В першому розділі кваліфікаційної роботи було розкрито поняття «мобільна гра» та «візуальна новела». Досліджено жанрові особливості, основні аспекти на яких заснована розробка ігор, проведений огляд конкурентів, а також перспективи даної сфери. Визначено основний порядок дій інді–розробника, і на

основі плану було здійснено порівняльну характеристику найбільш актуальних ігрових рушіїв та стилістик. У результаті було визначено, що для створення візуального роману «Механічне серце», використовуватиметься ігровий рушій Unity.

Також після огляду усіх варіантів графік для даного проекту візуальна частина буде виконана, за допомогою растрової графіки – зображення персонажів, фонів, інструментів та векторної – інтерфейс користувача.

Для розробки візуальної частини гри використовуватимуться програми Adobe Photoshop та Illustrator, а також зовсім новий спосіб генерації графічного контенту – штучний інтелект Leonardo.ai.

РОЗДІЛ 2. РЕАЛІЗАЦІЯ СЮЖЕТНО–РОЛЬОВОЇ ГРИ «МЕХАНІЧНЕ СЕРЦЕ»

2.1 Розробка концепту та вмісту прологу історії

Перед початком розробки візуальної новели, необхідно визначитися з контекстом, в якому відбуватимуться події історії, оскільки це впливає на створення графіки, персонажів, фонів та відповідний підбір шрифтів для створення загальної атмосфери.

Отже, історія розгортається в фантастичному світі, в якому поєднуються стилістика стімпанку та епохи вікторіанської Англії кінця 19 – початку 20 століття.

Стімпанк – це під жанр фантастики, що заснований у вигаданому світі, де парові машини замінюють електроніку та комп'ютери[16].

Завданням демо–версії новели для даної кваліфікаційної роботи – показати пролог історії, зацікавити потенційного гравця та ознайомити його з головною героїнею. Пролог розповідає про подорож дівчини поїздом, на яку вона давно мріяла, разом із незнайомкою. Під час короткого діалогу головна героїня помічає схожість з незнайомкою, але потім розуміє, що це вона сама. "Незнайомка" постійно нагадує головній героїні, що треба прокинутися і закрити двері, інакше станеться лихо. Поступово цей діалог переходить у сварку, яку «незнайомка» перериває клацнувши пальцями, і героїня прокидається у своїй опочивальні. Усвідомлюючи, що за вікном її спостерігають люди голового антогоніста історії Понтифіка, Головна героїня наказує механічному роботу негайно закрити двері, і сцена завершується.

Незважаючи на короткість, пролог має достатню міру загадковості, його мета полягає в тому, щоб зацікавити читача і збудити його бажання дізнатися, що станеться далі в історії. Крім того, за допомогою цього фрагмента історії можна з'ясувати, які елементи гри необхідно розробити, зокрема: Хоч пролог досить короткий, але достатньо інтригуючий, його завдання викликати у читача

дізнатися що буде далі по історії. Також завдяки цій частинці історії можна зрозуміти які елементи гри нам необхідно розробити, а саме:

– Персонажі (Leanardo.ai та Adobe Photoshop) – Головна героїня(емоції щастя, смутку, здивування, стурбованості та хитрості), «Таємнича незнайомка» (емоції спокою, смутку стурбованості та хитрості), Гвардієць Понтифіка (планується що даний персонаж матиме маску – емоції не потрібні) та Тінь механічного робота.

– Фони (Leanardo.ai та Adobe Photoshop) – кабіна поїзда, спальня головної героїні, двері майстерні.

– Дизайн інтерфейсу (Adobe Illustrator) – фон, рамка, годинник, діалогове вікно, кнопки налаштування та запуску історії.

Шрифт для основної історії – Academia Libera, а для назви історії – Melon Honey.

Отже, сформувавши план розробки, узгодивши список необхідних спрайтів та основної стилістки та сюжету, можна приступати до розробки дизайну та налаштування Unity.

2.2 Скачування та налаштування Unity

Перш за все слід провести налаштування ігрового рушія Unity. Для початку роботи із даним фреймворком спершу необхідно завантажити головний лаунчер – Unity Hub[19]. У ньому міститься всі необхідні елементи для гейм–розробника, такі як: вікно із всіма доступними проєктами (даними про їх версію, та файлове розміщення), доступ до Unity Learn, де можна безкоштовно переглядати освітні відео для підвищення кваліфікації та до спільноти розробників. Але найголовніше це контроль доступних версій рушія, тут містяться дані по кожну нову версію, як довготривалої підтримки так і бета–версії.

При розробці даної кваліфікаційної роботи було обрано версію 2021.3.22f, яка на момент написання роботи є найактуальнішою та найстабільнішою у роботі. Також при скачуванні версії необхідно вказати, що треба завантажити

пакет білдингу та розробки на ОС Android – Android Build Support, і при потребі документацію

На рисунку 2.1 зображено вибрані аспекти для розробки мобільної візуальної новели.

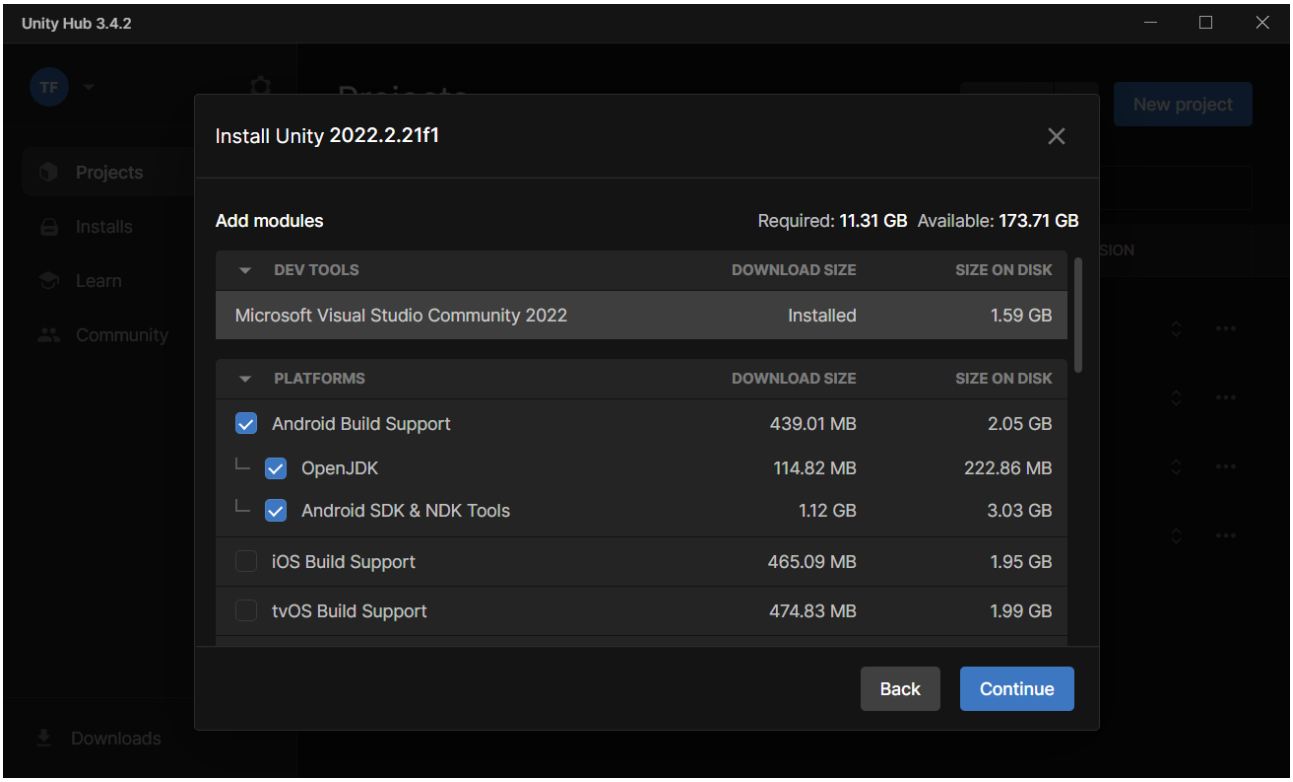

Рисунок 2.1 – Налаштування версії у Unity Hub

Створивши новий проект спершу треба провести налаштування проекту для розробки на ОС Android, для цього необхідно перейти у вкладку File/Build Settings і у відкритому вікні у вкладці Platforms, обрати Android та натиснути на кнопку Switch Platform. Після цього Unity імпортує всі необхідні бібліотеки та параметри для роботи із даною платформою.

Наступним кроком буде вибір основного розширення та підключення зовнішнього пристрою (у даному випадку смартфон Xiaomi Redmi Note 7). Стандартом розширення за багатьма літературними та інтернет джерелами для телефонів вважається 1920 на 1080. І на початковому етапі гра розроблялася саме у даному розширенні, але у подальшій роботі стало зрозуміло що дана

інформація застаріла, і більшість сучасних телефонів мають розширення екрану 2960 на 1440. Також при зміні формату, стало набагато легше розробляти адаптивний дизайн гри, для інших розширень платформи Android.

На рисунку 2.2 зображено вигляд головного меню гри при розширенні екрану 2960 на 1440 та 1920 на 1080.

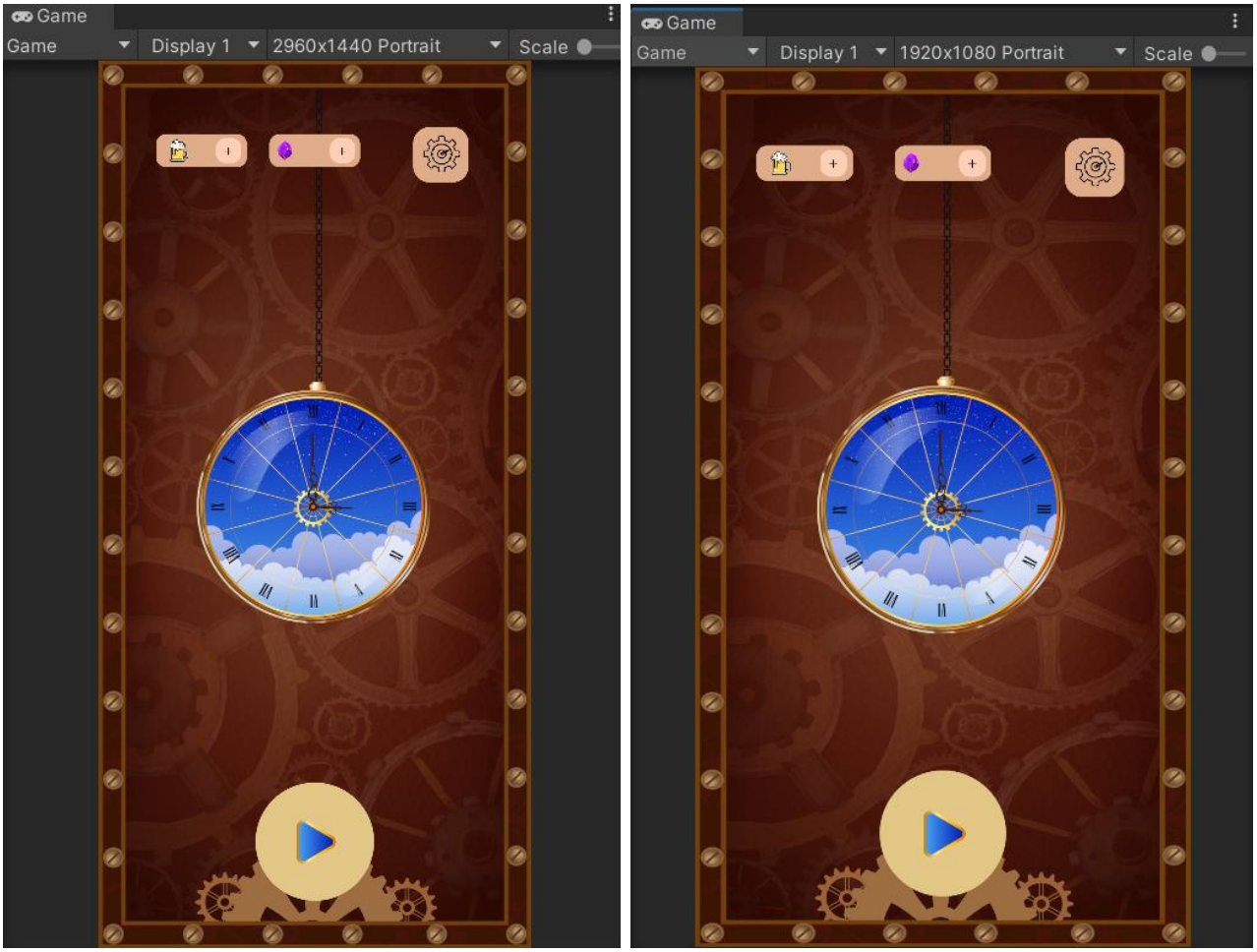

Рисунок 2.2 – Адаптивність інтерфейсу гри

Останнім пунктом налаштування рушія для роботи із мобільними пристроями буде підключення технології емулятора Unity Remote 5[21].

Unity Remote 5 це мобільний додаток який дозволяє перевіряти робочу здатність гри безпосередньо при етапі розробки. Для того аби смартфон став тестувальним пристроєм необхідно зробити певні кроки, а саме:

1. Завантажити та встановити Unity Remote 5 на Android–пристрій з Google Play Store.

2. Відкрити проект в Unity. У головному меню перейти до вкладки «Edit» (Редагувати) > «Preferences» (Налаштування).

3. У вікні налаштувань вибрати «External Tools» (Зовнішні інструменти).

4. У розділі «Android» знайти поле «Unity Remote» і вибрати «Unity Remote 5» зі списку випадаючих пунктів.

5. Підключити Android–пристрій до комп'ютера за допомогою USB–кабелю.

6. На Android–пристрої відкрити Unity Remote 5.

7. На комп'ютері у Unity Editor натиснути кнопку «Play» (Відтворити).

8. Unity Editor повинен автоматично виявити підключений пристрій, і гра буде запущена на пристрої.

У результаті, ігровий рушій Unity буде повністю налаштований під розробку для операційної системи Android, і можна приступати безпосередньо до роботи.

2.3 Розробка сюжетно–орієнтованої гри

Після розробки візуальної частини гри, налаштування ігрового рушія настає найважливіший та найважчий етап – розробка ігрових механік, та зібрання усіх елементів гри воєдино.

2.3.1 Організація ієрархії каталогів.

Кожен проект Unity включає значну кількість різноманітних каталогів, елементів, об'єктів, матеріалів і т.д. З метою забезпечення зручності роботи кожний розробник, при створенні нової гри на цій платформі, прагне належним чином організувати своє робоче середовище. Більша частина інформації про проект та його роботу буде зосереджена в каталозі Assets, де імпортуються всі необхідні пакети для розробки гри, такі як TextMeshPro, Visual Scripting, Cinemachine і т.д.

Однак, для ясного розмежування файлів, які будуть створені під час виконання завдань, ми створимо порожній каталог з назвою «[Main]». В цьому каталозі будуть знаходитись всі ключові та додаткові елементи візуальної новели, такі як лістинги, асети, аудіо доріжки анімацій і т.д. Для кожного із цих елементів створимо окремий каталог із відповідною назвою.

Каталог «Animations» містить анімації персонажів.

«Configuratons» – зберігає два скриптовані об'єкти, які містять дані про дійових персонажів та загальні характеристики системи діалогів. Докладніше про цей каталог буде описано у наступних пунктах.

«Resources» – містить всі графічні компоненти новели, такі як звуки, музика, персонажі, інтерфейс, текстові документи тощо.

«Scenes» містить всі сцени проекту, про які буде згадано в наступному пункті.

«Scripts» – містить повний перелік коду гри.

І останній у цьому списку каталог «Shaders» – містить об'єкти шейдерів, які відповідають за зміну фонового зображення у грі.

На рисунку 2.3 зображено остаточний вигляд каталогу [Main]

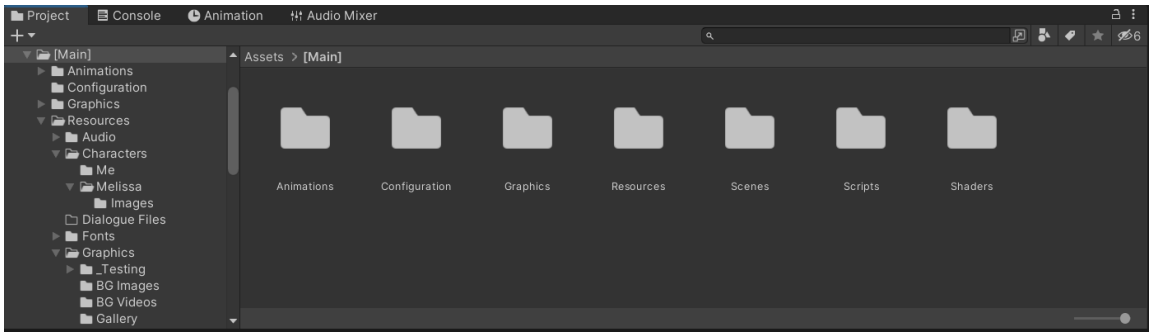

Рисунок 2.3 – Вигляд каталогу [Main]

Наступним кроком буде створення діючих сцен гри.

Сцена виступає як основний модуль, що визначає організацію об'єктів у пам'яті. У сценах містяться об'єкти, з яких складається гра. У базовому варіанті,

сцена представляє окремо взятий рівень гри, де завантажується лише одна сцена в будь–який момент часу.

2.3.2 Створення основних сцен та реалізація їх контролю.

Сцена виступає як основний модуль, що визначає організацію об'єктів у пам'яті. У сценах містяться об'єкти, з яких складається гра. У базовому варіанті, сцена представляє окремо взятий рівень гри, де завантажується лише одна сцена в будь–який момент часу[20].

Зазвичай коли користувач заходить у нову гру, йому показують так званий Splash Screen, або простими словами сцену завантаження.

Splash Screen – це екран, який з'являється, при відкриті додатка на мобільному пристрої. Іноді це називається екран запуску або початковий екран. Коли завантаження завершиться, користувач переходить на більш функціональний екран(у нашому випадку це головне меню), де можна виконати дії[8]. Тобто основним завданням даної сцени є відображення завантаження гри, і повинна містити: головне зображення стартового екрану, надпис назви гри і блок завантаження.

Для створення нової сцени заходимо у каталог «Scenes», натискаємо праву кнопку мишки, вікно Create > Scenes, і називаємо сцену відповідно. При найменуванні кожного об’єкта у Unity, варто пам’ятати що платформа запам’ятовує тільки першу назву об’єкта, і при перейменуванні можуть виникати помилки та проблеми, аби цього уникнути бажано зразу вказувати правильне ім’я без граматичних помилок чи жартів. Особливо це стосується ініціалізації коду.

Далі варто розібратися із панеллю «Ієрархія».

Ієрархія – це вікно, яке відображає ієрархічну структуру всіх об'єктів у проєкті. Вона показує взаємозв'язки між об'єктами і дозволяє легко керувати їхнім розташуванням, ієрархією батьківських та дочірніх відносин, а також змінювати їх властивості та компоненти[22].

У ієрархії сцени «Splash Screen», створюємо елемент «Canvas» (від англ. – полонто) – компонент, який використовується для створення 2D або 3D графічного простору, на якому відображаються всі інші елементи інтерфейсу в грі. Він визначає область, на якій можна розміщувати кнопки, тексти, зображення та інші UI елементи, і розміщуємо у ньому основні елементи сцени.

Основною «фішкою» даної сцени є панель завантаження, де показується скільки залишилося відсотків до запуску гри. Це реалізується за допомогою лістинг під назвою Loading (див. лістинг 1.1):

Лістинг 2.1 – Код Loading завдання якого полягає у створенні процесу завантаження гри з відображенням прогресу на заповнювачі FillImage та переходом до наступної сцени після завершення завантаження.

```
using UnityEngine;
using UnityEngine.SceneManagement;
using UnityEngine.UI;

public class Loading : MonoBehaviour
{
    public Image FillImage;
    float time, second;
    [SerializeField]
    void Start()
    {
        second = 5;
        Invoke("LoadGame", 5f);
    }
    void Update()
    {
        if (time < 5)
        {
            time += Time.deltaTime;
            FillImage.fillAmount = time / second;
        }
    }
    public void LoadGame()
    {
        SceneManager.LoadScene(1);
    }
```

На рисунку 2.4 зображено фінальний вигляд та ієрархію сцени Splash Screen.

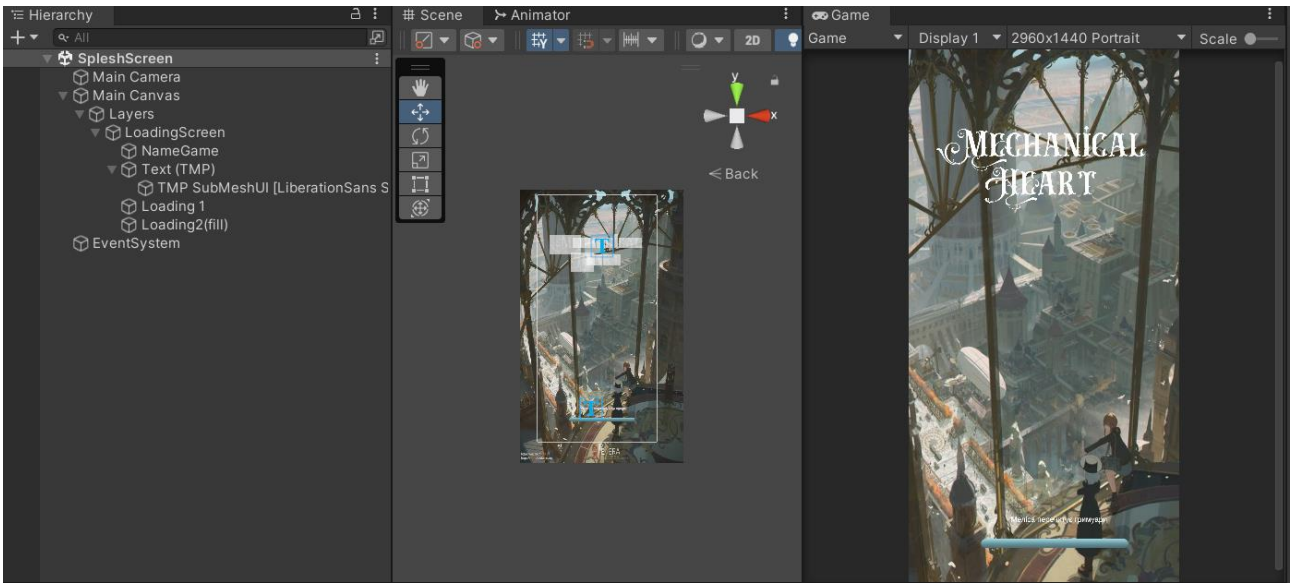

Рисунок 2.4 – Сцена Splash Screen.

Тим же способом ми створюємо сцени «MainMenu» та «VisualNovel», проте з деякими особливостями. «MainMenu» використовує кнопку запуску для переключення між сценами, на відміну від автоматичної зміни, яка відбувається на Splash Screen за замовчуванням. Крім того, вона включає вікно «Settings», що відповідає за загальні налаштування гри (наприклад, управління музикою, зв'язок з розробником у соціальних мережах та повідомлення про помилки).

Найважливіша сцена – «VisualNovel», в якій відбуваються всі події гри. Ієрархія цієї сцени може бути складною для розуміння, але вона добре продумана.

Як зображено на рисунку 2.5, ієрархія сцени складається з пустих об'єктів:

– [TESTING]: об'єкт, який містить тимчасові елементи для перевірки правильності роботи коду, системи та виявлення помилок.

– [VN CONTROLLER]: основний об'єкт гри, що включає камеру, полотно та об'єкт LAYERS.

– LAYERS: відображає принцип роботи шарів, які застосовуються у програмі Photoshop. Кожен наступний елемент відображається поверх попереднього, що дозволяє керувати відображенням елементів гри без

використання координати Z. У об'єкті LAYERS містяться елементи, такі як Background (задній фон гри), Characters (діючі персонажі) та Dialogue (панель діалогу та текст історії).

– [Managers]: об'єкти, що містять перевірені та функціональні скрипти, необхідні для стабільної роботи гри.

На рисунку 2.5 зображено фінальний вигляд та ієрархію сцени VisualNovel.

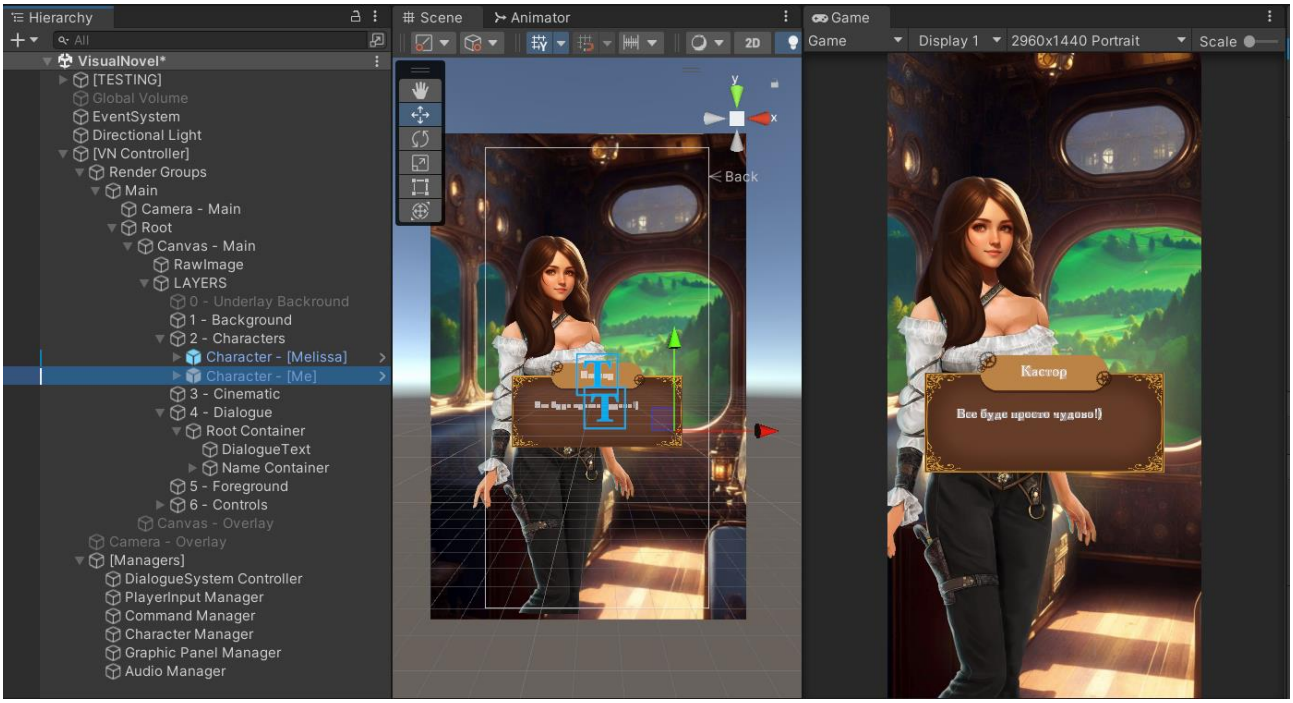

Рисунок 2.5 – Сцена VisualNovel.

Створивши усі сцени необхідно організувати їхню зміну. Схема зміни сцен наступна: спершу повинна появитися сцена Splash Screen(це реалізоується за допомогою лістинга Loading (див. лістинг 1.1), після завантаження, відбувається автоматичний перехід на сцену MainMenu, тут гравець може відкрити вікно налаштувань, або ж натиснувши на кнопку «Пуск» переходить на сцену Visual Novel. Для цього напишемо невеличкий лістинг від назвою «SceneCh»:

```
public class SceneCh : MonoBehaviour
{
    public void LoadScene(int sceneid)
    {
        SceneManager.LoadScene(sceneid);
```

```
        }
    }
```

Накладаємо цей скрипт, на пустий об’єкт Button, сцени Main Menu. На кнопці «Play», додаємо подію OnClick, у ввідне поле під RunTime Only, переносимо пустий об’єкт Button, і з випадаючого списку вибираємо скрипт SceneCh. Останнє, що треба зробити аби все запрацювало, ввести id сцени, на яку кнопка переключатиме гравця. Для того щоб взнати id сцени необхідно зайти у вкладку Build and Settings.

Таким чином ми створили усі необхідні сцени гри, налаштували їх перехід одна між одною, розробили зрозумілу ієрархію об’єктів і можемо переходити до написання самої візуальної новели.

2.3.3 Розробка механік гри

У цьому пункті розглянуто основні функціональні елементи гри та їхній принцип роботи. В загальному можна сказати що даний проєкт стоїть на так званих трьох черепахах: Character (або Speaker) (він же ш відображений персонаж, той хто говорить репліку), Dialogue (відображуваний текст історії) та Command(спосіб керування елементами гри). У додатку А подано повноцінний лістинг усіх розглянутих нижче скриптів.

У даній кваліфікаційній роботі буде розглянуто 3 основних контролерів роботи гри, а саме: «ConversationManager», «CharacterSpriteLayer» та «Command Manager», та докладно описано що саме виконує той чи інший клаптик коду.

Спершу розглянемо «ConversationManager» (див. Додаток А).

ConversationManager, відповідає за керування діалоговими розмовами у грі. Він обробляє кожен рядок розмови, виконує логіку виведення діалогу та команд, а також очікує ввід користувача, якщо це потрібно.

У корутині RunningConversation відбувається циклічна обробка кожного рядка розмови. Якщо рядок порожній, він пропускається. Виконується логіка

виведення діалогу та команд для кожного рядка, а також очікується ввід користувача у випадку наявності діалогу для відображення.

Корутини Line_RunDialogue та Line_RunCommands відповідають за відображення діалогу та виконання команд відповідно. Вони викликають додаткові функції та корутини для виконання специфічних операцій.

BuildLineSegments та BuildDialogue відповідають за побудову тексту діалогу. Вони крок за кроком додають текст та очікують його завершення. WaitForUserInput очікує ввід користувача. Він перевіряє змінну userPrompt, яка встановлюється в true при події вводу користувача.

Цей скрипт використовує інші класи та компоненти, такі як DialogueSystem(основний контролер для старту і контролю діалогу), TextArchitect, CommandManager та CharacterManager, для забезпечення різних функціональних можливостей у діалогових розмовах.

На рисунку 2.6 зображено Dialogue System Controller.

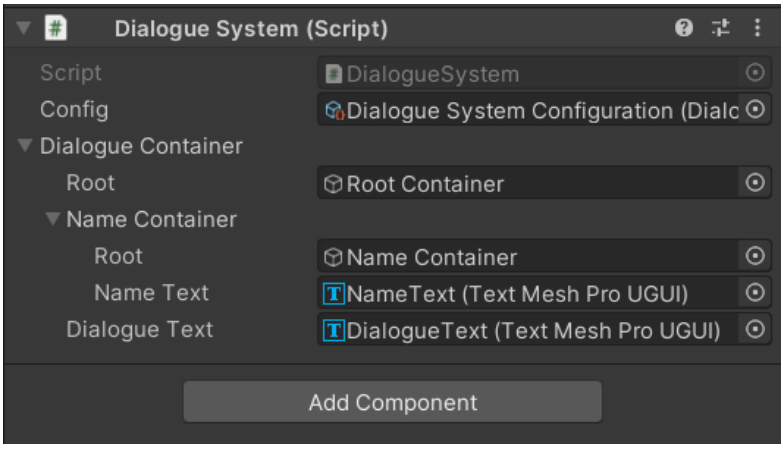

Рисунок 2.6 – DialogueSystem Controller

Поєдання із самим проєктом в Unity відбувається у пустому об’єкті Dialogue System Controller, за допомогою DialogueSystem, де створюється екземеляр класу ConversationManager, із вже існуючим Root Container в сцені.

Наступний скрипт який розглядатиметься буде «CharacterSpriteLayer» (див. Додаток Б).

Цей код представляє клас CharacterSpriteLayer , який відповідає за управління спрайтами персонажів у шарі графічного інтерфейсу (GUI). Основні функції цього класу включають:

– Установка спрайту для шару персонажа за допомогою методу SetSprite .

– Перехід до нового спрайту з плавною анімацією за допомогою методу TransitionSprite .

– Перехід до нового кольору спрайту з плавною анімацією за допомогою методу TransitionColor .

– Зміна кольору спрайту за допомогою методу SetColor .

– Обертання спрайту вліво або вправо з плавною анімацією за допомогою методів FaceLeft , FaceRight і Flip .

Клас CharacterSpriteLayer містить різні поля та приватні змінні, такі як transitionSpeedMultiplier для налаштування швидкості переходу, oldRenderers для зберігання старих спрайтів, co_transitioningLayer , co_levelingAlpha , co_changingColor та co_flipping для зберігання посилань на активні асинхронні функції та інші.

Цей клас використовує CharacterManager для запуску та зупинки корутин (асинхронних функцій). Він також використовує клас Image з Unity для відображення спрайтів персонажів та CanvasGroup для керування прозорістю спрайтів.

У коді також присутні різні допоміжні методи, такі як CreateRenderer , TryStartLevelingAlphas , RunAlphaLeveling і т.д., які допомагають виконувати необхідні операції для зміни спрайтів, кольорів і орієнтації персонажів.

На рисунку 2.7 зображено роботу класу CharacterSpriteLayer, яки відображає персонажів з правої екрану та з лівою почергово, що дає ілюзію діалогу.

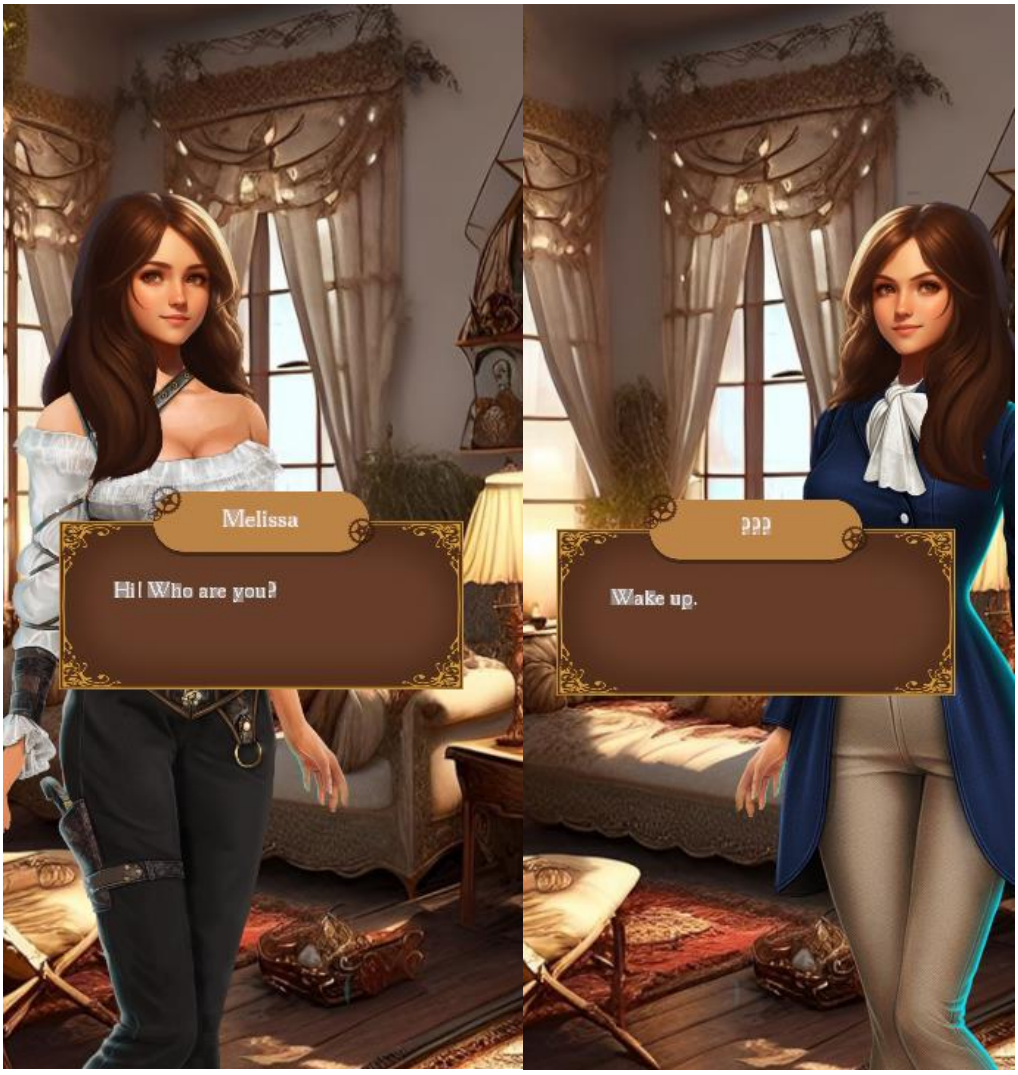

Рисунок 2.7 – Результат роботи системи Characters

І останній скрипт який буде докладно описаний, це скрипт створений для керування процесами у грі, а саме – Command Manager (див. Додаток В).. Завдяки спеціальному вводу у текстовий документ, менеджер розумітиме, що повинен відображати не екрані куди повинні рухатися персонажі, або який фон повинен зараз з’явитися перед гравцем. Розглянемо основні елементи коду:

Awake(): виконується під час запуску гри та ініціалізує CommandManager. Він створює об'єкт CommandDatabase, завантажує розширення бази даних та налаштовує початкові значення.

Execute(string commandName, params string[] args): виконує команду за її назвою commandName та переданими аргументами args. Він перевіряє, чи є команда в базі даних та запускає відповідний процес для виконання команди.

StartProcess(string commandName, Delegate command, string[] args): створює новий процес для виконання команди command з аргументами args. Він додає новий процес до списку активних процесів та запускає його.

RunningProcess(CommandProcess process): головним циклом процесу. Він очікує, доки процес не завершиться, і потім його закінчує.

WaitingForProcessToComplete(Delegate command, string[] args): очікує, доки команда command з аргументами args не завершить своє виконання.

AddTerminationActionToCurrentProcess(UnityAction action): додає дію action до поточного процесу, яка буде виконана при його завершенні.

CreateSubDatabase(string name): Цей метод створює базу даних з ім'ям name та повертає її для подальшого використання.

Даний проєкт містить набагато більшу кількість цікавих моментів, особливостей та рішень, але загальновідомі обмеження та рамки розміру даної кваліфікаційної роботи, не дають змогу описати кожен детально. Але нижче наведений список усіх лістингів гри та їхній коротенький опис.

TextArchitect –принцип відповідальності класу (class responsibility або CRC), для побудови та появи тексту динамічним шляхом.

DialogueSystem – основний контролер для старту і контролю діалогу та розмови на екрані.

DialogueContainer – діалоговий графічний дисплей, з усіма dialogue box елементами. Може бути зміненим для різного відображення інформації. Простими словами у ньому лежить всі елементи інтерфейсу які так чи інакше будуть залучені при виведені діалогу.

FileManager – займається збереженням, завантаженням, та шифруванням файлів у проєкті.

FilePath – хаб доступу(шлях) до файлу який містить дані діалогу та розташування каталогу.

DialogueParser – система, яка обробляє функцію розділення для перетворення рядка в рядок діалогу.

DIALOGUE_LINE – контейнер зберігання для інформації про діалог, яка була проаналізована та виділена із рядка.

PlayerInputManager – хаб для контролю вводу гравцем(при натиску на кнопку показується наступний елемент діалогу тощо).

ConversationManager – обробляє усю логіку виводу діалогу на екран.

NameContainer – логічний контролер, який є дочірнім елементом DialogueContainer, якому належить ім’я героя.

DL_COMMAND_DATA – контейнер даних який містить команди які використовуватимуться для діалогових стрічок.

DL_DIALOGUE_DATA – контейнер даних який містить усю інформацію про сегментацію діалогової стрічки.

DL_SPEAKER_DATA – контейнер даних який містить інформацію що відноситься до персонажів, та їх зв’язок із діалогом.

CommandManager – принцип відповідальності класу для перевірки та виконання спеціальних команд.

CMD_DatabaseExtension – клас який використовується для розширення доступних команд у CommandDatabase.

CommandDatabase – база даних усіх команд які є доступні для використання у CommandManager.

Character – клас у якому міститимуться усі види персонажів проєкту.

CharacterConfigData –контейнер даний який визначає параметри конфігурації для персонажів у візуальній новелі

Character ConfigSO – надає змогу створити scriptable object який містить усіх персонажів гри та їх базові параметри.

DialogueSystemConfiguration SO – також дає змогу створити scriptable object тільки тут вже містить налаштування діалогу

CharacterManager – центральний хаб який відповідає за створення отримання та управління персонажів у сцені.

CharacterSpriteLayer – відповідає за управління спрайтами персонажів у шарі графічного інтерфейсу (GUI).

AudioChannel – віртуальний канал для програвання та керування аудіо треками.

AudioManager – обробляє звукові ефекти, голоса, звуки ти музику.

AudioTrack – віртуальний контролер програє музику з аудіо каналу.

GraphicPanelManager – менеджер який контролює усю графічну панель, таку як задній фон та сінематік.

У результаті було розроблено демоверсію сюжетно–рольової візуальної новели, які містить у собі текстову архітектуру, систему контролю персонажами, аудіо контроль, та систему зміни заднього фону.

2.4 Висновок до другого розділу

В другому розділі кваліфікаційної роботи було описано як за допомогою ігрового рушія Unity було створено демоверсію сюжетно орієнтованої візуальної новели «Механічне серце».

Спершу було визначено основну концепцію гри, стилістику та коротко описано про що розповідатиме пролог демоверсії гри. Завдяки цьому був сформований список необхідних графічних елементів та за допомогою редакторів растрової та векторної графіки – Adobe Photoshop та Illustrator, і експериментального способу генерації картинок – штучного інтелекту Leonardo.ai розроблено візуальну частину гри.

Створено ігрові моделі персонажів, а також локації, де будуть відбуватися основні події відеогри. Перейшовши у ігровий рушій Unity спершу було поведено налаштування для розробки на ОС Android. Наступним кроком стала організація робочого простору, ініціалізація трьох головних сцен проекту та система їхнього контролю. Проведено налаштування та впорядкування головного каталогу «Assets». Та розроблено основні механіки візуальної новели, такі як: текстова архітектура , система контролю персонажів та базу даних команд, для контролю усієї гри загалом.

РОЗДІЛ 3. БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ, ОСНОВИ ОХОРОНИ ПРАЦІ

3.1. Вплив електромагнітного випромінювання від мобільних телефонів

В сьогоденних реаліях майже не можливо уявити своє життя без використання мобільного телефону. Завдяки ним ми завжди на зв’язку з рідними і близькими, дізнаємося останні новини світу або ж розслабляємося від тяжкого робочого дня граючи у мобільні ігри. Але багато користувачів не звертають уваги наскільки даний пристрій є потенційно небезпечним, і яких правили користування мобільними телефонами варто слідувати аби запобігти їхньому негативному впливу на організм.

Найбільшу небезпеку з фізіологічної точки зору яку можуть наносити такі пристрої є їхнє електромагнітне випромінювання.

Електромагнітне випромінювання(ЕМВ) – це форма енергії, що поширюється у вигляді хвиль електричного та магнітного поля [28]. Мобільні телефони випромінюють електромагнітне випромінювання високої частоти, відоме як радіочастотне випромінювання. Це випромінювання генерується антеною телефону та використовується для передачі і отримання сигналів зі стільникових мереж.

Електромагнітне випромінювання утворює електромагнітне поле (ЕМП) яке може впливати на організм людини залежно від різних факторів. Серед цих факторів належить зазначити характер поля, яке створюється на робочому місці, чи то високочастотне (ВЧ) чи низькочастотне (НВЧ), відстань від джерела випромінювання, тривалість дії, діапазон частот, інтенсивність випромінювання.

Для оцінки впливу опромінення ВЧ та НВЧ використовується величина інтенсивності електричного поля, яка вимірюється в вольтах на метр (В/м).

Поблизу джерел ВЧ полів формуються зони індукції і зони випромінювання, які розповсюджуються на різну відстань залежно від частоти.

На відстані, меншій ніж 1/6 довжини хвилі, переважають поля індукції і цю область можна вважати зоною індукції. На більш віддалених відстанях переважають поля випромінювання, і ця область вважається зоною випромінювання.

У зоні індукції людина піддається періодичному змінному електричному та магнітному полю. У зоні випромінювання людину оточує електромагнітне поле з однаковою та одночасно змінною електричною та магнітною компонентами.

ЕМП впливає на організм людини шляхом часткового поглинання енергії поля тканинами тіла. Високочастотні поля ВЧ ЕМП збуджують іони в тканинах, що призводить до виникнення високочастотних струмів і теплового ефекту поглинання енергії поля.

Провідність тканин пропорційна кількості рідини, що міститься в них. Кров та м'язи мають найбільшу провідність, а жирові тканини - найменшу. Товщина жирового шару в опроміненій ділянці тіла впливає на відбивання хвиль від поверхні тіла людини. Головний і спинний мозок мають незначний жировий шар, а очі взагалі не мають жиру, тому вони найбільше піддаються опроміненню.

Тривале та систематичне вплив ЕМП на працюючих може викликати зміни в організмі, такі як головний біль, порушення сну, підвищена втомленість, роздратованість, понижений кров'яний тиск, зміни в печінці та селезінці, підвищена температура тіла, випадіння волосся та ламкість нігтів.

У залежності від ступеня впливу ЕМП розрізняються 3 ступені ураження:

1. Легке ураження - характеризується тимчасовими функціональними змінами в організмі, які не вимагають тривалого лікування і не впливають на працездатність потерпілого.

2. Ураження середньої ступені - характеризується вираженими та стійкими порушеннями нервової системи, ендокринної системи та кровообігу.

3. Важкі ураження - рівень ураження, який не згадується в літературі, оскільки виробничі підприємства не допускають працювати людей, які мають навіть легке або середнє ураження.

Законодавство передбачає проведення попередніх та періодичних медичних оглядів, а також відбір працівників, які працюють з ЕМП, для забезпечення профілактики професійних захворювань [28].

Враховуючи те що даний пристрій перебуває постійно поряд із тілом людини, вплив електромагнітного випромінювання хоч і не великий, але має накопичувальну дію і може дати свої «результати» у майбутньому [5]. Аби мінімізувати ризики шкоди здоров'ю варто дотримуватися простих але важливих правил використання мобільних телефонів, а саме:

– Використовуйте безпровідні навушники. Це дозволить збільшити відстань між телефоном і головою, знизивши таким чином вплив електромагнітного поля на мозок та мінімізуючи можливу теплову експозицію.

– Використовуйте вбудовані функції зменшення випромінювання. Більшість сучасних мобільних телефонів мають функції зменшення випромінювання, такі як «режим польоту» або «економія енергії». Вони можуть допомогти знизити радіочастотне випромінювання, коли ви не активно використовуєте свій телефон.

– Уникайте носити телефон біля тіла. Намагайтеся тримати телефон подалі від тіла, особливо в кишені або прикріплений до пояса. Чим ближче телефон до вас, тим більша експозиція електромагнітному випромінюванню.

– Багато виробників мобільних телефонів надають рекомендації щодо використання їхніх пристроїв, щоб мінімізувати можливий вплив на здоров'я. Ознайомтесь з інструкціями користувача та дотримуйтесь рекомендацій, наданих виробником.

Узагалі, більшість досліджень показують, що рівень електромагнітного випромінювання від мобільних телефонів на даний момент знаходиться в межах безпечних норм. Однак, важливо продовжувати дослідження та моніторинг, щоб забезпечити безпеку користування мобільними телефонами в майбутньому.

3.2 Психофізіологічне розвантаження при застосуванні мобільного додатка «Механічне серце»

На сьогоднішній день майже кожна друга людина на планеті, проводить свій вільний час граючи у комп’ютерні та(або) мобільні ігри. Ігри забезпечують відчуття розслаблення, легкості, у якійсь мірі навіть втечі від реальності[30]. Але багато гравців забувають за те що через мірне проводження за іграми та безпосередньо за екранами та моніторами негативно впливає на організм як у фізіологічному плані так і у психічному.

Тому майбутнім гравцям мобільного додатку «Механічне серце», за для забезпечення збереження здоров’я, необхідно буде проводити психофізіологічне розвантаження при застосуванні додатка.

Основним правилом використання мобільних пристроїв це дотримання дистанції між зіницею ока та екраном телефону, дана відстань повинна бути не менше 40 сантиметрів. Таким чином очі втомлюватимуться не так швидко, що забезпечить менші ризики виникнення проблем із зором.

Також після кожних 30-40 хвилин використання пристрою необхідно робити невеличні перерви по 15 хвилин. Головна умова – не змінювати діяльність з телефона на комп’ютер або інший ґаджет[14].

Фізична активність також є важливим аспектом психофізіологічного розвантаження. Регулярні перерви на фізичну активність, такі як короткі прогулянки, вправи розтяжки чи йога, можуть допомогти зняти напруження з м'язів і зміцнити фізичне здоров'я. Фізична активність сприяє покращенню кровообігу, підвищенню енергетичного рівня і зниженню стресу.

Націлювання на зелене світло є ще одним способом психофізіологічного розвантаження. Дослідження показують, що спостереження за природою і зеленими ландшафтами має помітний позитивний ефект на наше самопочуття та знижує рівень стресу. Тому, рекомендується гравцям «Механічного серця» час

від часу відволікатися від гри і подивитися на зелене світло, виходячи на природу або спостерігаючи за рослинами.

Додатковими методами психофізіологічного розвантаження можуть бути слухання музики, медитація, глибоке дихання та інші техніки релаксації. Ці методи сприяють зниженню рівня стресу, поліпшенню психічного стану і загальному самопочуттю гравців.

У підсумку, психофізіологічне розвантаження є важливим елементом для забезпечення здорового геймінгу. Використання мобільного додатка «Механічне серце» може бути захоплюючим та емоційним досвідом, але необхідно пам'ятати про необхідність періодичного відпочинку. Зорові вправи, фізична активність, спостереження за зеленим світлом та інші методи психофізіологічного розвантаження можуть допомогти гравцям підтримувати збалансований стан та насолоджуватися грою з високим комфортом.

3.3 Висновок до третього розділу

У третьому розділі було досліджено вплив електромагнітного випромінювання від мобільних телефонів на користувачів. Та розглянуто методи мінімізації негативного впливу електромагнітного випромінювання на організм людини.

Щодо психофізіологічного розвантаження гравців мобільної гри «Механічне серце», важливо забезпечувати комфортні умови гри та використовувати техніки релаксації для зниження стресу. Подальші дослідження та спільні зусилля гравців, розробників ігор та організацій можуть сприяти поліпшенню безпеки та психофізіологічного добробуту у цифровому світі. Важливо підтримувати баланс між користуванням мобільними телефонами та здоровим способом життя.

ВИСНОВКИ

У результаті виконання даної кваліфікаційної роботи було розроблено демоверсію сюжетно-орієнтованої гру у жанрі візуальна новела «Механічне серце» за допомогою ігрового рушія Unity, графічних редакторів Adobe Photoshop та Adobe Illustrator, для растрової та векторної графіки відповідно, та використано експериментальний спосіб генерування зображень за допомогою штучного інтелекту – Leanardo.ai.

В першому розділі кваліфікаційної роботи освітнього рівня «Бакалавр»:

– Висвітлено поняття «мобільна гра», її жанрова різноманітність, та на прикладі відображено перспективність даної галузі розробки;

– Проведено огляд потенційних конкурентів, їх переваги та недоліки;

– Розглянуто найвідоміші та найактуальніші ігрові рушії для розробки візуальних новел: Unity, Unreal Engine, Godot та Ren`Py. Проведено їх порівняльну характеристику;

– Проаналізовано необхідне програмне забезпечення для створення візуальної частини гри.

В другому розділі кваліфікаційної роботи:

– Сформульовано основну ідею, концепцію, стилістику та сюжет прологу історії гри;

– Створено інтерфейс користувача, три основні сцени, та систему організації їх у проєкті;

– Розроблено текстову архітектуру гри, організовано систему контролю персонажів та базу даних команд, для контролю усієї гри загалом.

У розділі «Безпека життєдіяльності, основи охорони праці» розглянуто вплив електромагнітного випромінювання від мобільних телефонів на людину, та описано як реалізовується психоемоційне розвантаження користувачів мобільного додатку «Механічне серце», яке у подальшому враховувалося при розробці гри.

ПЕРЕЛІК ДЖЕРЕЛ

1. Adobe Help Center – Нові можливості в Illustrator. [Електронний ресурс] – Режим доступу до ресурсу:
https://helpx.adobe.com/ua/illustrator/using/whats-new.html

2. Adobe Help Center – Що нового у Photoshop. [Електронний ресурс] – Режим доступу до ресурсу: https://helpx.adobe.com/ua/photoshop/using/whats-new.html?promoid=2SLRC6G5&mv=other

3. Artstation – офіційна сторінка художника Le Vuong. [Електронний ресурс] – Режим доступу до ресурсу: https://www.artstation.com/vuogle

4. Avedon A. A. Sutton-Smith The Ambiguity of Play: Harvard University Press / Araf Araftar Avedon., 2019. – 288 c.

5. BBC News Україна – Випромінювання від смартфонів: чи це небезпечно і як захиститися [Електронний ресурс] – Режим доступу до ресурсу: https://www.bbc.com/ukrainian/features-43238017

6. Cases – Растрова та векторна графіка. [Електронний ресурс] – Режим доступу до ресурсу: https://cases.media/article/rastrova-ta-vektorna-grafika

7. Costikyan A. B. Uncertainty in Games: MIT Press / Aram Brel Costikyan., 2018. – 152 p.

8. Designtalk – Що таке Splash screen і як робити його дизайн правильно? [Електронний ресурс] – Режим доступу до ресурсу: https://designtalk.club/shho-take-splash-screen-jak-jogo-robyty/

9. Deviantart – офіційна сторінка художника shiningmelon [Електронний ресурс] – Режим доступу до ресурсу: https://www.deviantart.com/shiningmelon

10. Dribbble – офіційна сторінка художника John Mevis. [Електронний ресурс] – Режим доступу до ресурсу: https://dribbble.com/johnmevis

11. Gamedev.Dou – Що таке Unity? Курс Unity для митців. [Електронний ресурс] – Режим доступу до ресурсу: https://gamedev.dou.ua/forums/topic/38048/

12. GamesIndustry.biz – Genshin Impact makes $3.7 billion in mobile lifetime revenue. [Електронний ресурс] – Режим доступу до ресурсу: https://www.gamesindustry.biz/genshin-impact-makes-37-billion-in-mobile-lifetime-revenue

13. Hocking J. A. Unity in Action: Multiplatform Game Development in C# with Unity 5 1st Edition / Jared Antony Hocking., 2018. – 352 с.

14. Ideas Center – Користь і шкода комп'ютерних ігор, вплив на організм людини [Електронний ресурс] – Режим доступу до ресурсу – https://ideas-center.com.ua/?p=11042

15. Steam Store – сторінка гри Phoenix Wright: Ace Attorney Trilogy. [Електронний ресурс] – Режим доступу до ресурсу: https://store.steampowered.com/app/787480/Phoenix_Wright_Ace_Attorney_Trilogy/

16. Termin.in.ua – Стімпанк (Steampunk) – що це таке та в чому суть жанру і стилю. [Електронний ресурс] – Режим доступу до ресурсу: https://termin.in.ua/stimpank-steampunk/

17. Termin.in.ua – Штучний інтелект (ШІ) – що це таке, як працює і навіщо потрібен. [Електронний ресурс] – Режим доступу до ресурсу: https://termin.in.ua/shtuchnyy-intelekt/

18. Twitter – офіційна сторінка художника damaoye233. [Електронний ресурс] – Режим доступу до ресурсу: https://twitter.com/damaoye233

19. Unity – What is the Unity Hub? [Електронний ресурс] – Режим доступу до ресурсу: https://unity.com/download

20. Unity Documentation: – Scenes Manual 2021.3 (LTS) [Електронний ресурс] – Режим доступу до ресурсу: https://docs.unity3d.com/ru/530/Manual/CreatingScenes.html

21. Unity Documentation: – Unity Remote Manual 2021.3 (LTS) [Електронний ресурс] – Режим доступу до ресурсу: https://docs.unity3d.com/Manual/UnityRemote5.htm

22. Unity Documentation: – Unity The Hierarchy window Manual 2021.3 (LTS). [Електронний ресурс] – Режим доступу до ресурсу: https://docs.unity3d.com/Manual/Hierarchy.html

23. Voki Games – Action, MOBA, RTS: гайд жанрами мобільних ігор VOKI. [Електронний ресурс] – Режим доступу до ресурсу: https://vokigames.com/ua/action-moba-rts-gajd-zhanrami-mobilnih-igor/

24. Voki Games – Розробка ігор: 7 головних етапів створення мобільної free-to-play гри.[Електронний ресурс] – Режим доступу до ресурсу: https://vokigames.com/ua/rozrobka-igor-7-golovnyh-etapiv-stvorennya-mobilnoyi-free-to-play-gry/

25. Vyviska – Векторна VS растрова графіка? Спільне та відмінне між ними. [Електронний ресурс] – Режим доступу до ресурсу: https://vyviska.com.ua/vektorna-vs-rastrova-grafika-spilne-ta-vidminne-mizh-nymy/

26. Whitson J. R. The New Spirit of Capitalism in the Game Industry / Jason Rojer Whitson., 2019. –789 с.

27. Znaishov – Leonardo AI. Безкоштовна нейромережа для генерації зображення. [Електронний ресурс] – Режим доступу до ресурсу: https://znayshov.com/News/Details/leonardo_ai_bezkoshtovna_neiromerezha_dlia_heneratsii_zobrazhennia

28. БДМУ – Вплив електромагнітних полів (мобільні телефони, Wi-Fi мережі) на здоров'я людини [Електронний ресурс] – Режим доступу до ресурсу – https://www.bsmu.edu.ua/blog/1930-vplyv-electromagnitnyh-poliv/

29. Вікіпедія – Візуальна новела. [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Візуальна_новела

30. Радіо Свобода – Вплив смартфонів на рівень щастя, інтелекту та нове покоління «iGen» [Електронний ресурс] – Режим доступу до ресурсу – https://www.radiosvoboda.org/a/28705680.html

31. Фурсова Н. А. Розробка мережевої комп'ютерної гри з використанням Unity Engine / Н. А. Фурсова., 2018. – 244 с.

ДОДАТКИ

Додаток А

Програмний C# код для обробки виводу діалогової системи у гру «ConversationManager»

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

namespace DIALOGUE
{
    public class ConversationManager
    {
        private DialogueSystem dialogueSystem => DialogueSystem.instance;

        private Coroutine process = null;
        public bool isRunning => process != null;

        private TextArchitect architect = null;
        private bool userPrompt = false;

        public ConversationManager(TextArchitect architect)
        {
            this.architect = architect;
            dialogueSystem.onUserPrompt_Next += OnUserPrompt_Next;
        }

        private void OnUserPrompt_Next()
        {
            userPrompt = true;
        }

        public Coroutine StartConversation(List<string> conversation)
        {
            StopConversation();

            process =
dialogueSystem.StartCoroutine(RunningConversation(conversation));

            return process;
        }

        public void StopConversation()
        {
            if (!isRunning)
                return;

            dialogueSystem.StopCoroutine(process);
            process = null;
        }

        IEnumerator RunningConversation(List<string> conversation)
        {
            for (int i = 0; i < conversation.Count; i++)
            {
                //Dont show any blank lines or try to run any logic on them.
                if (string.IsNullOrWhiteSpace(conversation[i]))
                    continue;

                DIALOGUE_LINE line = DialogueParser.Parse(conversation[i])
```

```
                //Show dialogue
                if (line.hasDialogue)
                    yield return Line_RunDialogue(line);

                //Run any commands
                if (line.hasCommands)
                    yield return Line_RunCommands(line);

                if (line.hasDialogue)
                {
                    //Wait for user input
                    yield return WaitForUserInput();

                    CommandManager.instance.StopAllProcesses();
                }

            }
        }

        IEnumerator Line_RunDialogue(DIALOGUE_LINE line)
        {
            //Show or hide the speaker name if there is one present.
            if (line.hasSpeaker)
                HandleSpeakerLogic(line.speakerData);

            //Build dialogue
            yield return BuildLineSegments(line.dialogueData);
        }

        private void HandleSpeakerLogic(DL_SPEAKER_DATA speakerData)
        {
            bool characterMustBeCreated = (speakerData.makeCharacterEnter ||
speakerData.isCastingPosition || speakerData.isCastingExpressions);

            Character character =
CharacterManager.instance.GetCharacter(speakerData.name, createIfDoesNotExist:
characterMustBeCreated);

            if (speakerData.makeCharacterEnter && (!character.isVisible &&
!character.isRevealing))
                character.Show();

            //Add character name to the UI
            dialogueSystem.ShowSpeakerName(speakerData.displayname);

            //Now customize the dialogue for this character - if applicable

DialogueSystem.instance.ApplySpeakerDataToDialogueContainer(speakerData.name);

            //Cast position
            if (speakerData.isCastingPosition)
                character.MoveToPosition(speakerData.castPosition);

            //Cast Expression
            if (speakerData.isCastingExpressions)
            {
                foreach (var ce in speakerData.CastExpressions)
                    character.OnReceiveCastingExpression(ce.layer,
ce.expression);
            }
        }

        IEnumerator Line_RunCommands(DIALOGUE_LINE line)
        {
            List<DL_COMMAND_DATA.Command> commands = line.commandData.commands;
```

```
            foreach(DL_COMMAND_DATA.Command command in commands)
            {
                if (command.waitForCompletion || command.name == "wait")
                {
                    CoroutineWrapper cw =
CommandManager.instance.Execute(command.name, command.arguments);
                    while (!cw.IsDone)
                    {
                        if (userPrompt)
                        {
                            CommandManager.instance.StopCurrentProcess();
                            userPrompt = false;
                        }
                        yield return null;
                    }
                }
                else
                    CommandManager.instance.Execute(command.name,
command.arguments);
            }

            yield return null;
        }

        IEnumerator BuildLineSegments(DL_DIALOGUE_DATA line)
        {
            for(int i = 0; i < line.segments.Count; i++)
            {
                DL_DIALOGUE_DATA.DIALOGUE_SEGMENT segment = line.segments[i];

                yield return WaitForDialogueSegmentSignalToBeTriggered(segment);

                yield return BuildDialogue(segment.dialogue,
segment.appendText);
            }
        }

        IEnumerator
WaitForDialogueSegmentSignalToBeTriggered(DL_DIALOGUE_DATA.DIALOGUE_SEGMENT
segment)
        {
            switch(segment.startSignal)
            {
                case DL_DIALOGUE_DATA.DIALOGUE_SEGMENT.StartSignal.C:
                case DL_DIALOGUE_DATA.DIALOGUE_SEGMENT.StartSignal.A:
                    yield return WaitForUserInput();
                    break;
                case DL_DIALOGUE_DATA.DIALOGUE_SEGMENT.StartSignal.WC:
                case DL_DIALOGUE_DATA.DIALOGUE_SEGMENT.StartSignal.WA:
                    yield return new WaitForSeconds(segment.signalDelay);
                    break;
                default:
                    break;
            }
        }

        IEnumerator BuildDialogue(string dialogue, bool append = false)
        {
            //Build the dialogue
            if (!append)
                architect.Build(dialogue);
            else
                architect.Append(dialogue);

            //Wait for the dialogue to complete.
```

```
            while (architect.isBuilding)
            {
                if (userPrompt)
                {
                    if (!architect.hurryUp)
                        architect.hurryUp = true;
                    else
                        architect.ForceComplete();

                    userPrompt = false;
                }
                yield return null;
            }
        }

        IEnumerator WaitForUserInput()
        {
            while (!userPrompt)
                yield return null;

            userPrompt = false;
        }
    }
}
```

Додаток Б

Програмний C# код для управління спрайтами персонажів «CharacterSpriteLayer»

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

namespace CHARACTERS
{
    public class CharacterSpriteLayer
    {
        private CharacterManager characterManager => CharacterManager.instance;

        private const float DEFAULT_TRANSITION_SPEED = 3f;
        private float transitionSpeedMultiplier = 1;

        public int layer { get; private set; } = 0;
        public Image renderer { get; private set; } = null;
        public CanvasGroup rendererCG => renderer.GetComponent<CanvasGroup>();

        private List<CanvasGroup> oldRenderers = new List<CanvasGroup>();

        private Coroutine co_transitioningLayer = null;
        private Coroutine co_levelingAlpha = null;
        private Coroutine co_changingColor = null;
        private Coroutine co_flipping = null;
        private bool isFacingLeft =
Character.DEFAULT_ORIENTATION_IS_FACING_LEFT;
        public bool isTransitioningLayer => co_transitioningLayer != null;
        public bool isLevelingAlpha => co_levelingAlpha != null;
        public bool isChangingColor => co_changingColor != null;
        public bool isFlipping => co_flipping != null;

        public CharacterSpriteLayer(Image defaultRenderer, int layer = 0)
        {
            renderer = defaultRenderer;
            this.layer = layer;
        }

        public void SetSprite(Sprite sprite)
        {
            renderer.sprite = sprite;
        }

        public Coroutine TransitionSprite(Sprite sprite, float speed = 1)
        {
            if (sprite == renderer.sprite)
                return null;

            if (isTransitioningLayer)
                characterManager.StopCoroutine(co_transitioningLayer);

            co_transitioningLayer =
characterManager.StartCoroutine(TransitioningSprite(sprite, speed));

            return co_transitioningLayer;
        }

        private IEnumerator TransitioningSprite(Sprite sprite, float
speedMultiplier)
        {
```

```
            transitionSpeedMultiplier = speedMultiplier;

            Image newRenderer = CreateRenderer(renderer.transform.parent);
            newRenderer.sprite = sprite;

            yield return TryStartLevelingAlphas();

            co_transitioningLayer = null;
        }

        private Image CreateRenderer(Transform parent)
        {
            Image newRenderer = Object.Instantiate(renderer, parent);
            oldRenderers.Add(rendererCG);

            newRenderer.name = renderer.name;
            renderer = newRenderer;
            renderer.gameObject.SetActive(true);
            rendererCG.alpha = 0;

            return newRenderer;
        }

        private Coroutine TryStartLevelingAlphas()
        {
            if (isLevelingAlpha)
                return co_levelingAlpha;

            co_levelingAlpha =
characterManager.StartCoroutine(RunAlphaLeveling());

            return co_levelingAlpha;
        }

        private IEnumerator RunAlphaLeveling()
        {
            while (rendererCG.alpha < 1 || oldRenderers.Any(oldCG => oldCG.alpha
> 0))
            {
                float speed = DEFAULT_TRANSITION_SPEED *
transitionSpeedMultiplier * Time.deltaTime;

                rendererCG.alpha = Mathf.MoveTowards(rendererCG.alpha, 1,
speed);

                for (int i = oldRenderers.Count - 1; i >= 0; i--)
                {
                    CanvasGroup oldCG = oldRenderers[i];
                    oldCG.alpha = Mathf.MoveTowards(oldCG.alpha, 0, speed);

                    if (oldCG.alpha <= 0)
                    {
                        oldRenderers.RemoveAt(i);
                        Object.Destroy(oldCG.gameObject);
                    }
                }

                yield return null;
            }

            co_levelingAlpha = null;
        }

        public void SetColor(Color color)
        {
```

```
            renderer.color = color;

            foreach (CanvasGroup oldCG in oldRenderers)
            {
                oldCG.GetComponent<Image>().color = color;
            }
        }

        public Coroutine TransitionColor(Color color, float speed)
        {
            if (isChangingColor)
                characterManager.StopCoroutine(co_changingColor);

            co_changingColor =
characterManager.StartCoroutine(ChangingColor(color, speed));

            return co_changingColor;
        }

        public void StopChangingColor()
        {
            if (!isChangingColor)
                return;

            characterManager.StopCoroutine(co_changingColor);

            co_changingColor = null;
        }

        private IEnumerator ChangingColor(Color color, float speedMultiplier)
        {
            Color oldColor = renderer.color;
            List<Image> oldImages = new List<Image>();

            foreach (var oldCG in oldRenderers)
            {
                oldImages.Add(oldCG.GetComponent<Image>());
            }

            float colorPercent = 0;
            while (colorPercent < 1)
            {
                colorPercent += DEFAULT_TRANSITION_SPEED * speedMultiplier *
Time.deltaTime;

                renderer.color = Color.Lerp(oldColor, color, colorPercent);

                foreach (Image oldImage in oldImages)
                {
                    oldImage.color = renderer.color;
                }

                yield return null;
            }

            co_changingColor = null;
        }

        public Coroutine FaceLeft(float speed = 1, bool immediate = false)
        {
            if (isFlipping)
                characterManager.StopCoroutine(co_flipping);

            isFacingLeft = true;
```

```
            co_flipping =
characterManager.StartCoroutine(FaceDirection(isFacingLeft, speed, immediate));

            return co_flipping;
        }

        public Coroutine Flip(float speed = 1, bool immediate = false)
        {
            if (isFacingLeft)
                return FaceRight(speed, immediate);
            else
                return FaceLeft(speed, immediate);
        }

        public Coroutine FaceRight(float speed = 1, bool immediate = false)
        {
            if (isFlipping)
                characterManager.StopCoroutine(co_flipping);

            isFacingLeft = false;
            co_flipping =
characterManager.StartCoroutine(FaceDirection(isFacingLeft, speed, immediate));

            return co_flipping;
        }

        private IEnumerator FaceDirection(bool faceLeft, float speedMultiplier,
bool immediate)
        {
            float xScale = faceLeft ? 1 : -1;
            Vector3 newScale = new Vector3(xScale, 1, 1);

            if (!immediate)
            {
                Image newRenderer = CreateRenderer(renderer.transform.parent);

                newRenderer.transform.localScale = newScale;

                transitionSpeedMultiplier = speedMultiplier;
                TryStartLevelingAlphas();

                while (isLevelingAlpha)
                    yield return null;

            }
            else
            {
                renderer.transform.localScale = newScale;
            }

            co_flipping = null;
        }
    }
}
```

Додаток В

Програмний C# код для перевірки реалізації команд у грі «CommandManager»

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using System.Reflection;
using System.Linq;
using System;
using UnityEngine.Events;
using CHARACTERS;

namespace COMMANDS
{
    public class CommandManager : MonoBehaviour
    {
        private const char SUB_COMMAND_IDENTIFIER = '.';
        public const string DATABASE_CHARACTERS_BASE = "characters";
        public const string DATABASE_CHARACTERS_SPRITE = "characters_sprite";

        public static CommandManager instance { get; private set; }

        private CommandDatabase database;
        private Dictionary<string, CommandDatabase> subDatabases = new
Dictionary<string, CommandDatabase>();

        public List<CommandProcess> activeProcesses = new
List<CommandProcess>();

        private CommandProcess topProcess => activeProcesses.Last();

        private void Awake()
        {
            if (instance == null)
            {
                instance = this;

                database = new CommandDatabase();

                Assembly assembly = Assembly.GetExecutingAssembly();
                Type[] extensionTypes = assembly.GetTypes().Where(t =>
t.IsSubclassOf(typeof(CMD_DatabaseExtension))).ToArray();

                foreach (Type extension in extensionTypes)
                {
                    MethodInfo extendMethod = extension.GetMethod("Extend");
                    extendMethod.Invoke(null, new object[] { database });
                }
            }
            else
                DestroyImmediate(gameObject);
        }

        public CoroutineWrapper Execute(string commandName, params string[]
args)
        {
            if (commandName.Contains(SUB_COMMAND_IDENTIFIER))
                return ExecuteSubCommand(commandName, args);

            Delegate command = database.GetCommand(commandName);

            if (command == null)
                return null;
```

```
            return StartProcess(commandName, command, args);
        }

        private CoroutineWrapper ExecuteSubCommand(string commandName, string[]
args)
        {
            string[] parts = commandName.Split(SUB_COMMAND_IDENTIFIER);
            string databaseName = string.Join(SUB_COMMAND_IDENTIFIER,
parts.Take(parts.Length - 1));
            string subComandName = parts.Last();

            if (subDatabases.ContainsKey(databaseName))
            {
                Delegate command =
subDatabases[databaseName].GetCommand(subComandName);
                if (command != null)
                {
                    return StartProcess(commandName, command, args);
                }
                else
                {
                    Debug.LogError($"No command called '{subComandName}' was
found in sub database '{databaseName}'");
                    return null;
                }
            }

            string characterName = databaseName;
            //If we've made it here then we should try to run as a character
command
            if (CharacterManager.instance.HasCharacter(characterName))
            {
                List<string> newArgs = new List<string>(args);
                newArgs.Insert(0, characterName);
                args = newArgs.ToArray();

                return ExecuteCharacterCommand(subComandName, args);
            }

            Debug.LogError($"No sub database called '{databaseName}' exists!
Command '{subComandName}' could not be run.");
            return null;
        }

        private CoroutineWrapper ExecuteCharacterCommand(string commandName,
params string[] args)
        {
            Delegate command = null;

            CommandDatabase db = subDatabases[DATABASE_CHARACTERS_BASE];
            if (db.HasCommand(commandName))
            {
                command = db.GetCommand(commandName);
                return StartProcess(commandName, command, args);
            }

            CharacterConfigData characterConfigData =
CharacterManager.instance.GetCharacterConfig(args[0]);
            switch (characterConfigData.characterType)
            {
                case Character.CharacterType.Sprite:
                case Character.CharacterType.SpriteSheet:
                    db = subDatabases[DATABASE_CHARACTERS_SPRITE];
                    break;
```

```
            command = db.GetCommand(commandName);

            if (command != null)
                return StartProcess(commandName, command, args);

            Debug.LogError($"Command Manager was unable to execute command
'{commandName}' on character '{args[0]}'. The character name or command may be
invalid.");
            return null;
        }

        private CoroutineWrapper StartProcess(string commandName, Delegate
command, string[] args)
        {
            System.Guid processID = System.Guid.NewGuid();
            CommandProcess cmd = new CommandProcess(processID, commandName,
command, null, args, null);
            activeProcesses.Add(cmd);

            Coroutine co = StartCoroutine(RunningProcess(cmd));

            cmd.runningProcess = new CoroutineWrapper(this, co);

            return cmd.runningProcess;
        }

        public void StopCurrentProcess()
        {
            if (topProcess != null)
                KillProcess(topProcess);
        }

        public void StopAllProcesses()
        {
            foreach (var c in activeProcesses)
            {
                //Debug.Log($"Stop process '{c.command}({string.Join(',',
c.args)}) [term={c.onTerminateAction}]'");

                if (c.runningProcess != null && !c.runningProcess.IsDone)
                    c.runningProcess.Stop();

                c.onTerminateAction?.Invoke();
            }

            activeProcesses.Clear();
        }

        private IEnumerator RunningProcess(CommandProcess process)
        {
            yield return WaitingForProcessToComplete(process.command,
process.args);

            KillProcess(process);
        }

        public void KillProcess(CommandProcess cmd)
        {
            activeProcesses.Remove(cmd);

            if (cmd.runningProcess != null && !cmd.runningProcess.IsDone)
                cmd.runningProcess.Stop();

            cmd.onTerminateAction?.Invoke();
```

```
        }
        private IEnumerator WaitingForProcessToComplete(Delegate command,
string[] args)
        {
            if (command is Action)
                command.DynamicInvoke();
            else if (command is Action<string>)
                command.DynamicInvoke(args.Length == 0 ? string.Empty :
args[0]);
            else if (command is Action<string[]>)
                command.DynamicInvoke((object)args);
            else if (command is Func<IEnumerator>)
                yield return ((Func<IEnumerator>)command)();
            else if (command is Func<string, IEnumerator>)
                yield return ((Func<string, IEnumerator>)command)(args.Length ==
0 ? string.Empty : args[0]);
            else if (command is Func<string[], IEnumerator>)
                yield return ((Func<string[], IEnumerator>)command)(args);
        }

        public void AddTerminationActionToCurrentProcess(UnityAction action)
        {
            CommandProcess process = topProcess;

            if (process == null)
                return;

            process.onTerminateAction = new UnityEvent();
            process.onTerminateAction.AddListener(action);
        }

        public CommandDatabase CreateSubDatabase(string name)
        {
            name = name.ToLower();

            if (subDatabases.TryGetValue(name, out CommandDatabase db))
            {
                Debug.LogWarning($"A database by the name of '{name}' already
exists!");
                return db;
            }

            CommandDatabase newDatabase = new CommandDatabase();
            subDatabases.Add(name, newDatabase);

            return newDatabase;
        }
    }
}
```