Міністерство освіти і науки України
Національний технічний університет
«Дніпровська політехніка»

Інститут електроенергетики
(інститут)

Факультет інформаційних технологій
(факультет)

Кафедра Програмного забезпечення комп'ютерних систем
(повна назва)

ПОЯСНЮВАЛЬНА ЗАПИСКА
кваліфікаційної роботи ступеня

бакалавра
(назва освітньо-кваліфікаційного рівня)

студента *Пилипенко Кароліни Дмитрівни*
(ПІБ)

академічної групи *122-19-4*
(шифр)

спеціальності *122 Комп'ютерні науки*
(код і назва спеціальності)

освітньої програми *Комп'ютерні науки*
(назва освітньої програми)

на тему: *Розробка багаторівневої відеогри на основі кросплатформного рушія Unity*

Керівники	**Прізвище, ініціали**	**Оцінка за шкалою**		**Підпис**
		рейтинговою	**інституційною**	
кваліфікаційної роботи	*доц. Реута О.В.*			
розділів:				
спеціальний	*доц. Реута О.В*			
економічний	*проф. Вагонова О. Г.*			
Рецензент				
Нормоконтролер	*доц. Гуліна І. Г.*			

Дніпро
2023

Міністерство освіти і науки України
НТУ «Дніпровська політехніка»

ЗАТВЕРДЖЕНО:
завідувач кафедри
програмного забезпечення комп'ютерних систем
(повна назва)

М.О. Алексєєв
(підпис) (прізвище, ініціали)

« » 2023 року

ЗАВДАННЯ
на кваліфікаційну роботу

бакалавра
(назва освітньо-кваліфікаційного рівня)

студента *122-19-4* *Пилипенко К. Д.*
(група) (прізвище та ініціали)

тема кваліфікаційної роботи *Розробка багаторівневої відеогри на основі кросплатформного рушія Unity*

затверджена наказом ректора НТУ «ДП» від *16.05.2023* № *350-с*

Розділ	Зміст виконання	Термін виконання
Спеціальний	***На основі матеріалів проєктно-технологічної практики та інших науково-технічних джерел провести аналіз стану рішення проблеми та постановку задачі. Обґрунтувати вибір та здійснити реалізацію методів вирішення проблеми***	***13.05.2023 р.***
Економічний	***Провести розрахунок трудомісткості розробки програмного забезпечення, витрат на створення ПЗ й тривалості його розробки***	***27.05.2023 р.***

Завдання видав _____ *доц. Реута О.В.*
(підпис) (посада, прізвище, ініціали)

Завдання прийняв до виконання _____ *Пилипенко К.Д.*
(підпис) (прізвище, ініціали)

Дата видачі завдання: *14.01.2023 р.*

Термін подання кваліфікаційної роботи до ЕК: *12,06.2023 р.*

РЕФЕРАТ

Пояснювальна записка: 77 с., 33 рис., 0 табл., 3 дод., 23 джерел.

Об'єкт розробки: багаторівнева відеогра на основі кроссплатформного рушію Unity.

Мета кваліфікаційної роботи: розробка багаторівневої відеогри на основі кроссплатформного рушія Unity, яка буде зосереджена роботі з жанром квест-головоломка.

У вступі розглядається аналіз та сучасний стан проблеми, конкретизується мета кваліфікаційної роботи та галузь її застосування, наведено обґрунтування актуальності теми та уточняється постановка завдання.

У першому розділі проаналізовано предметну галузь, визначено актуальність завдання та призначення розробки, сформульовано постановку завдання, зазначено вимоги до програмної реалізації, технологій та програмних засобів.

У другому розділі проаналізовані наявні рішення, обрано платформи для розробки, виконано проектування і розробка програми, описана робота програми, алгоритм і структура її функціонування, а також виклик та завантаження програми, визначено вхідні і вихідні дані, охарактеризовано склад параметрів технічних засобів.

В економічному розділі визначено трудомісткість розробленої інформаційної системи, проведений підрахунок вартості роботи по створенню програми та розраховано час на його створення.

Практичне значення полягає у створенні ігрового додатка, що надає гравцю занурення у віртуальний світ, повний загадок та таємниць, цікавих головоломок та інтересної історії, створення унікального досвіду для кожного гравця.

Актуальність ігрового додатку визначається значним зростанням популярності комп'ютерних ігор, загальним стабільним та потужним розвитком ігрової індустрії, а також розвитком окремих ігрових жанрів, таких як квест-головоломка.

Список ключових слів: ІГРОВИЙ ДОДАТОК, ГРА, ВІДЕОГРА, ГРАВЕЦЬ, КВЕСТ, ГОЛОВОЛОМКА, КОРИСТУВАЧ, ІСТОРІЯ, ПОШУК ПРЕДМЕТІВ, ПЛАТФОРМА, UNITY, ІГРОВИЙ ДВИГУН.

ABSTRACT

Explanatory note: 77 pp., 33 fig. 0 table, 3 appendix, 23 sources.

Object of development: multi-level video game based on the cross-platform Unity engine.

The goal of the qualification work: development of a multi-level video game based on the cross-platform Unity engine, which will focus on working with the quest-puzzle genre.

In the introduction, the analysis and current state of the problem is considered, the purpose of the qualification work and the field of its application are specified, the justification of the relevance of the topic is given, and the statement of the task is clarified.

In the first section, the subject area is analyzed, the relevance of the task and the purpose of the development is determined, the task statement is formulated, and the requirements for software implementation, technologies and software tools are specified.

In the second section, available solutions are analyzed, platforms for development are selected, program design and development are performed, program operation, algorithm and structure of its functioning are described, as well as program calling and loading, input and output data are determined, and the composition of technical means parameters is characterized.

In the economic section, the labor intensity of the developed information system is determined, the cost of work on creating the program is calculated, and the time for its creation is calculated.

The practical value is in the creation a game application that immerses the player in a virtual world full of mysteries and secrets, different puzzles and an interesting story, creating a unique experience for each player.

The relevance of the game application is determined by the significant increase in the popularity of computer games, the general stable and powerful development of the game industry, as well as the development of certain game genres, such as quest-puzzle.

List of keywords: GAME APP, GAME, VIDEO GAME, PLAYER, QUEST, PUZZLE, USER, STORY, ITEM SEARCH, PLATFORM, UNITY, GAME ENGINE.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

UNITY – багатоплатформовий інструмент для розробки відеоігор і застосунків;

C# – мова програмування;

EXE – розширення виконуваного файлу, що застосовується в системах Microsoft Windows;

JSON – JavaScript Object Notation;

UI – User interface;

TMPro – Text Mesh Pro;

ЕОМ – електронно-обчислювальна машина.

ВСТУП

Комп'ютерні ігри – це інтерактивна форма розваги, в якій гравці взаємодіють із віртуальним світом, створеним на комп'ютері. Вони пропонують різноманітні жанри, від екшену та пригод до стратегій та головоломок.

Сьогодні ігри стали не лише популярним видом розваги, а й потужною економічною силою. Розробка комп'ютерних ігор включає широкий спектр діяльності, таких як дизайн, програмування, художнє оформлення, анімація, звуковий супровід і маркетинг. Команди розробників можуть містити десятки або навіть сотні людей, які працюють спільно для створення ігрових проектів. З розвитком технологій та доступності ігрових платформ, включаючи персональні комп'ютери, консолі, мобільні пристрої та віртуальну реальність, ігри стали більш реалістичними та інтерактивними. Індустрія також сильно впливає на інші сфери, такі як фільми, музика та мода, з інтеграцією персонажів та елементів ігор у ці галузі.

Саме тому темою даної кваліфікованої роботи є «Розробка багаторівневої відеогри на основі кроссплатформного рушію Unity». Мета цієї роботи – розробити ігровий додаток, який може перенести гравця у віртуальний світ та подарувати великий спектр емоцій, а також розвинути когнітивні навички.

Щоб отримати такий ефект, жанр для гри було вибрано квест-головоломку. Комп'ютерні ігри жанру квест-головоломка пропонують гравцям захоплюючу подорож, повну загадок та головоломок. Вони зазвичай включають складні логічні завдання, які гравцям необхідно вирішувати, щоб просуватися за сюжетом гри. Ці ігри розвивають інтелектуальні навички та сприяють розвитку креативності та аналітичного мислення у гравців.

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1. Загальні відомості з предметної галузі

Індустрія комп'ютерних ігор є однією з найбільш швидкозростаючих і найбільш адаптованих галузей нашого століття.

Головною його характерною рисою є постійне відкриття нових жанрів і створення нових ігрових механік, які привертають увагу навіть тих, хто намагається триматися подалі від ігор. Для кожної людини знайдеться щось, що може його зачепити, будь то карткова стратегія, симулятор знайомств або захоплюючі битви посеред космосу. Універсальність такої шкали пояснюється гнучкістю даної галузі до будь-яких змін споживчих запитів і поглядів. Якщо один жанр ігор починає застарівати, в його концепції починають вписуватися нові механіки, які продовжують тримати інтерес публіки [3].

Історія цієї індустрії починається ще з середини дев'яностих, коли проводилися перші експерименти з комп'ютерними іграми на великих і дорогих мейнфреймах. Один із найвідоміших прикладів – "Spacewar!". Розроблена у 1962 році, "Spacewar!" є однією з найраніших комп'ютерних ігор у світі. Створили гру Стів Рассел та Мартін Ґраф, два студенти з Массачусетського технологічного інституту (MIT). Гра була створена для комп'ютера PDP-1, який був одним з перших масово доступних комп'ютерів. "Spacewar!" була космічною аркадною грою, в якій два гравці керували космічними кораблями, які змагалися один з одним у космічному просторі. Гравці могли використовувати реактивні двигуни та фізику, щоб переміщати свої кораблі, стріляти і намагатися знищити один одного. Гра використовувала векторну графіку та мала елементи стратегії, що робило її досить складною та захоплюючою для гравців [2].

"Spacewar!" вплинула на подальший розвиток ігрової індустрії, створюючи основу для багатьох ідей та концепцій, які зараз широко використовуються у відеоіграх. Вона покладала початок для подальшого зростання та інновацій у

галузі комп'ютерних ігор.

Потім, у 1970-ті роки з'явилися перші аркадні ігрові автомати, такі як "Pong" від Atari. Консолі домашнього використання, такі як Atari 2600, стали популярними.

У 1980-ті роки почалася золота ера аркадних ігор. "Pac-Man", "Donkey Kong" та "Super Mario Bros." стали культовими іграми цього часу. Відеоігрові консолі стали доступнішими, а графіка й геймплей продовжили свій розвиток.

Вже у 1990-ті роки отримали своє розширення комп'ютерні ігри та домашні консолі. З'явилися нові платформи, такі як Sega Genesis, Super Nintendo Entertainment System (SNES) і PlayStation. Виникнення 3D-графіки та ігор у жанрі "шутерів".

У 2000-ті почався розвиток мобільних ігор та ігор для персональних комп'ютерів. Поява PlayStation 2, Xbox і Nintendo GameCube. Онлайн-ігри та мультиплеєр стали популярними.

Зараз, у 2023-му році, ігрова індустрія продовжує розвиватися з використанням новітніх технологій, таких як штучний інтелект (AI), хмарні геймінгові платформи та розширена реальність. Завдяки сучасним технологіям, зараз комп'ютерні ігри здатні надати максимальне занурення в придумані світи. Саме тому зростання популярності ігор серед населення різних країн не спадає а не на мить [1].

Інтерес до комп'ютерних ігор пішов від тяги людей до звичайних ігор. Раніше ігри використовувалися в основному для розваги. У сучасних реаліях комп'ютерні ігри (і не тільки комп'ютерні, мобільні теж) являють собою досить цікаве поєднання декількох факторів: вони можуть бути як і розвагою, щоб «вбити час» або просто відпочити, так і розвиваючим або тренувальним механізмом, здатним забезпечити досить якісний рівень знань або навичок. Так, наприклад, за допомогою ігор у віртуальній реальності, допомагають боротися зі страхами людей, або отримати якісь певні навички. У хірургії існує практика, коли перед складною операцією можна провести попередню операцію у віртуальній реальності. Для учнів це може стати можливістю перевірити свої

навички, нікого не покалічивши. Також комп'ютерні ігри сприяють розвитку логічного мислення і профілактики передчасної деменції [8] завдяки тому, що ігри змушують працювати відразу кілька ділянок мозку.

Основними завданнями комп'ютерних ігор можна назвати [5,7]:

- Надати унікальний досвід і максимальне занурення в гру, в основному за рахунок геймплея, історії, ігрового світу, музики і спецефектів;
- Надати гравцеві проблему і надати різні способи її виправлення або вирішення;
- Надати розвиток когнітивних навичок, таких як мислення, увага, спостережливість, логічне мислення та прийняття рішень. Ігри можуть пропонувати гравцеві головоломки, логічні завдання або стратегічні ситуації, які вимагають активного мислення та вирішення проблем;
- Надати соціальну взаємодію. Деякі комп'ютерні ігри мають мультиплеєрний режим, де гравці можуть взаємодіяти один з одним. Це може сприяти розвитку соціальних навичок, співпраці, комунікації та взаєморозуміння. Взаємодія з іншими гравцями може покращити соціальні навички, спілкування та розвиток командної роботи;
- Надати гравцеві нові емоції, щоб поглибити ігровий досвід. Емоції, при цьому, можуть бути різними, як радість і веселощі, так і страх, гнів, бажання відтворити справедливість та ін.;
- Надати гравцеві можливість вчитися і навіть навчати інших. Комп'ютерні ігри можуть мати навчальну складову, спрямовану на викладання певних знань, навичок або конкретних предметів. Ці ігри можуть використовуватися у навчанні для залучення та зацікавлення студентів або дітей;
- Надати гравцеві розвагу, відпочинок і задоволення;
- Створити для гравця виклик та досягнення, за боротьбу з цим викликом;
- Навчити емпатичному мисленню.

1.2. Призначення розробки та галузь застосування

Ігри жанру «квест-головоломка» сприяють розвитку логічного мислення, а також дають можливість перейнятися цікавою історією. Вони дозволяють гравцям відволіктися від повсякденних справ та відчути задоволення від розв'язування складних загадок та завдань. Це також стимулює внутрішній пошук та задоволення від досягнення успіху [8,9].

Основне призначення розробки ігрового застосунка є:

- Надати користувачеві гарний ігровий досвід;
- Розвиток когнітивних навичок у користувача;
- Забезпечення розваги та відпочинку;
- Підвищення творчості та проблемного мислення.

Область застосування різноманітна, так як кожен получає з цього власний досвід, власні враження та власні думки. Але, Звісно ж, основна галузь застосування – це розвага.

1.3. Підстава для розробки

Підставами для розробки «виконання кваліфікаційної роботи» є:

- Освітня програма 122 «Комп’ютерні науки»;
- Навчальний план та графік навчального процесу;
- Наказ ректора Національного технічного університету «Дніпровська політехніка» №350-с від 16.05.2023р;
- Завдання на кваліфікаційну роботу на тему “Розробка багаторівневої відеогри на основі кросплатформного рушія Unity ”.

1.4. Постановка завдання

Завданням даної кваліфікаційної роботи є розробка комп’ютерної гри в жанрі “квест-головоломка”.

Результатом роботи має бути розроблений ігровий застосунок, який має певний перелік механік та функцій, котрі наведені нижче:

- Лінійне оповідання історії, до якого підв'язані ряд подій та тригерів;
- Можливість головного героя рухатись по мапі, взаємодіяти з об’єктами та іншими персонажами;
- Наявність різних міні-ігор та головоломок;
- Наявність декількох рівнів, які обмежені певними локаціями;
- Наявність переходу між локаціями без взаємодії з інтерфейсом;
- Наявність системи збереження та загрузки;
- Наявність системи діалогів, яка має охоплення як головного героя та інших персонажів, так і думок головного героя, котрі появляються у окремій зоні.

1.5. Вимоги до програми або програмного виробу

1.5.1. Вимоги до функціональних характеристик

Вимоги, яких повинен дотримуватися кінцевий продукт:

- Зручний та адаптивний UI до різних розмірів екрану;
- Система збереження та загрузки даних гравця;
- Зручний та простий геймплей;
- Можливість вибору між декількома ігровими аккаунтами (між декількома «Нових ігор»).

1.5.2. Вимоги до інформаційної безпеки

Для забезпечення інформаційної безпеки користувача має бути наявним такий перелік умов:

- Забезпечення захисту конфіденційності даних користувача;
- Забезпечення неможливості передачі даних користувача третім сторонам;

- Повідомлення про помилки для розробників;
- Стабільність роботи системи;
- Наявність обробки виняткових ситуацій.

1.5.3. Вимоги до складу та параметрів технічних засобів

Щоб програмний додаток працював без перебоїв, мінімальні технічні параметри комп'ютера, на який буде встановлено цей додаток, повинні відповідати наступним вимогам:

- Операційна система: Windows 7 (SP1+) або вище/ Linux / MacOs;
- Процесор: Intel Core 2 Duo з частотою не менше 2.4 ГГц та двома ядрами;
- Відеокарта: NVIDIA GeForce GTX 1060 3Гб;
- Оперативна пам'ять: 2 Гб або вище;
- Жорсткий диск: 128 Гб;
- Тип жорсткого диску: HDD.

1.5.4. Вимоги до інформаційної та програмної сумісності

Застосунок буде реалізовано на мові програмування C# з використанням платформи Unity. Unity являється дуже простим, але не менш корисним фреймворком для розробки програмного забезпечення (у більшості випадків - ігор) для майже усіх сучасних операційних систем.

Середовище розробки обрано Visual Studio, яка надає багатий функціонал при розробці ігор та у програмуванні і цілому.

В якості графічного обробника було обрано 2 програми: SAI та Photoshop. Кожна з них надає ряд можливостей з взаємодією та редагуванням графічних зображень. За допомогою SAI буде створено текстури, а за допомогою Photoshop буде їх редагування та створення деяких анімацій.

РОЗДІЛ 2
АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ ТА ПОСТАНОВКА ЗАДАЧІ

2.1 Функціональне призначення програми

Результатом кваліфікованої роботи є комп'ютерна гра жанру квест-головоломка. Гра розрахована на одного гравця. Даний додаток має широкий спектр функціонального призначення, до переліку якого входить [6,8,9]:

1. Розвага: Основна мета гри – надати гравцю розвагу та позитивні емоції. Ця гра пропонує унікальні та цікаві завдання, які викликають інтригу та стимулюють гравця до вирішення складнощів. Гра надає різноманітність головоломок, сюжетних поворотів та можливостей для дослідження, що у результаті створює захоплюючу ігрову атмосферу.

2. Стимуляція розумової діяльності: Гра вимагає від гравця використовувати свої розумові здібності, такі як логічне мислення, запам'ятовування та інше. Додаток сприяє розвитку когнітивних навичок та здібностей, а також допомагає тренувати мозок та покращувати розумові процеси.

3. Поліпшення навичок вирішення проблем: Тут є складні завдання, які потребують пошуку рішень та подолання перешкод. Гра розвиває навички пошуку інформації, аналізу та оцінки варіантів, прийняття рішень та виконання завдань в обмеженому часі чи ресурсах. Ці навички можуть бути корисними не лише в ігровому контексті, а й у реальному житті.

4. Різноманітність та вибір: Ця гра пропонує різні рівні складності для головоломок при переході на нові локації, різноманітні загадки та варіативність сценаріїв. Це дозволяє гравцеві випробувати різні рівні складності та насолоджуватися різноманітністю ігрового процесу, забезпечуючи гнучкість та персоналізацію ігрового досвіду.

2.2. Опис застосованих математичних методів

У цьому додатку, здебільшого, використовувалися лише елементарні арифметичні дії. З формул застосовувалася лише формула знаходження відстані. Складні математичні методи не використовувалися.

2.3 Опис використаних технологій та мов програмування

При розробці цієї програми основними інструментами для роботи було обрано Unity та C#. А також, допоміжні технології та інструменти, такі як:

- TextMeshPro;
- Unity UI;
- DoTween;
- Timeline.

Unity – один з найбільш популярних і широко використовуваних ігрових двигунів в промисловості. Завдяки своїм потужним можливостям, гнучкості та підтримці різних платформ, Unity дозволяє розробникам втілювати свої творчі ідеї у захоплюючі та високоякісні ігрові проекти.

У ключові особливості та можливості Unity можна додати кросплатформність. Unity підтримує розробку ігор для різних платформ, включаючи ПК, мобільні пристрої, консолі, Інтернет та інші. Це дозволяє розробникам створювати ігри, які можуть бути запущені на різних пристроях без необхідності переписувати код із нуля [10].

Також основним моментом є те, що Unity пропонує інтуїтивний візуальний редактор, який дозволяє створювати та налаштовувати ігрові об'єкти, сцени, анімації та інтерфейси користувача без необхідності писати код. Це робить розробку доступнішою для новачків і дозволяє швидко прототипувати ітерації ігрових проектів. Unity має компонентну архітектуру. Він заснований на компонентній моделі розробки, де функціональність ігрових об'єктів досягається шляхом комбінування та налаштування компонентів. Це дозволяє легко

створювати поведінку та функціональність об'єктів шляхом додавання, видалення та налаштування компонентів [11].

Звичайно, однією з найважливіших особливостей є скриптинг та програмування. Unity підтримує кілька мов програмування, включаючи C# та JavaScript. Це впливає на різноманітність ресурсів та інструментів, які пропонує Unity, включаючи бібліотеки ассетів, моделей, текстур, звуків та ефектів, які можуть бути використані для створення візуальних та звукових ефектів у грі.

C# – об'єктно-орієнтована мова програмування, яка розроблена Microsoft. Він був представлений у 2000 році і став однією з основних мов програмування у платформі розробки .NET. Одними з основних особливостей є те, що C# повністю підтримує принципи об'єктно-орієнтованого програмування (ООП), такі як інкапсуляція, спадкування та поліморфізм. Він дозволяє створювати класи, об'єкти та використовувати механізм успадкування для організації та структурування коду [12].

Головне, через що C# був обраний для роботи в даному проекті – сумісність із Unity та його компонентами. Основними моментами, в яких проявляється ця сумісність [20, 21], є:

- C# є основною мовою програмування Unity. Він надає розробникам повний доступ до функціональності Unity API та дозволяє створювати ігровий контент, включаючи сцени, об'єкти, анімації, логіку гри та багато іншого.
- Велика документація та ресурси: Unity пропонує велику документацію, навчальні матеріали, приклади коду та спільноту розробників, які допомагають новачкам та досвідченим розробникам вивчати та використовувати C# у контексті Unity. Це робить процес розробки більш доступним та забезпечує підтримку та допомогу при виникненні питань.
- Розширюваність та плагінна підтримка: C# в Unity підтримує можливість створення та використання власних плагінів та розширень. Розробники можуть створювати власні скрипти, компоненти та інструменти, щоб розширити функціональність Unity та адаптувати його під свої потреби.

- Інтегроване середовище розробки: Unity надає інтегроване середовище розробки (IDE) з повною підтримкою C#. Unity IDE має потужні інструменти для розробки, налагодження, автодоповнення та аналізу коду на C#. Це спрощує та прискорює процес створення ігрового контенту.

Отже, сумісність C# з Unity робить їх сильним та ефективним комбінацією для розробки ігрового контенту, включаючи цей проект.

Говорячи про допоміжні інструменти, можна почати з TextMeshPro, який є розширенням для Unity, яке надає більш просунуті можливості форматування та відображення тексту в порівнянні зі стандартним компонентом Text. Він має покращену якість відображення тексту: TextMeshPro використовує власний рендеринг тексту, заснований на механізмах субпіксельного згладжування, що призводить до більш чіткого та читаного відображення тексту навіть на різних дозволах і щільності пікселів [13]. Різницю відображення тексту можна побачити на рис. 2.1.

Рис. 2.1. Різниця якості шрифтів. Зверху – звичайний Text, знизу – TMPro

TextMeshPro має багатий функціонал форматування тексту. Він пропонує широкий набір інструментів для форматування тексту, таких як різні шрифти та стилі, розміри та вирівнювання, налаштування міжрядкового та міжлітерного інтервалів, зовнішні контури та тіні, багатомовна підтримка та багато іншого (рис.2.2). Це дозволяє створювати стильний текстовий контент, що налаштовується [13].

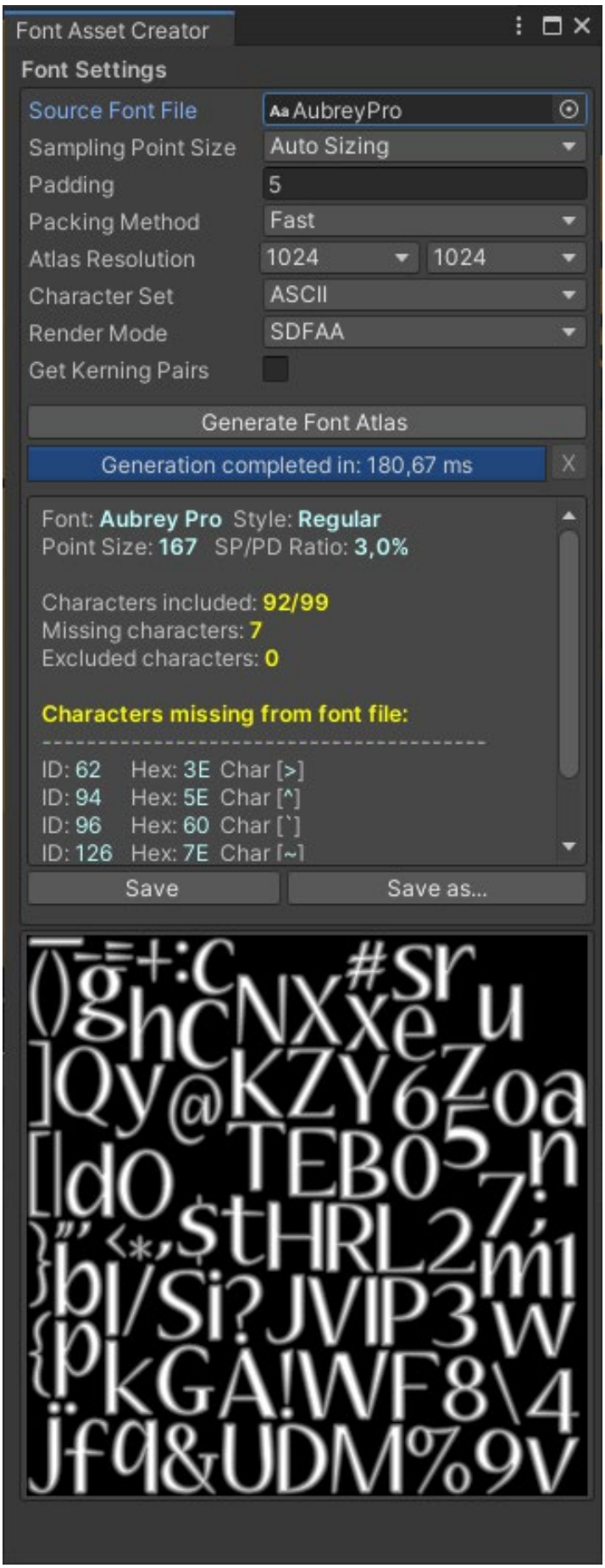


Рис. 2.2. Панель налаштування шрифту

Unity UI (User Interface) - це система для створення інтерфейсу користувача в іграх і додатках, що розробляються в Unity. Unity UI надає різні компоненти та інструменти для створення інтерактивних та адаптивних інтерфейсів користувача. Unity UI пропонує різноманітні компоненти для створення інтерфейсу користувача, такі як кнопки, текстові поля, зображення, панелі, списки, перемикачі та багато іншого (рис. 2.3). Розробники можуть легко створювати та налаштовувати ці компоненти, щоб будувати інтерфейси, що відображають інформацію та забезпечують взаємодію з користувачем [14].

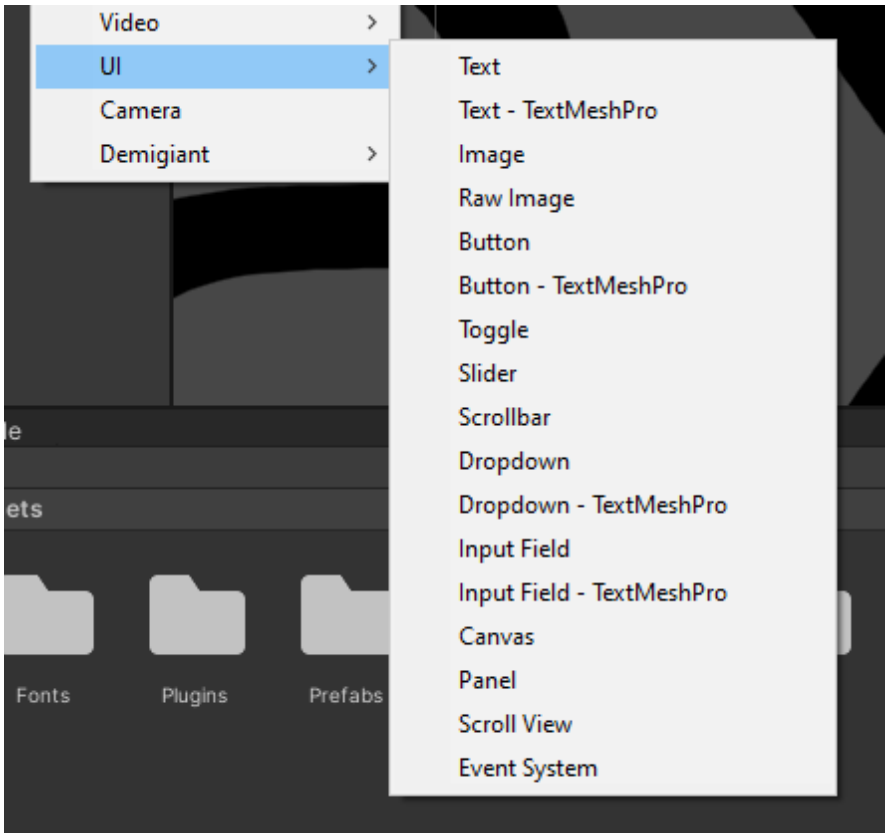


Рис. 2.3. Панель створення UI

DoTween (DOTween) - це бібліотека анімації та твінінгу (tweening) для Unity, яка надає зручні та потужні інструменти для створення плавних та динамічних анімацій в іграх та додатках. DoTween надає простий та інтуїтивно зрозумілий API для створення анімацій. За допомогою кількох рядків коду можна анімувати властивості об'єктів, такі як положення, обертання, масштаб, колір та інші, легко налаштовувати параметри анімації. DoTween надає різні

методи керування анімацією, такі як пауза, відтворення, переривання та зворотне відтворення. Це дозволяє розробникам контролювати час та порядок анімацій, створювати петлі та налаштовувати поведінку анімацій у реальному часі [15].

Timeline (Монтажна стрічка) - це інструмент у Unity, який дозволяє розробникам створювати та керувати складними часовими лініями та сценаріями анімації. Він надає гнучкий спосіб керування анімацією об'єктів, подіями, звуками та іншими елементами гри у часі. Timeline надає інтуїтивно зрозумілий візуальний редактор, у якому розробники можуть створювати часові лінії та компонувати елементи анімації та подій. За допомогою інтерфейсу можна легко додавати та налаштовувати ключові кадри, таймлайни анімації та звуку, події та інші елементи. У Timeline є потужні інструменти для керування тимчасовою шкалою (рис. 2.4). Тут можна змінювати тривалість тимчасової лінії, переміщати та змінювати тривалість елементів анімації, налаштовувати затримки та часові інтервали, а також використовувати криві анімації для контролю плавності та прогресу анімацій.

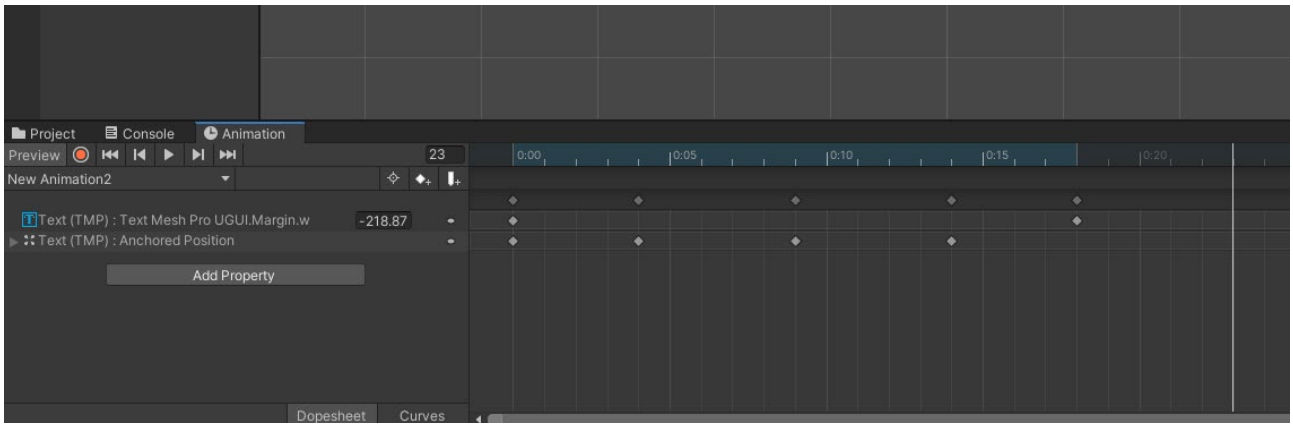


Рис. 2.4. Панель керування анімацією

2.4 Опис структури програми та алгоритмів її функціонування.

Так як гра складається з безлічі ігрових екранів, для детального опису послідовностей та структури проекту були використані UML діаграми Use Case, а також блок-схеми.

На діаграмі нижче представлено функціонал головного меню гри. Головне меню містить стандартний набір можливостей для ігор жанру квест-головоломка. Тут користувач має можливість створити нову гру, вибрати вже збережену, змінити налаштування гри, подивитися титри або залишити гру (рис.2.5).

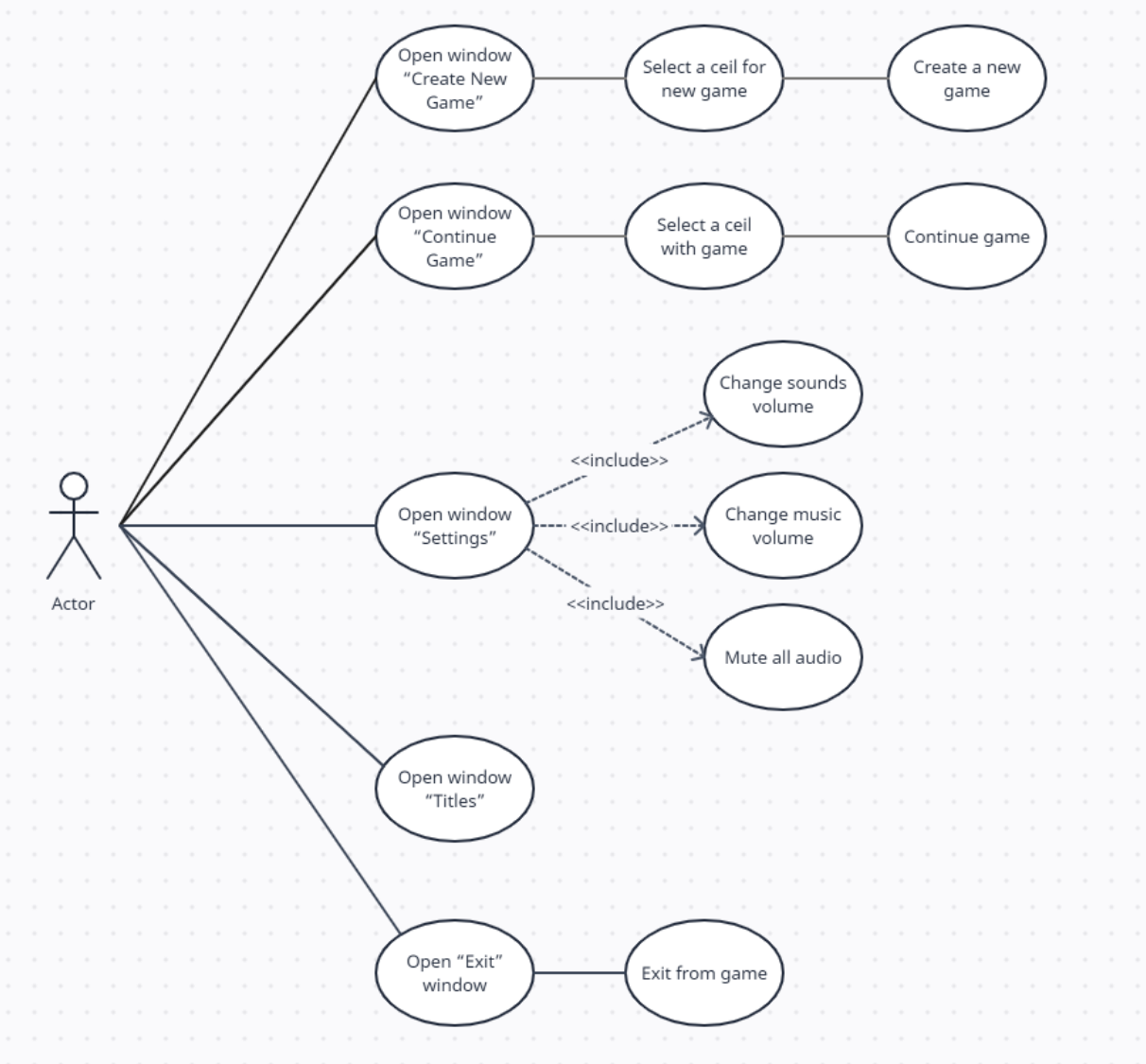


Рис. 2.5. Діаграма Use Case головного екрана гри

Наступна діаграма описує можливі дії гравця безпосередньо у процесі гри. Дані дії доступні гравцеві в момент відсутності сценарної виконуючої, або ж по-іншому, в період, коли поточний момент гри не містить діалогу, події або вже використаного однієї з можливих дій. Також обов'язково наявність елемента, з

яким цю взаємодію можна виконати (якщо говорити про інший персонаж або об'єкт) (рис. 2.6.). До переліку можливих дій входить:

- Пересування персонажа з локації;
- Взаємодія/можливість розпочати діалог з іншими персонажами;
- Взаємодія з предметами/можливість їх підібрати;
- Відкрити вікно з міні-грою;
- Відкрити меню паузи;
- Відкрити інвентар.

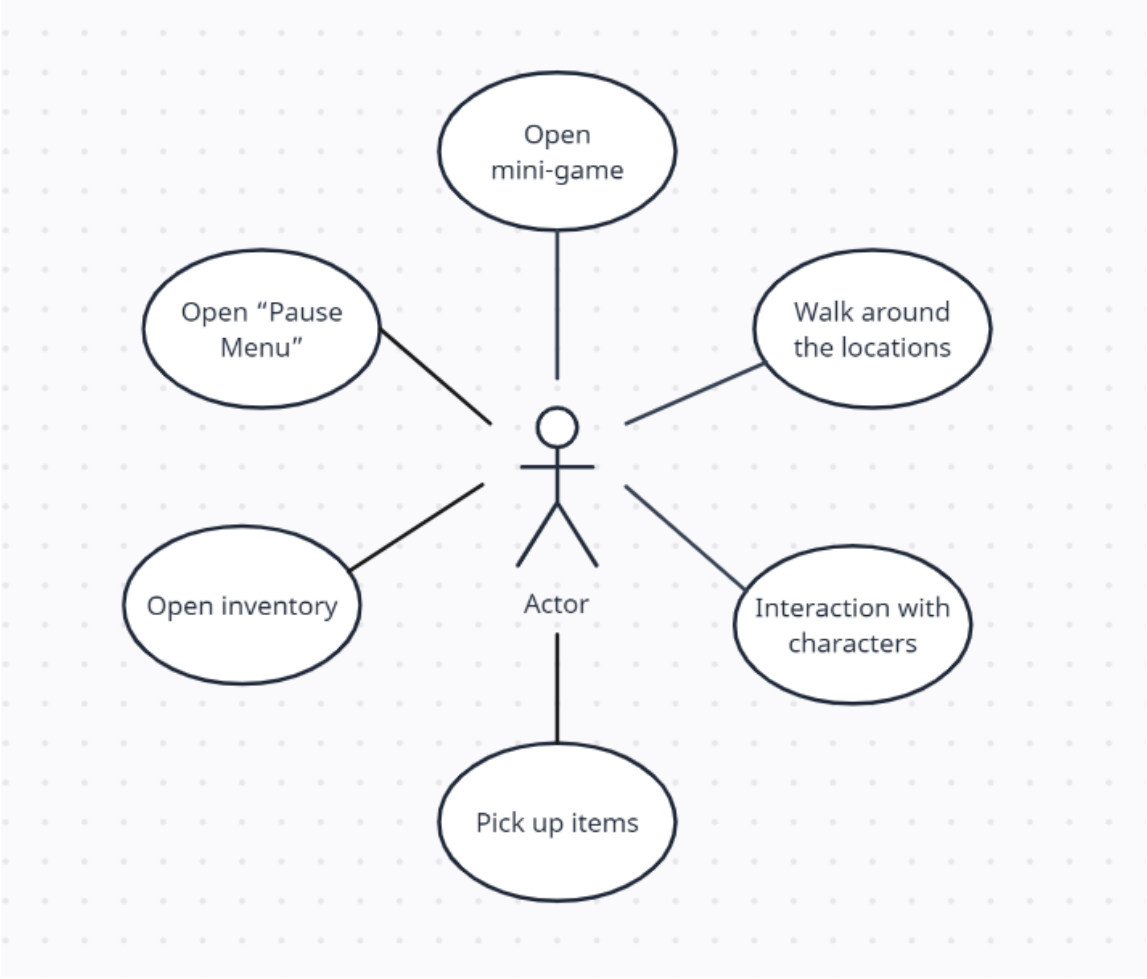


Рис. 2.6. Діаграма Use Case основногоекрана гри

Діаграма нижче показує опції меню паузи гри. Тут є можливість продовжити гру, змінити налаштування або повернутися до головного меню (рис. 2.7.).

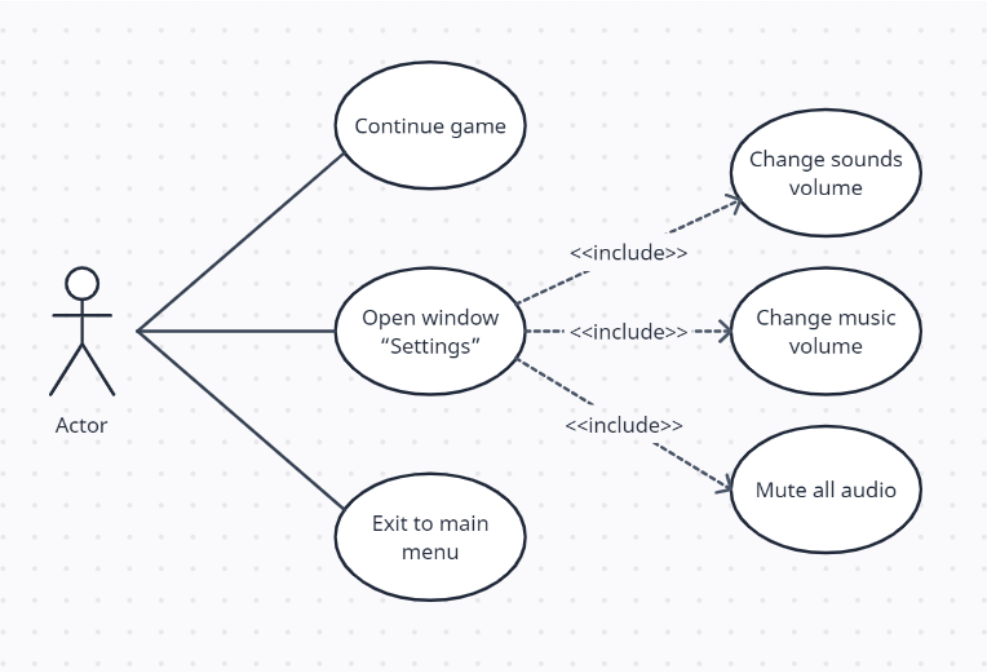


Рис. 2.7. Діаграма Use Case меню паузи гри

Діаграма інвентарю показує функції, доступні під час роботи з підібраними об'єктами. Серед них: використати предмет, показати опис предмета, а також об'єднати предмети (рис. 2.8.).

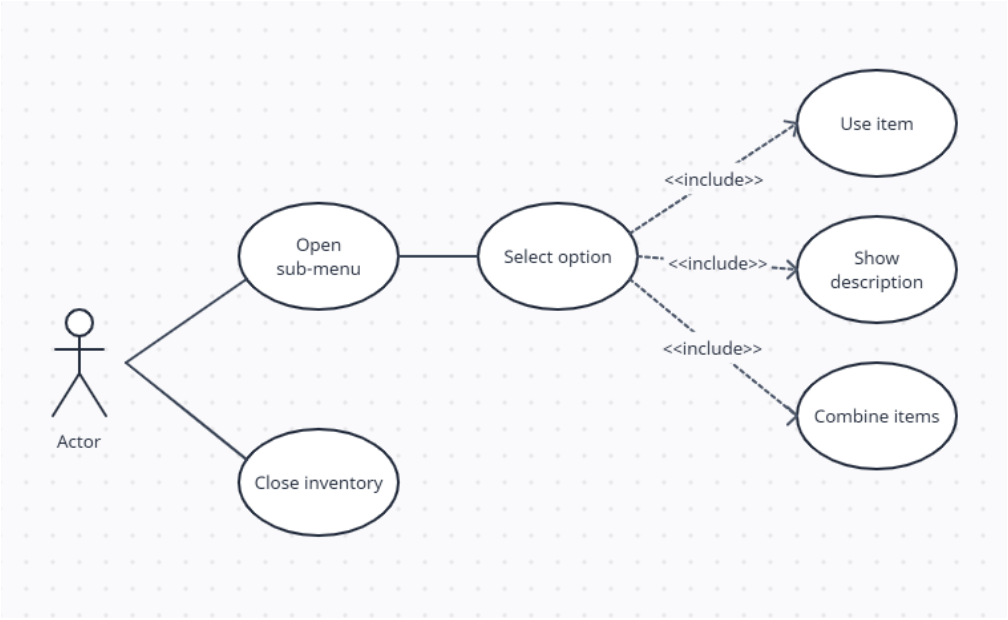


Рис. 2.8. Діаграма Use Case інвентарю

Основна логіка гри укладена в блок-схемі нижче. Вся базова структура програми базується на обробці даних із зовнішніх файлів сценаріїв формату

JSON. Ключовий елемент тут – це рядок сценарію, доступ до елементів якого ми отримуємо через його ID. Усі події в грі, кожен діалог чи доступність об'єктів залежать від того, в якому рядку сценарію вони знаходяться. Тому вся основна логіка підв'язана до різних варіацій роботи з ID сюжетного рядка (рис. 2.9).

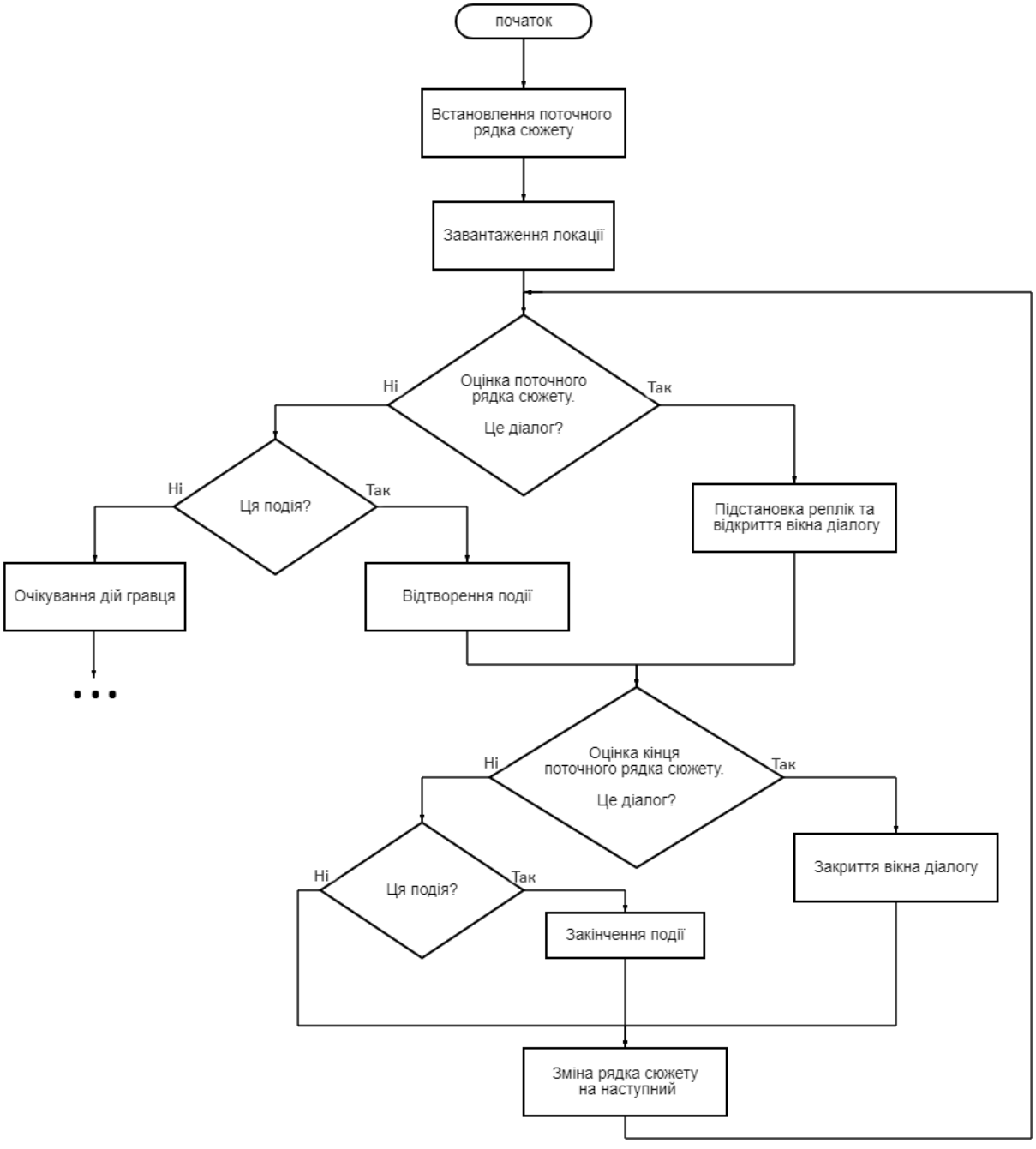


Рис. 2.9. Блок-схема основної логіки гри, частина 1

Продовження попередньої схеми. Більш детальний опис алгоритму дій

гравця та їх наслідків. У цю схему не додані моменти вибору меню паузи та відкриття інвентарю, оскільки це робота з інтерфейсом, а ця блок-схема описує алгоритм виключно ігрового процесу, без урахування взаємодії з інтерфейсом (рис. 2.10.).

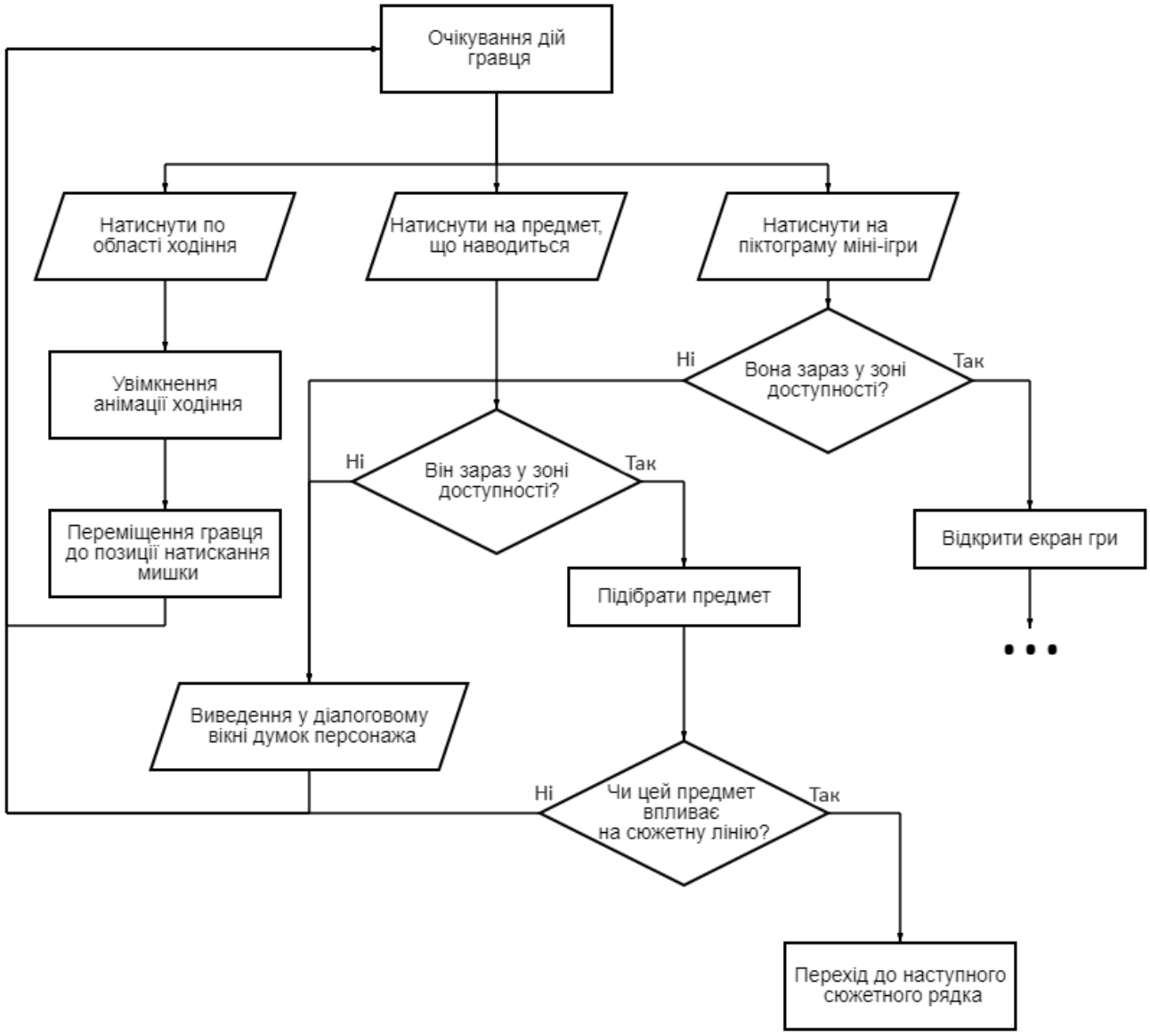


Рис. 2.10. Блок-схема основної логіки гри, частина 2

Наступна схема – схема ігрової логіки міні-ігри. Тут детально описано логіку гри цятки (точніше, як вона реалізована в даному проекті) (рис. 2.11.).

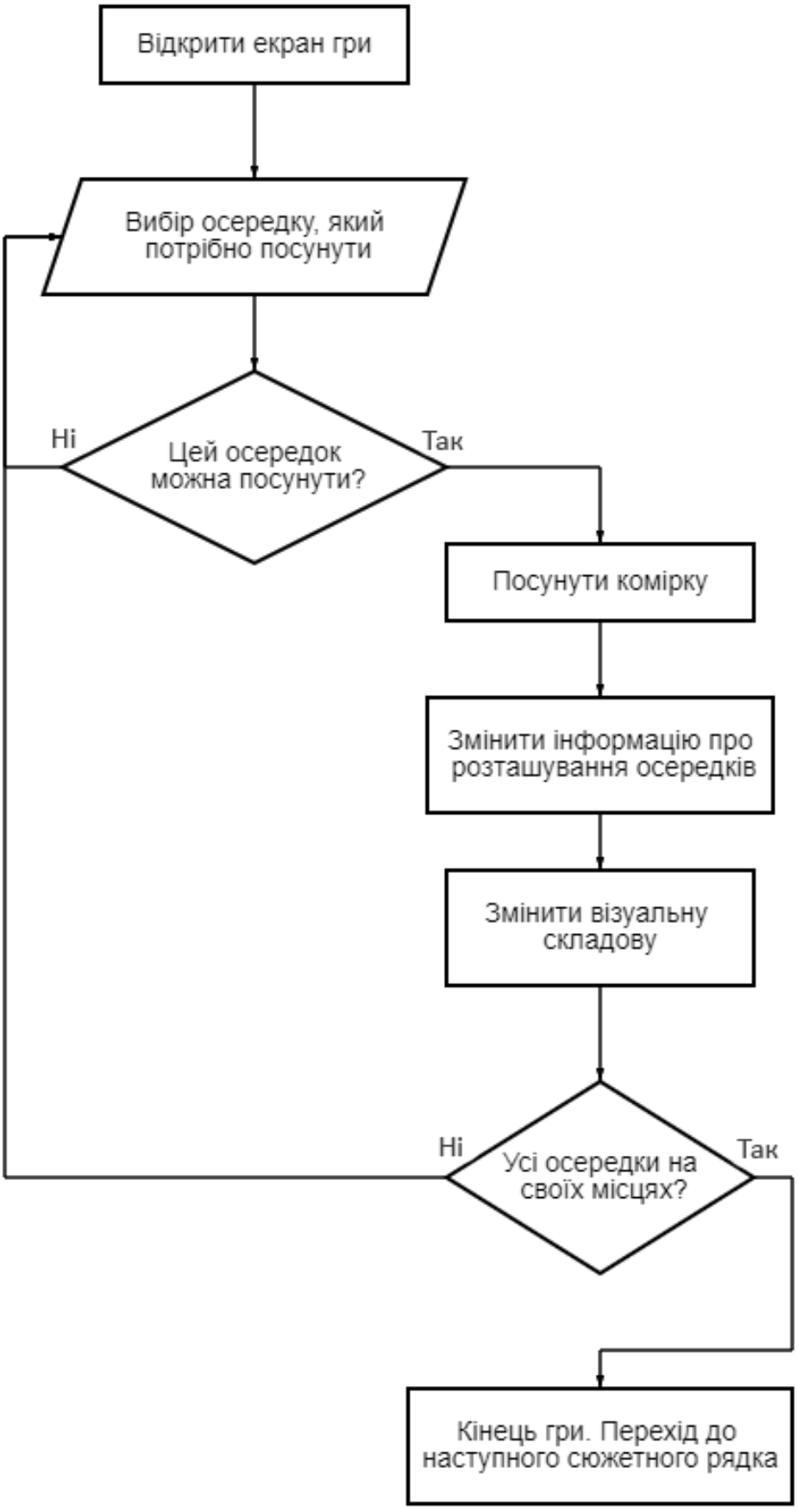


Рис. 2.11. Логіка міні-гри

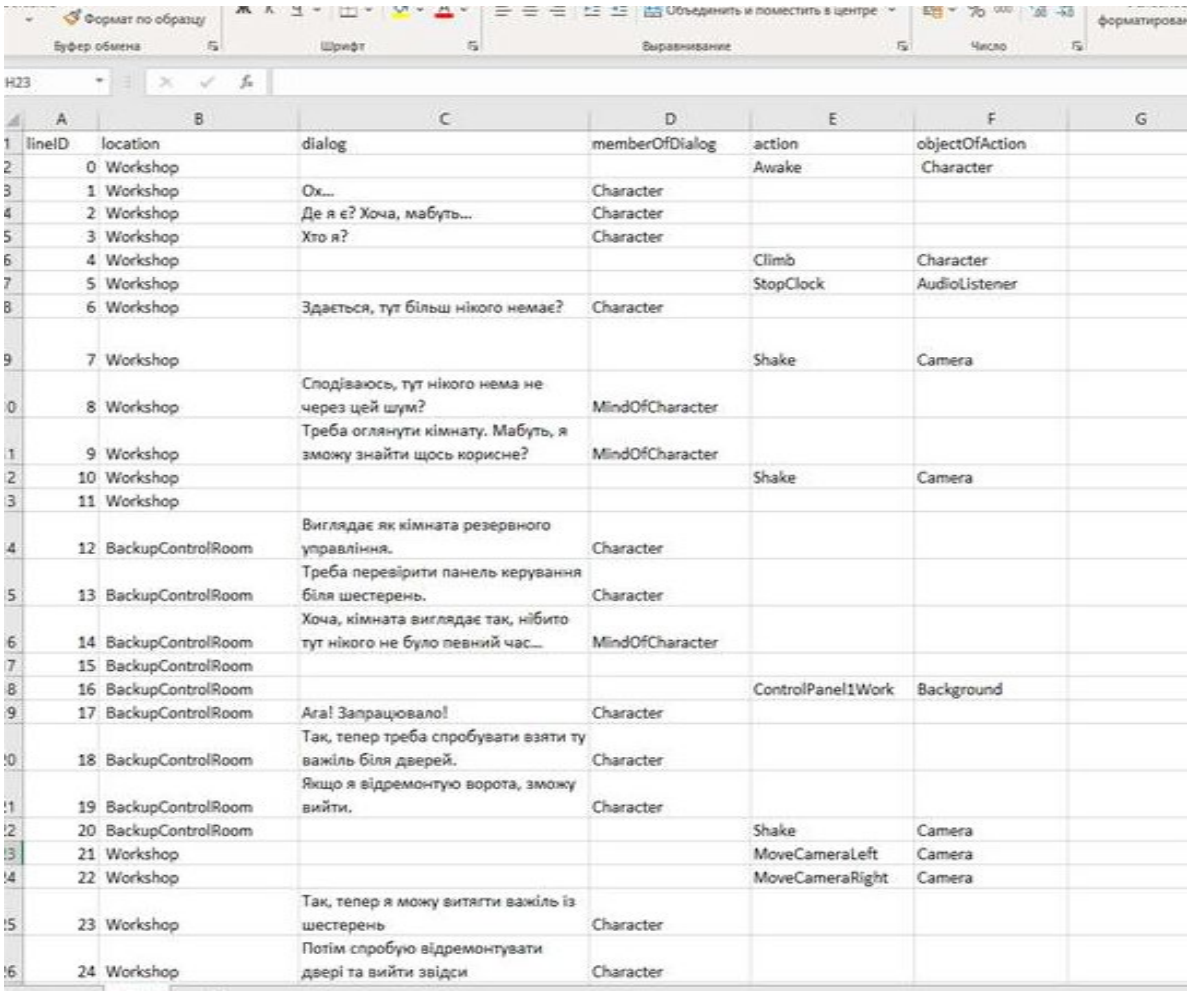
2.5 Обґрунтування та організація вхідних та вихідних даних програми

Вхідними даними для цієї програми є дані із зовнішніх файлів сценаріїв формату JSON. Перед тим, як конвертувати сценарій у файл JSON, він записується у таблицях EXCEL. Це робиться для зручнішого перегляду та редагування вхідних даних. Ключовим елементом даних таблицях є ID поточного рядка сценарію. Кожен рядок сценарію може містити 1 з 3 типів можливого дій:

- Діалог;
- Подія;
- Вільний час (коли рядок порожній, персонаж може переміщатися по карті та взаємодіяти з іншими персонажами/об'єктами).

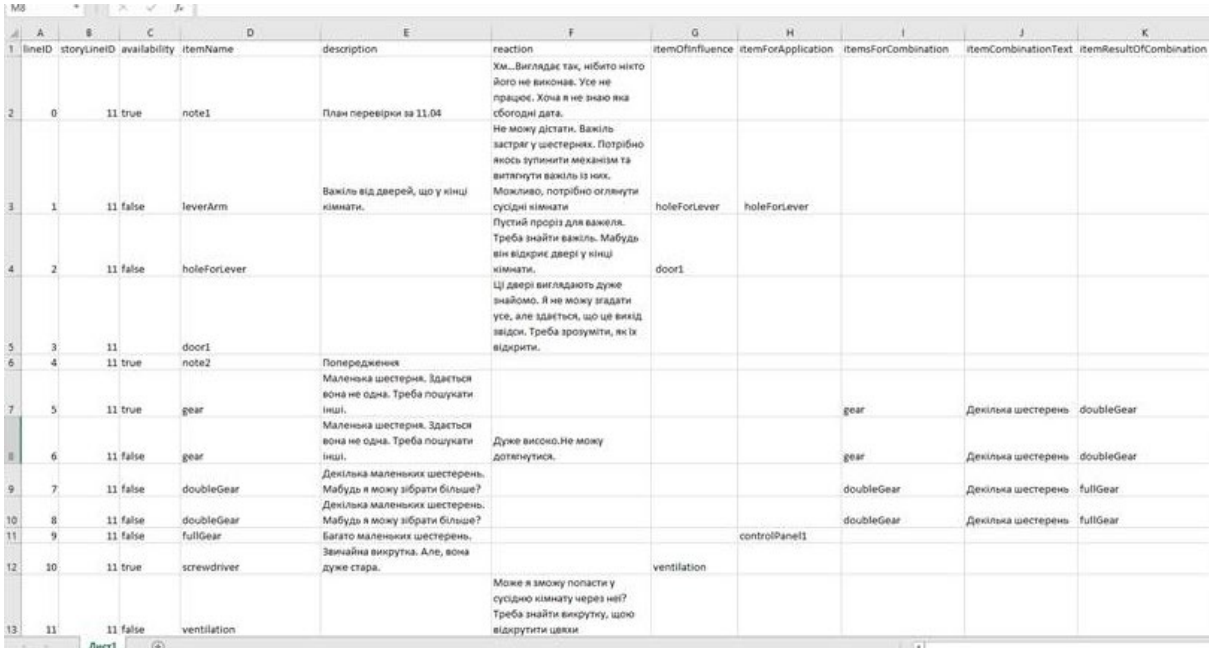
Одночасно (в одному рядку) кілька подій не може бути. Якщо потрібний послідовний виклик певних подій, краще вказувати їх у кількох рядках поспіль. Прикладом цього можуть бути рядки 4 і 5 або рядки 20,21,22 на рис. 2.12.

Усього вихідних таблиць 2: таблиця з основною сюжетною лінією (рис. 2.12) та таблиця предметів (рис. 2.13.). У таблиці з історією знаходяться всі події та діалоги, які доступні в грі, а також персонажі, яким вони належать або об'єкти, до яких застосовна подія. Тут також вказується локація кожного рядка сюжету. Локація в таблиці вказується більшою мірою для завантаження гри при натисканні кнопки в меню «Продовжити гру», ніж для відтворення самої історії. У таблиці з предметами міститься інформація про всі предмети у грі, з якими можна взаємодіяти. Таким предметом може бути як частина заднього фону, так і предмет, який можна підібрати в інвентар. У таблиці вказується доступність предмета в певний момент часу, на що впливає його використання, з чим можна використовувати, з чим його можна комбінувати і те, що виходить в результаті комбінування. Також певні події з предметом можуть проводити доступність іншого предмета.



	A	B	C	D	E	F	G
1	lineID	location	dialog	memberOfDialog	action	objectOfAction	
2	0	Workshop			Awake	Character	
3	1	Workshop	Ох...	Character			
4	2	Workshop	Де я є? Хоча, мабуть...	Character			
5	3	Workshop	Хто я?	Character			
6	4	Workshop			Climb	Character	
7	5	Workshop			StopClock	AudioListener	
8	6	Workshop	Здається, тут більш нікого немає?	Character			
9	7	Workshop			Shake	Camera	
10	8	Workshop	Сподіваюсь, тут нікого нема не через цей шум?	MindOfCharacter			
11	9	Workshop	Треба оглянути кімнату. Мабуть, я зможу знайти щось корисне?	MindOfCharacter			
12	10	Workshop			Shake	Camera	
13	11	Workshop					
14	12	BackupControlRoom	Виглядає як кімната резервного управління.	Character			
15	13	BackupControlRoom	Треба перевірити панель керування біля шестерень.	Character			
16	14	BackupControlRoom	Хоча, кімната виглядає так, нібито тут нікого не було певний час...	MindOfCharacter			
17	15	BackupControlRoom					
18	16	BackupControlRoom			ControlPanel1Work	Background	
19	17	BackupControlRoom	Ага! Запрацювало!	Character			
20	18	BackupControlRoom	Так, тепер треба спробувати взяти ту важіль біля дверей.	Character			
21	19	BackupControlRoom	Якщо я відремонтую ворота, зможу вийти.	Character			
22	20	BackupControlRoom			Shake	Camera	
23	21	Workshop			MoveCameraLeft	Camera	
24	22	Workshop			MoveCameraRight	Camera	
25	23	Workshop	Так, тепер я можу витягти важіль із шестерень	Character			
26	24	Workshop	Потім спробую відремонтувати двері та вийти звідси	Character			

Рис. 2.12. Таблиця з історією



	A	B	C	D	E	F	G	H	I	J	K
1	lineID	storyLineID	availability	itemName	description	reaction	itemOfInfluence	itemForApplication	itemsForCombination	itemCombinationText	itemResultOfCombination
2	0	11	true	note1	План перевірки за 11.04	Хм...Виглядає так, нібито ніхто його не виконав. Усе не працює. Хоча я не знаю яка сьогодні дата.					
3	1	11	false	leverArm	Важіль від дверей, що у кінці кімнати.	Не можу дістати. Важіль застряг у шестернях. Потрібно якось зупинити механізм та витягнути важіль із них. Можливо, потрібно оглянути сусідні кімнати	holeForLever	holeForLever			
4	2	11	false	holeForLever		Пустий проріз для важеля. Треба знайти важіль. Мабудь він відкриє двері у кінці кімнати.	door1				
5	3	11		door1		Ці двері виглядають дуже знайомо. Я не можу згадати усе, але здається, що це вихід звідси. Треба зрозуміти, як їх відкрити.					
6	4	11	true	note2	Попередження						
7	5	11	true	gear	Маленька шестерня. Здається вона не одна. Треба пошукати інші.				gear	Декілька шестерень	doubleGear
8	6	11	false	gear	Маленька шестерня. Здається вона не одна. Треба пошукати інші.	Дуже високо.Не можу дотягнутися.			gear	Декілька шестерень	doubleGear
9	7	11	false	doubleGear	Декілька маленьких шестерень. Мабудь я можу зібрати більше?				doubleGear	Декілька шестерень	fullGear
10	8	11	false	doubleGear	Декілька маленьких шестерень. Мабудь я можу зібрати більше?				doubleGear	Декілька шестерень	fullGear
11	9	11	false	fullGear	Багато маленьких шестерень.			controlPanel1			
12	10	11	true	screwdriver	Звичайна викрутка. Але, вона дуже стара.		ventilation				
13	11	11	false	ventilation		Може я зможу попасти у сусідню кімнату через неї? Треба знайти викрутку, щоб відкрутити цяхи					

Рис. 2.13. Таблиця з предметами

Далі файли EXCEL конвертуються у файли JSON у будь-якому доступному онлайн конверторі. На виході виходить файл, готовий до використання у програмі. На скріншотах нижче показані файли JSON (рис. 2.14, 2.15.).

Рис. 2.14. Таблиця з історією у форматі JSON

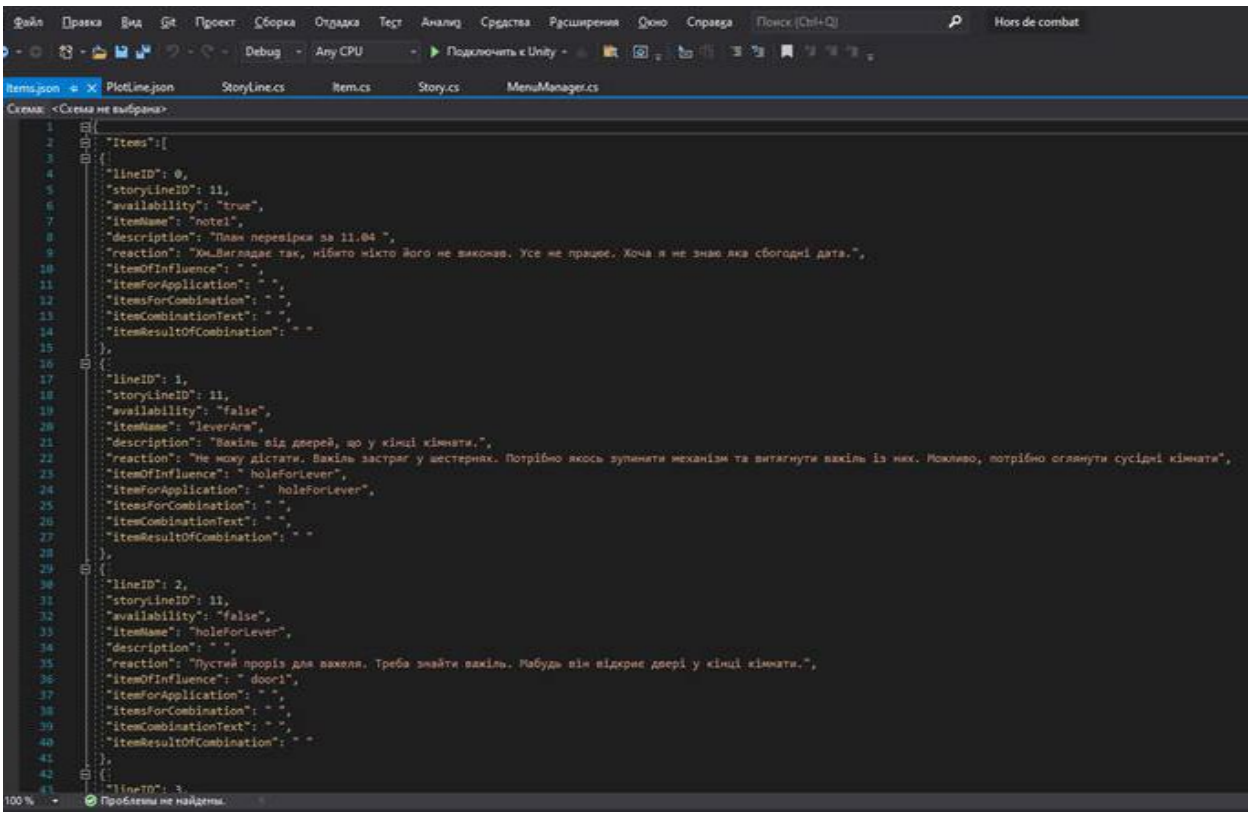


Рис. 2.15. Таблиця предметів у форматі JSON

За допомогою JsonUtility (клас в Unity, який надає зручні методи для серіалізації (перетворення в JSON) та десеріалізації (перетворення з JSON) об'єктів. Він дозволяє працювати з даними у форматі JSON, який є поширеним форматом обміну даними та зберігання структурованої інформації [16]) дані вилучаються із файлів JSON.

Далі дані обробляються вже всередині програми. У програмі передбачені 2 класи заготівлі та 2 класи, які приймають у себе та обробляють вхідні дані, унаслідок чого формуються 2 масиви з ініціалізованими об'єктами класів заготовок. Трохи докладніше про це нижче:

- Перший клас заготівля для файлу з даними з основною сюжетною лінією:

```
[Serializable]
public class StoryLine
{
   public int lineID;
```

```
    public string location;
    public string dialog;
    public string memberOfDialog;
    public string action;
    public string objectOfAction;
}
```

Дані витягуються та записуються в екземпляри класу StoryLine у класі Story (де відбувається базова робота з отриманими даними), далі з цими даними працює StoryManager вже через клас Story. Як можна побачити, поля класу StoryLine відповідають полям у відповідному файлі JSON, а також іменуванні колонок у таблиці EXCEL.

- Другий клас заготівля для файлу з даними про предмети:

```
[Serializable]
public class Item
{
    public int lineID;
    public int storyLineID;
    public bool availability;
    public string itemName;
    public string description;
    public string reaction;
    public string itemOfInfluence;
    public string itemForApplication;
    public string itemsForCombination;
    public string itemCombinationText;
    public string itemResultOfCombination;

}
```

Дані витягуються та записуються в екземпляри класу Item у класі ItemManager, де потім і використовуються.

Також, вхідними даними можна назвати 2 списки (підняті предмети та покладені предмети) і ID поточного рядка, які ми отримуємо при завантаженні гри із зовнішніх, раніше збережених файлів. При першому запуску цих файлів або немає, або вони порожні, у разі відсутності хоча б першого осередку з грою. В інших випадках вони повертають перелік предметів і номер поточного рядка сюжету, на якому зупинився гравець при виході з гри.

Вихідними даними у цьому проекті є файл, створений PlayerPrefs, у якому містяться рядок сюжету, списки підібраних та покладених предметів. Які, згодом, будуть вхідними даними при наступному запуску гри.

PlayerPrefs - це клас у Unity, який надає простий спосіб збереження та завантаження даних гравця. Він дозволяє зберігати та отримувати значення змінних, які зберігаються між сеансами гри та доступні для використання при наступному запуску гри. PlayerPrefs дозволяє зберігати та завантажувати дані різних типів, таких як цілі числа, речові числа, рядки та булеві значення. Дані зберігаються у файлі на диску або реєстрі операційної системи і доступні для використання навіть після перезапуску гри. Дані, збережені за допомогою PlayerPrefs, є глобальними та доступними для всіх сцен та об'єктів у грі. Це означає, що дані можуть бути використані для настроювання ігрових параметрів, збереження стану гри або відстеження прогресу гравця протягом усього ігрового процесу. PlayerPrefs має деякі обмеження, такі як максимальний розмір даних, який можна зберегти, і обмежена підтримка типів даних. Рекомендується використовувати PlayerPrefs для зберігання невеликих обсягів даних, таких як налаштування користувача або результати гри, та уникати збереження великих обсягів даних або частих операцій збереження/завантаження [17].

2.6. Опис розробленої системи

2.6.1 Використані технічні засоби

Для розробки ігрового додатка був використаний ноутбук із такими характеристиками:

- Операційна система: Windows 10;
- Процесор: Intel(R) Core(TM) i7-3840QM CPU 2.80GHz;
- Відеокарта: NVIDIA Quadro K3000M 2Гб;
- Оперативна пам’ять: 16 Гб;
- Жорсткий диск: 500 Гб;
- Тип жорсткого диску: SSD.

2.6.2. Використані програмні засоби

Unity – це популярний кросплатформний редактор для ігор, який використовується для створення різних типів інтерактивного контенту, включаючи відеоігри, віртуальну реальність, доповнену реальність, тривимірні анімації та симуляції. Він надає розробникам потужний інструментарій та середовище розробки, які дозволяють створювати ігрові проекти різних жанрів та масштабів.

Visual Studio – це інтегроване середовище розробки (IDE) від Microsoft, призначене для створення програмного забезпечення. Вона пропонує потужні інструменти та функціональність для розробки додатків різними мовами програмування, включаючи C#. Visual Studio підтримує кілька мов програмування, дозволяючи розробникам вибирати найбільш підходящу мову для своїх проектів. Кожна мова має свої інструменти, редактори та відладчики, специфічні для його синтаксису та особливостей. Visual Studio пропонує потужний редактор коду з автодоповненням, підсвічуванням синтаксису, функцією переходу до визначення та іншими зручностями. Він допомагає розробникам збільшити продуктивність та знизити кількість помилок, надаючи інструменти для більш ефективного написання та редагування коду.

Adobe Photoshop – це програмне забезпечення для редагування та обробки растрових зображень. Він є одним з найпопулярніших і широко використовуваних програм у галузі графічного дизайну, фотографії, веб-дизайну та інших суміжних областях. Photoshop надає широкий спектр інструментів та

функцій для редагування та покращення зображень. У цьому проекті він використовувався для створення текстур та візуалізації ігрового процесу [18].

SAI (PaintTool SAI) – це програмне забезпечення для цифрового живопису та ілюстрації, розроблене японською компанією Systemax. SAI пропонує різноманітні інструменти для малювання та кольору, такі як пензлі з різними формами та текстурами, інструменти для створення градієнтів та ефектів, інструменти для роботи з шарами та багато іншого. При розробці цієї ігрової програми SAI був використаний як додатковий інструмент для обробки або доопрацювання графіки, створеної в Adobe Photoshop [19].

2.6.3 Виклик та завантаження програми

Після завантаження та розпакування архіву з грою, ігрову програму можна запустити з exe файлу (рис. 2.16.).

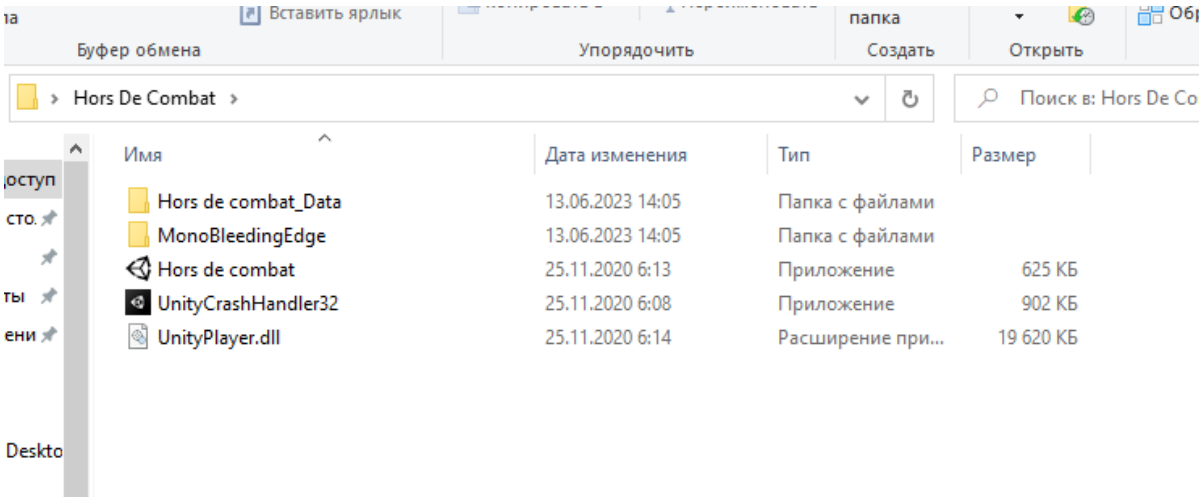


Рис. 2.16. Папка с файлом запуску гри

2.6.4. Опис інтерфейсу користувача

Коли гравець включає гру, перше, що він бачить – екран головного меню. У головному меню є 5 пунктів: "Нова гра", "Продовжити гру", "Налаштування", "Титри", "Вихід" (рис. 2.17.).

Рис. 2.17. Головне меню гри

Натискання кнопки «Нова гра» відкриває вікно, в якому користувач може створити нову гру, перезаписати вже існуючу або видалити гру (2.18.).

Рис. 2.18. Панель створення нової гри

Щоб створити гру, потрібно натиснути кнопку + Нова гра, щоб перезаписати – натиснути на іконку пера, щоб видалити – натиснути на хрестик біля гри, яку хочете видалити (рис. 2.19, 2.20).

Рис. 2.19. Панель перезапису гри

Рис. 2.20. Панель видалення гри

При натисканні кнопки "Продовжити гру" з'являється вікно, в якому гравець може вибрати збереження та продовжити грати. Також тут можна видалити збереження (рис. 2.21, 2.22.).

Рис. 2.21. Панель продовження гри

Рис. 2.22. Панель видалення запису

Кнопка «Параметри» показує доступні опції роботи зі звуком. Тут можна вимкнути звук і музику або настроїти їх гучність (2.23).

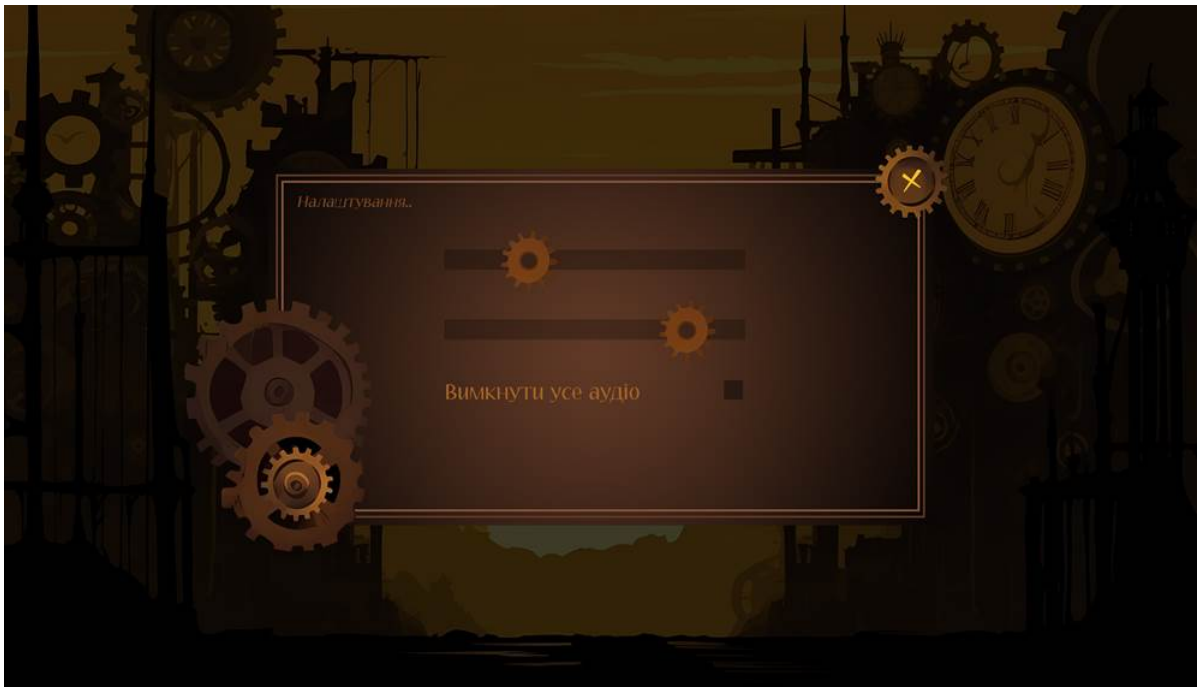


Рис. 2.23. Налаштування гри

Пункт "Титри" показує творців гри (рис. 2.24.).

Рис. 2.24. Панель з титрами

Остання кнопка «Вийти з гри» надає гравцеві можливість залишити гру (рис. 2.25.).

Рис. 2.25. Панель виходу з гри

Після того, як гравець створює нову гру або продовжує стару, у нього з'являється основний екран з грою. Нижче на скріншоті показані вільні дії гравця (рис.2.26.).

Рис. 2.26. Основний екран гри

На цьому скріншоті показано приклад системи діалогу (рис. 2.27.).

Рис. 2.27. Приклад вікна діалогу

Приклад думок персонажа про подію або предмет показано нижче (рис. 2.28.):

Рис. 2.28. Приклад думок персонажа

Вгорі панелі є 2 кнопки: «Показати меню паузи» та «Інвентар». Коли у перше відкрити інвентар, він пустий (рис. 2.29.). Але після того як підібрати предмет, він з’являється в інвентарі (рис. 2.30.).

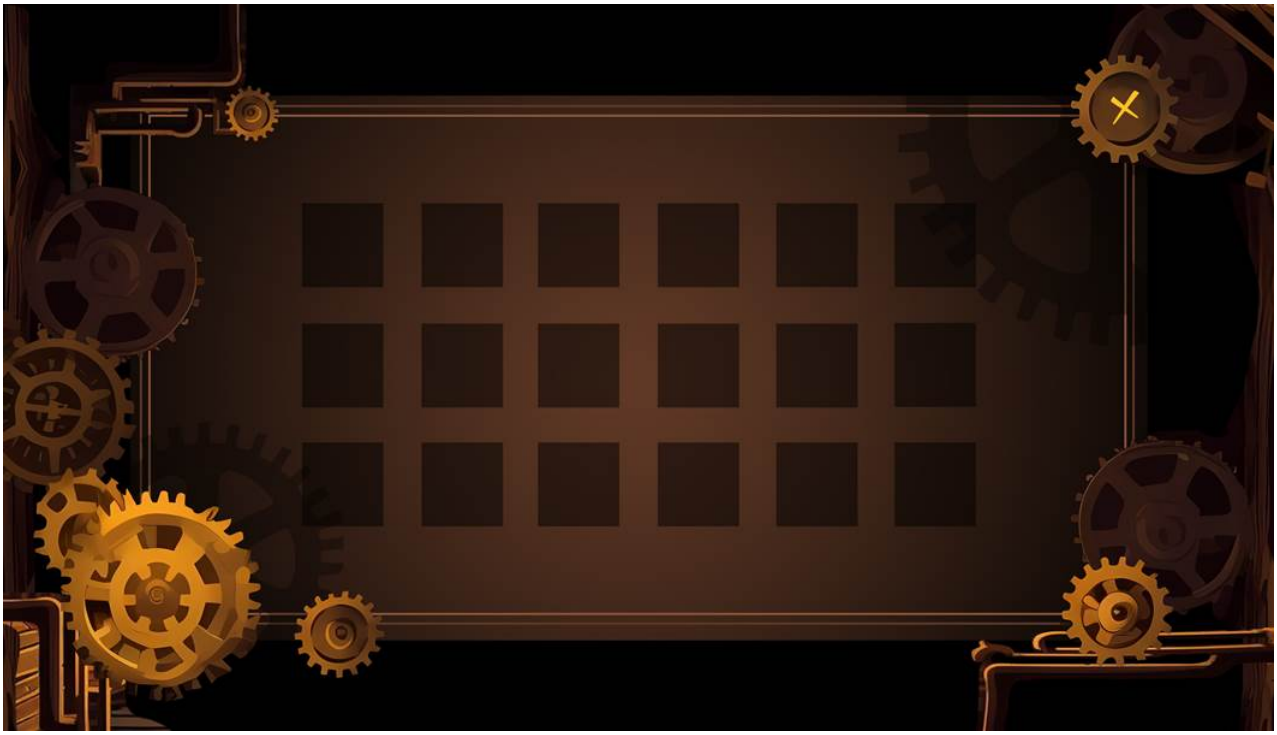


Рис. 2.29. Пустий інвентар

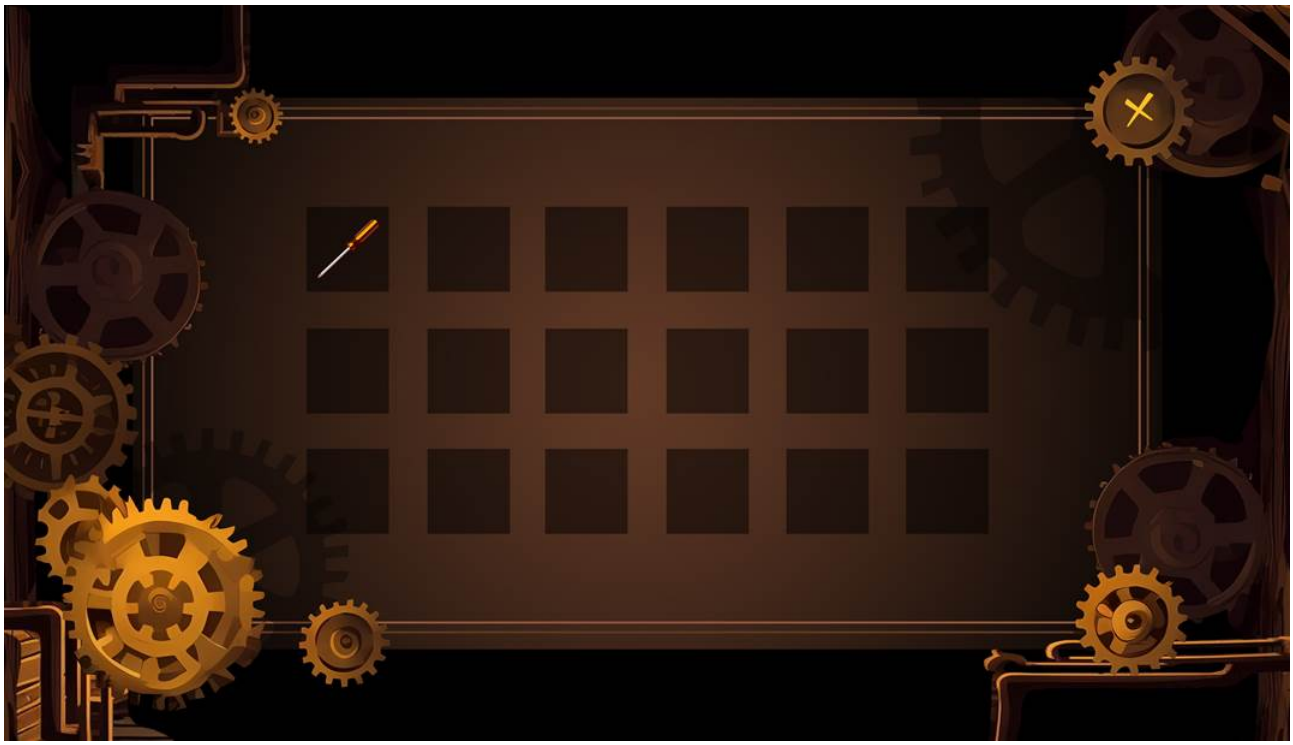


Рис. 2.30. Нові речі у інвентарі

Також у інвентарі є 3 дії, які можна робити з предметами (рис. 2.31.): «Використати», «Інформація» (рис. 2.32.), «Об’єднати з…».

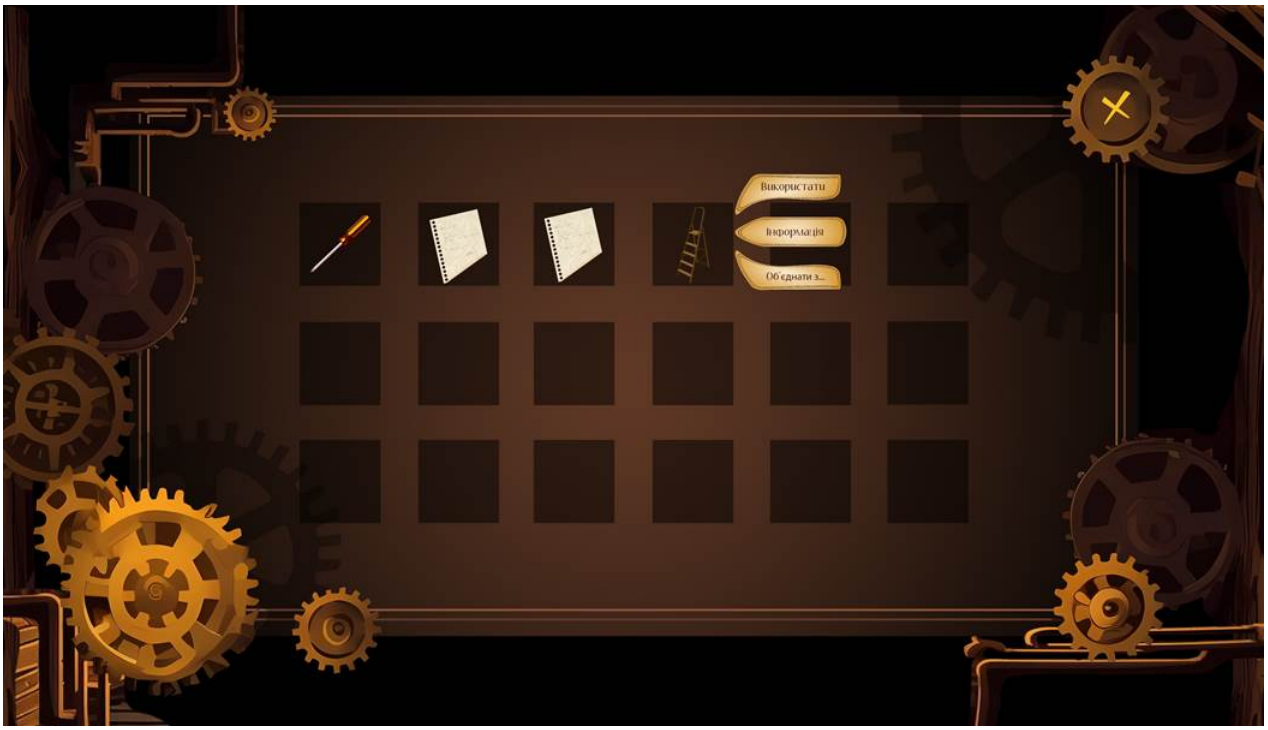


Рис. 2.31. Функції інвентаря

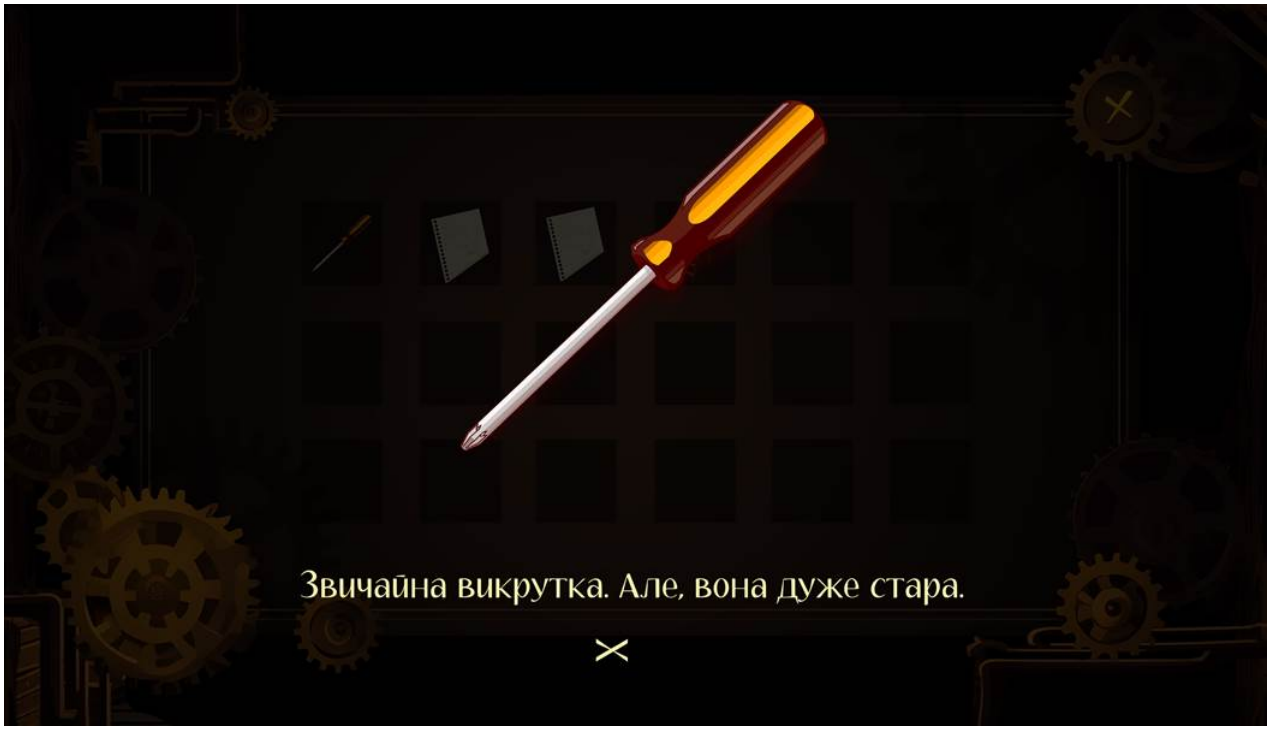


Рис. 2.32. Інформація про предмет

В додатку присутні головоломки та міні-ігри. Панель з міні-грою виглядає так (рис. 2.33):

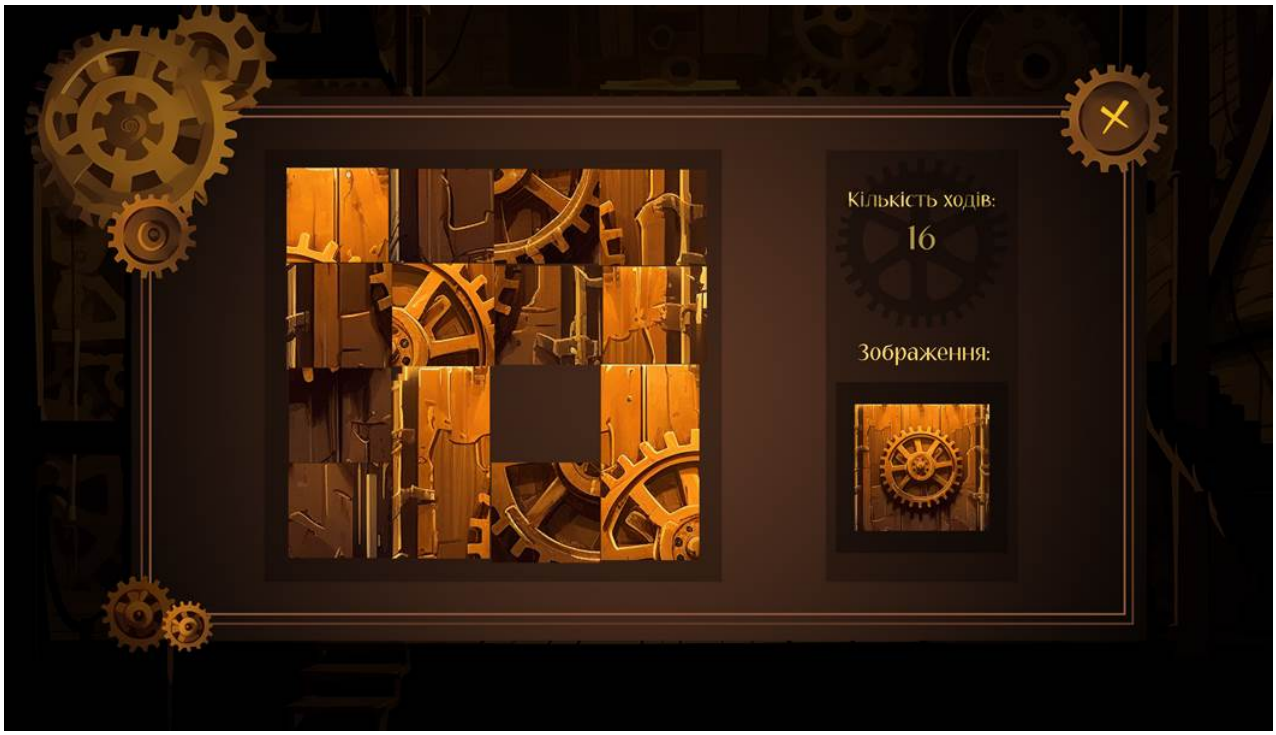


Рис. 2.33. Панель з міні-грою

РОЗДІЛ 3
ЕКОНОМІЧНИЙ РОЗДІЛ

3.1. Розрахунок трудомісткості та вартості розробки програмного продукту

Вхідні дані:

1) передбачуване число операторів – 1480;
2) коефіцієнт корекції програми в ході її розробки – 0,1;
3) коефіцієнт складності програми – 1,4;
4) годинна заробітна плата програміста – 207,8 грн/год;
5) коефіцієнт збільшення витрат праці в наслідок недостатнього опису задачі – 1,25;
6) коефіцієнт кваліфікації програміста, обумовлений від стажу роботи з даної спеціальності – 1;
7) вартість машино-години ЕОМ – 9,1 грн/год;

Заробітна плата за годину Junior C#/Unity Developer була вирахувана згідно «Української спільноти програмістів (DOU)» [22]. Станом на 2023 рік зарплата Junior C#/Unity Developer вар'юється від 800$ до 1200$. Вирахувавши середню заробітну плату програміста маємо плату 1000$ у місяць. При курсі валют НБУ на кінець грудня 2023 року один американський долар дорівнює 36,57 грн, тому середня зарплата в гривнях дорівнює 36 570 грн. При стандартному графіку (176 годин/місяць) зарплата за годину буде становити близько 207,8 грн.

Виходячи з того, що для розробки ігрового додатку необхідна значна потужність ПК, вдалим рішенням буде оренда ноутбуку.

Вартість оренди потужного ноутбуку на місяць становить 1600 грн [23]. При стандартному графіку (176 годин/місяць) вартість машино-години ЕОМ за годину роботи буде становити 9,1 грн.

Нормування праці в процесі створення ПЗ істотно ускладнено в силу творчого характеру праці програміста. Тому трудомісткість розробки ПЗ може

бути розрахована на основі системи моделей з різною точністю оцінки.

Трудомісткість розробки ПЗ можна розрахувати за формулою:

$$t = t_o + t_u + t_a + t_n + t_{\text{отл}} + t_{\text{д}}, \quad (3.1)$$

де t_o - витрати праці на підготовку й опис поставленої задачі (приймається 50 людино-годин);

t_u - витрати праці на дослідження алгоритму рішення задачі;

t_a - витрати праці на розробку блок-схеми алгоритму;

t_n - витрати праці на програмування по готовій блок-схемі;

$t_{отл}$ - витрати праці на налагодження програми на ЕОМ;

$t_д$ - витрати праці на підготовку документації.

Складові витрати праці визначаються через умовне число операторів у програмному забезпеченні, яке розробляється.

Умовне число операторів (підпрограм):

$$Q = q * C * (1 + p), \quad (3.2)$$

де q - передбачуване число операторів (1480);

C - коефіцієнт складності програми (1,4);

p - коефіцієнт корекції програми в ході її розробки (0,1).

За формулою (3.2) умовне число операторів в програмі становить:

$$Q = 1480 * 1{,}4 * (1 + 0{,}1) = 2279 \quad (3.3)$$

Витрати праці на вивчення опису задачі t_u визначається з урахуванням уточнення опису і кваліфікації програміста:

$$t_u = \frac{Q*B}{(75..85)*k}, \quad (3.4)$$

де В - коефіцієнт збільшення витрат праці внаслідок недостатнього опису задачі (1,25);

k - коефіцієнт кваліфікації програміста, обумовлений від стажу роботи з даної спеціальності. При стажі роботи від 2 до 3 років він складає 1.

З урахуванням коефіцієнта кваліфікації k = 1, за формулою (3.4) отримуємо витрати праці на вивчення опису завдання:

$$t_u = \frac{2279*1,25}{85*1} = 33,5, \text{ людино-годин,} \quad (3.5)$$

Витрати праці на розробку алгоритму рішення задачі визначаються за формулою:

$$t_a = \frac{Q}{(20...25)*k}, \quad (3.6)$$

де Q – умовне число операторів програми (2279);

k – коефіцієнт кваліфікації програміста (1).

Згідно формулі (3.6), отримаємо:

$$t_a = \frac{2279}{25*1} = 91,2, \text{ людино-годин,} \quad (3.7)$$

Витрати на складання програми по готовій блок-схемі:

$$t_n = \frac{Q}{(20...25)*k}, \quad (3.8)$$

Маючи формулу (3.8) витрати на складання програми по готовій блок-схемі:

$$t_n = \frac{2279}{25*1} = 91,2, \text{ людино-годин,} \quad (3.9)$$

Витрати праці на налагодження програми на ЕОМ:

– за умови автономного налагодження одного завдання:

$$t_{\text{отл}} = \frac{Q}{(4..5)*k}, \tag{3.10}$$

Витрати праці на налагодження програми на ЕОМ за умови автономного налагодження одного завдання за формулою (3.10):

$$t_{\text{отл}} = \frac{2279}{5*1} = 455{,}8, \text{ людино-годин}, \tag{3.11}$$

– за умови комплексного налагодження завдання:

$$t_{\text{отл}}^{k} = 1{,}5 * t_{\text{отл}}, \tag{3.12}$$

Витрати праці на налагодження програми на ЕОМ за умови комплексного налагодження завдання за формулою (3.12):

$$t_{\text{отл}}^{k} = 1{,}5 * 455{,}8 = 683{,}7, \text{ людино-годин}, \tag{3.13}$$

Витрати праці на підготовку документації визначаються за формулою:

$$t_{\text{д}} = t_{\text{др}} + t_{\text{до}}, \tag{3.14}$$

де $t_{\text{др}}$ - трудомісткість підготовки матеріалів і рукопису:

$$t_{\text{др}} = \frac{Q}{(15..20)*k}, \tag{3.15}$$

$t_{\text{до}}$ - трудомісткість редагування, печатки й оформлення документації:

$$t_{до} = 0{,}75 * t_{др}, \quad (3.16)$$

Маючи формули (3.15), (3.16) та (3.14) обчислюємо витрати праці на документацію:

$$t_{др} = \frac{2279}{20*1} = 113{,}95, \text{ людино-годин}, \quad (3.17)$$

$$t_{до} = 0{,}75 * 113{,}95 = 85{,}46, \text{ людино-годин}, \quad (3.18)$$

$$t_{д} = 113{,}95 + 85{,}46 = 199{,}41, \text{ людино-годин}, \quad (3.19)$$

Повертаючись до формули (3.1), отримаємо повну оцінку трудомісткості розробки програмного забезпечення:

$$t = 50 + 33{,}5 + 91{,}2 + 91{,}2 + 455{,}8 + 199{,}41 = 921{,}1, \text{ людино-годин.} \quad (3.20)$$

3.2. Розрахунок витрат на створення програми

Витрати на створення ПЗ КПО включають витрати на заробітну плату виконавця програми $З_{ЗП}$ і витрат машинного часу, необхідного на налагодження програми на ЕОМ:

$$К_{ПО} = З_{ЗП} + З_{МВ}, \text{ грн.} \quad (3.21)$$

Заробітна плата виконавців визначається за формулою:

$$З_{ЗП} = t * С_{ПР}, \text{ грн}, \quad (3.22)$$

де t - загальна трудомісткість, людино-годин (921,1);

$С_{ПР}$ - середня годинна заробітна плата програміста, грн/година.

З урахуванням того, що середня годинна зарплата програміста становить 207,8 грн / год, за формулою (3.22) отримуємо:

$$З_{ЗП} = 921{,}1 * 207{,}8 = 191\ 404{,}6, \text{ грн}, \quad (3.23)$$

Вартість машинного часу, необхідного для налагодження програми на ЕОМ, визначається за формулою:

$$З_{МВ} = t_{отл} * С_{Мч}, \text{ грн}, \quad (3.24)$$

де $t_{отл}$ - трудомісткість налагодження програми на ЕОМ, год (455,8 год);

$С_{Мч}$ - вартість машино-години ЕОМ, грн/год (9,1 грн/год).

Підставивши в формулу (3.24) відповідні значення, обчислюємо вартість необхідного для налагодження машинного часу:

$$З_{МВ} = 455{,}8 * 9{,}1 = 4\ 147{,}8, \text{ грн}, \quad (3.25)$$

Звідси, за формулою (3.21), витрати на створення програмного продукту:

$$К_{ПО} = 191\ 404{,}6 + 4\ 147{,}8 = 195\ 552{,}4, \text{ грн}, \quad (3.26)$$

Очікуваний період створення програмного застосунку:

$$T = \frac{t}{B_k * F_p}, \text{ міс}, \quad (3.27)$$

де B_k - число виконавців (1);

F_p - місячний фонд робочого часу (при 40 годинному робочому тижні F_p =176 годин).

За формулою 3.27 очікуваний період створення програмного забезпечення:

$$T = \frac{921,1}{1*176} \approx 5,23 \text{ міс.} \quad (3.28)$$

Висновки: ігровий додаток був розроблений для надання гравцям захоплюючи випробування та інтелектуальні виклики. Розробка даного програмного забезпечення склала 195 552,4 грн без додаткових витрат. Очікуваний термін розробки становить 921,1 години, або 5,23 місяці. Цей термін стосується кількості операторів, який включає в себе час, витрачений на вивчення та розробку алгоритму для вирішення поставленої задачі, програмування з використанням готового алгоритму, налагодження, підготовку та впровадження документації.

ВИСНОВКИ

У ході виконання даної кваліфікаційної роботи було успішно розроблено багаторівневу відеогру на основі кросплатформного рушія Unity. Інструменти, які використовувалися для розробки програми: ігровий рушій Unity та редактор коду Visual Studio. Також, допоміжними інструментами виступили Adobe Photoshop і SAI для створення ігрової графіки.

Основним призначенням ігрового додатку є надання гравцю гарного емоційного досвіду, цікавої історії, захоплюючої механіки дій та різноманітних загадок за рахунок того, що гра має жанр квест-головоломка. Також, це сприятиме розвитку когнітивних навичок у гравця за рахунок великої кількості інтерактивних елементів та об'єктів, між якими потрібно будувати логічний зв'язок.

У ході розробки використовувалися стандартні бібліотеки Unity (JsonUtility, PlayerPrefs) для роботи з файлами JSON та системою збереження/завантаження і стороння бібліотека (DOTween) для роботи з анімаціями.

Під час виконання даної кваліфікаційної роботи було проведено детальний аналіз трудомісткості розробленої системи та розрахована вартість роботи зі створення програми. За середньою заробітною платою розробника, встановленою на рівні 207,8 грн/час, було розраховано загальну вартість проекту 195 552,4 грн. Крім того, визначено, що час, витрачений на розробку програмного додатку, складає 921,1 людино-годин, що приблизно дорівнює 5,23 місяцям.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Коротка історія відеоігор. URL: https://hub.yamaha.com/audio /gaming/a -brief-history-of-video-games/ (дата звернення: 10.05.2023)

2. Історія відеоігор. URL: https://uk.wikipedia.org/wiki/%D0%86 %D1%81%D1%82%D0%BE%D1%80%D1%96%D1%8F_%D0%B2%D1%96%D0 %B4%D0%B5%D0%BE%D1%96%D0%B3%D0%BE%D1%80 (дата звернення: 10.05.2023)

3. Ігрова індустрія до 2020 року — мільярдні прибутки, епоха мобільних ігор та спортивний бізнес. URL: https://nachasi.com/creative/2018 /08/20/igrova-industriya-do-2020/ (дата звернення: 10.05.2023)

4. Жанри відеоігор. URL: https://uk.wikipedia.org/wiki/%D0%96%D0 %B0%D0%BD%D1%80%D0%B8_%D0%B2%D1%96%D0%B4%D0%B5%D0%B E%D1%96%D0%B3%D0%BE%D1%80 (дата звернення: 11.05.2023)

5. 13 ПЕРЕВАГ ТА НЕДОЛІКІВ ВІДЕОІГОР - ТЕХНОЛОГІЯ – 2023. URL: https://uk.sperohope.com/13-ventajas-y-desventajas-de-los-videojuegos (дата звернення: 11.05.2023)

6. Усі жанри комп’ютерних ігор. URL: https://uaplay.com.ua/usi-zhanry-pc-ihor/ (дата звернення: 13.05.2023)

7. Розвиток головного мозку за допомогою комп’ютерних ігор. URL: https://moezdorovia.com.ua/rozvitok-golovnogo-mozku-za-dopomogoyu-kompyuter ni h -gor (дата звернення: 13.05.2023)

8. Переваги ігор-головоломок. 10 найкращих ігор-головоломок для Android для вдосконалення навичок вирішення проблем. URL: https://techukraine.net/10-%D0%BD%D0%B0%D0%B9%D0%BA%D1%80%D0 %B0%D1%89%D0%B8%D1%85-%D1%96%D0%B3%D0%BE%D1%80-%D0 %B3%D0%BE%D0%BB%D0%BE%D0%B2%D0%BE%D0%BB%D0%BE%D0% BC%D0%BE%D0%BA-%D0%B4%D0%BB%D1%8F-android- %D0%B4%D0%BB%D1%8F/ (дата звернення: 13.05.2023)

9. Головоломка (жанр відеоігор). URL: https://uk.wikipedia.org/wiki/%

D0%93%D0%BE%D0%BB%D0%BE%D0%B2%D0%BE%D0%BB%D0%BE%D0%BC%D0%BA%D0%B0_(%D0%B6%D0%B0%D0%BD%D1%80_%D0%B2%D1%96%D0%B4%D0%B5%D0%BE%D1%96%D0%B3%D0%BE%D1%80) (дата звернення: 13.05.2023)

10. Patrick Felicia. Unity From Zero to Proficiency (Foundations): A step-by-step guide to creating your first game. Independently published, 2019. 234 p.

11. Paris Buttfield-Addison. Unity Game Development Cookbook: Essentials for Every Game. O'Reilly Media, 2019. 405 p.

12. Підсказки по C#: https://metanit.com/ (дата звернення: 25.05.2023)

13. TextMesh Pro Documentation. URL: https://docs.unity3d.com/Packages/com.unity.textmeshpro@3.2/manual/index.html (дата звернення: 20.05.2023)

14. UnityUI. URL: https://docs.unity3d.com/Manual/com.unity.ugui.html (дата звернення: 20.05.2023)

15. DoTween Documentation. URL: http://dotween.demigiant.com/documentation.php (дата звернення: 25.05.2023)

16. JsonUtility Documentation. URL: https://docs.unity3d.com/ScriptReference/JsonUtility.html (дата звернення: 25.05.2023)

17. PlayerPrefs Documentation. URL: https://docs.unity3d.com/ScriptReference/PlayerPrefs.html (дата звернення: 25.05.2023)

18. Bauer, Peter. Adobe Photoshop CC for dummies. John Wiley & Sons, 2021. 423 p.

19. SAI Overview. URL: https://systemax.jp/en/sai/ (дата звернення: 25.05.2023)

20. RB Whitaker. The C# Player's Guide (5th Edition). Starbound Software, 2022. 495 p.

21. Harrison Ferrone. Learning C# by Developing Games with Unity: Get to grips with coding in C# and build simple 3D games in Unity 2022 from the ground up, 7th Edition. Packt Publishing, 2022. 460 p.

22. Заробітна плата розробників ігор. URL: https://gamedev.dou.ua/

articles/gamedev-salaries-winter-2023/ (дата звернення: 30.05.2023)

23. Аренда ноутбуків. URL: https://prom.ua/ua/p1541921868-arenda-moschnogo-noutbuka.html (дата звернення: 30.05.2023)

ДОДАТОК А

КОД ПРОГРАМИ

Файл StoryLine.cs:

```
using System;

[Serializable]
public class StoryLine
{
    public int lineID;
    public string location;
    public string dialog;
    public string memberOfDialog;
    public string action;
    public string objectOfAction;
}
```

Файл Story.cs:

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

public class Story : MonoBehaviour
{
    [SerializeField]
    private TextAsset _fileStoryLine;
    private StoryLine[] storyLines;

    public int currentStoryLine { get; private set; }

    private void Initialization()
    {
        storyLines = JsonHandler.FromJson<StoryLine>(_fileStoryLine.text);
        currentStoryLine = LoadCurrentStoryLine();
    }

    public void NextStoryLine()
    {
        if (currentStoryLine != storyLines[storyLines.Length - 1].lineID)
            currentStoryLine += 1;
    }

    public void SaveCurrentStoryLine()
    {
        int currentPlayer = PlayerPrefs.GetInt("currentPlayer");
        PlayerPrefs.SetInt("CurrentStoryLine" + currentPlayer, currentStoryLine);
    }
```

```
    public int LoadCurrentStoryLine()
    {
        int currentPlayer = PlayerPrefs.GetInt("currentPlayer");
        return PlayerPrefs.GetInt("CurrentStoryLine"+ currentPlayer);
    }

    public StoryLine GetCurrentStoryLine()
    {
        if (storyLines == null) Initialization();
        return storyLines[currentStoryLine];
    }
}
```

Файл Item.cs:

```
using System;

[Serializable]
public class Item
{
    public int lineID;
    public int storyLineID;
    public bool availability;
    public string itemName;
    public string description;
    public string reaction;
    public string itemOfInfluence;
    public string itemForApplication;
    public string itemsForCombination;
    public string itemCombinationText;
    public string itemResultOfCombination;
}
```

Файл ItemManager.cs:

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

public class ItemManager : MonoBehaviour
{
    [SerializeField]
    private TextAsset _itemsList;
    [SerializeField]
    private GameObject _itemReactionWindow;
    [SerializeField]
    private TextMeshProUGUI _itemReactionText;

    public Item[] items { get; private set; }
    public List<string> takenItems { get; private set; }
```

```
public List<string> putItems { get; private set; }

private void Start()
{
   Initialization();
}

private void Initialization()
{
   items = JsonHandler.FromJson<Item>(_itemsList.text);
   LoadItems(takenItems, "takenItems");
   LoadItems(putItems, "putItems");
}

private void LoadItems(List<string> items, string nameOfList)
{
   string currentList = nameOfList + PlayerPrefs.GetInt("currentPlayer");
   string[] allItems = PlayerPrefs.GetString(currentList).Split(',');
   items = new List<string>(allItems);
}

private void SaveItems(List<string> items, string nameOfList)
{
   string currentList = nameOfList + PlayerPrefs.GetInt("currentPlayer");
   string allItems = string.Join(",", items);
   PlayerPrefs.SetString(currentList, allItems);
}

private void TakeItem(GameObject item)
{
   takenItems.Add(item.name);
   item.SetActive(false);
   SaveItems(takenItems, "takenItems");
}

private void UseItem(GameObject item)
{
   putItems.Add(item.name);
   takenItems.Remove(item.name);
   SaveItems(putItems, "putItems");
   item.SetActive(true);
}

private void ChangeAvailability(Item item)
{
   if (item.itemOfInfluence != " ")
   {
      Item itemForInfluence = Array.Find(items,
         i => i.itemName == item.itemOfInfluence);

      itemForInfluence.availability = true;
   }
```

```
    }

    private void ShowReactionOnItem(string itemReaction)
    {
        _itemReactionText.text = itemReaction;
        _itemReactionWindow.SetActive(true);
    }

    public void HideReactionOnItem()
    {
        _itemReactionWindow.SetActive(false);
    }

    public void CheckItemAvailability(GameObject item)
    {
        Item currentItem = Array.Find(items,
                i => i.itemName == item.name);

        if (currentItem.availability)
        {
            TakeItem(item);
            ChangeAvailability(currentItem);
        }
        else if(currentItem.reaction != " ")
        {
            ShowReactionOnItem(currentItem.reaction);
        }
    }

    public void CheckItemUsage(GameObject item)
    {
        string itemForUsing = PlayerPrefs.GetString("currentItem");

        Item currentItem = Array.Find(items,
                i => i.itemName == item.name);

        if (currentItem.itemForApplication == itemForUsing)
        {
            UseItem(item);
            ChangeAvailability(currentItem);
        }
        else
        {
            ShowReactionOnItem("Предмети не можна поєднати");
        }
    }
}
```

Файл MenuManager.cs:

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

public class MenuManager : MonoBehaviour
{
    [SerializeField]
    private List<GameObject> _newGameButtons;

    public void Initialization()
    {
        for (int i = 0; i < _newGameButtons.Count; i++)
        {
            if (PlayerPrefs.GetString("Game" + i) == "exist")
            {
                _newGameButtons[i].SetActive(false);
            }
        }
    }

    public void CreateNewGame(int gameID)
    {
        PlayerPrefs.SetInt("currentPlayer", gameID);
        PlayerPrefs.SetString("Game" + gameID, "exist");
        PlayerPrefs.SetInt("CurrentStoryLine" + gameID, 0);
        SceneManager.LoadScene("GameScreen");
    }

    public void DeleteGame(int gameID)
    {
        _newGameButtons[gameID].SetActive(true);
        PlayerPrefs.SetString("Game" + gameID, "doesnt exist");
    }

    public void LoadGame(int gameID)
    {
        PlayerPrefs.SetInt("currentPlayer", gameID);
        SceneManager.LoadScene("GameScreen");
    }
}
```

Файл ScreenLogic.cs:

```
using System.Collections.Generic;
using DG.Tweening;
using UnityEngine;
using UnityEngine.UI;

public class ScreenLogic : MonoBehaviour
{
    [SerializeField]
```

```
    private List<GameObject> _screensPrefabs, _windowsPrefabs;
    [SerializeField]
    private GameObject _screens, _windows;

    public void OpenScreenOrWindow(int nameID)
    {
        // nameID of screens is 100...199, nameID of windows is 200...299

        if (nameID >= 200)
        {
            GameObject window = Instantiate(_windowsPrefabs[nameID % 100],
_windows.transform);            //Creating of window
            GameObject background = window.transform.GetChild(0).gameObject;
            Transform panel = window.transform.GetChild(1);

            background.GetComponent<Image>().color = new Color(0f,0f,0f,0f);
//Set start alpha for background
            panel.localScale = new Vector3(0.5f, 0.5f, 0.5f);                                    //Set start size
for panel with content

            background.GetComponent<Image>().DOFade(0.82f, 0.4f);
//Appearance of background
            panel.DOScale(new Vector3(1.0f, 1.0f, 1.0f), 0.4f);
//Appearance of panel with content
        }
        else Instantiate(_screensPrefabs[nameID % 100], _screens.transform);
//Creating of screen
    }

    public void CloseScreenOrWindow(int nameID)
    {
        Transform parent = nameID < 200 ? _screens.transform : _windows.transform;
        GameObject child = parent.GetChild(parent.childCount - 1).gameObject;

        if (nameID >= 200)
        {
            child.transform.GetChild(0).GetComponent<Image>().DOFade(0f, 0.3f);
//Disappearance of background
            child.transform.GetChild(1).DOScale(new Vector3(0.3f, 0.3f, 0.3f), 0.3f);
//Disappearance of panel with content
            Destroy(child, 0.31f);
        }
        else Destroy(child);
    }
}
```

Файл DialogSystem.cs:

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

```
using UnityEngine;
using DG.Tweening;
using UnityEngine.UI;

public class DialogSystem : MonoBehaviour
{
    [SerializeField]
    private TextMeshProUGUI _thoughtsOfCharacter;
    [SerializeField]
    private TextMeshProUGUI _dialogTextOfCharacter;
    [SerializeField]
    private TextMeshProUGUI _dialogTextOfNPS;

    [SerializeField]
    private GameObject _dialogWindowOfMindCharacter;
    [SerializeField]
    private GameObject _dialogWindowOfCharacter;
    [SerializeField]
    private GameObject _dialogWindowOfNPS;

    private void SetText(TextMeshProUGUI dialogText, StoryLine storyLine)
    {
        dialogText.text = storyLine.dialog;
    }

    private void ShowDialog(GameObject dialogWindow)
    {
        dialogWindow.SetActive(true);

        GameObject background = dialogWindow.transform.GetChild(0).gameObject;
        Transform panel = dialogWindow.transform.GetChild(1);

        background.GetComponent<Image>().color = new Color(0f, 0f, 0f, 0f);
        panel.localScale = new Vector3(0.5f, 0.5f, 0.5f);

        background.GetComponent<Image>().DOFade(0.82f, 0.4f);
        panel.DOScale(new Vector3(1.0f, 1.0f, 1.0f), 0.4f);
    }

    private void HideDialog(GameObject dialogWindow)
    {
        GameObject background = dialogWindow.transform.GetChild(0).gameObject;
        Transform panel = dialogWindow.transform.GetChild(1);

        background.GetComponent<Image>().color = new Color(0f, 0f, 0f, 0f);
        panel.localScale = new Vector3(0.5f, 0.5f, 0.5f);

        background.GetComponent<Image>().DOFade(0f, 0.3f);
        panel.DOScale(new Vector3(0.3f, 0.3f, 0.3f), 0.3f);

        dialogWindow.SetActive(false);
    }
```

```
    public void SetCurrentDialog(StoryLine storyLine)
    {
        switch (storyLine.memberOfDialog)
        {
            case "Character":
                SetText(_dialogTextOfCharacter, storyLine);
                ShowDialog(_dialogWindowOfCharacter);
                break;

            case "MindOfCharacter":
                SetText(_thoughtsOfCharacter, storyLine);
                ShowDialog(_dialogWindowOfMindCharacter);
                break;

            case "NPS":
                SetText(_dialogTextOfNPS, storyLine);
                ShowDialog(_dialogWindowOfNPS);
                break;
        }
    }

    public void HideCurrentDialog(StoryLine storyLine)
    {
        switch (storyLine.memberOfDialog)
        {
            case "Character":
                HideDialog(_dialogWindowOfCharacter);
                break;

            case "MindOfCharacter":
                HideDialog(_dialogWindowOfMindCharacter);
                break;

            case "NPS":
                HideDialog(_dialogWindowOfNPS);
                break;
        }
    }
}
```

Файл Inventory.cs:

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

public class Inventory : MonoBehaviour
{
```

```
[SerializeField]
private ItemManager _item;
[SerializeField]
private ScreenLogic _screenLogic;

[SerializeField]
private List<GameObject> _itemsPropertiesCells;
[SerializeField]
private GameObject _combinationWindow;

[SerializeField]
private List<Image> _itemsCellsImages;
[SerializeField]
private Image[] _itemImages;
[SerializeField]
private Image _clearImage;
[SerializeField]
private Image _combinationImage;
[SerializeField]
private TextMeshProUGUI _text;

private bool combinationState;
private string currentItem;

private void Start()
{
   LoadItems();
}

private void LoadItems()
{
   currentItem = "";

   for (int j = 0; j < _item.takenItems.Count; j++)
   {
      Image currentImage = Array.Find(_itemImages,
            i => i.name == _item.takenItems[j]);

      _itemsCellsImages[j].sprite = currentImage.sprite;
   }
}

public void TakeItemFromInventory(int cellsID)
{
   string itemName = _item.takenItems[cellsID];

   if (!combinationState)
   {
      currentItem = itemName;
      PlayerPrefs.SetString("currentItem", currentItem);
   }
}
```

```
public void ClearCell(int id)
{
   if (!combinationState)
      _itemsCellsImages[id].sprite = _clearImage.sprite;
}

public void CombineItems(int cellsID)
{
   if (!combinationState)
   {
      TakeItemFromInventory(cellsID);
      combinationState = true;
   }
}

public void CheckItemsCombination(int cellsID)
{
   if(combinationState)
   {
      string itemName = _item.takenItems[cellsID];

      Item item = Array.Find(_item.items,
         i => i.itemName == currentItem);

      if(item.itemsForCombination == itemName)
      {
         Image currentImage = Array.Find(_itemImages,
           i => i.name == item.itemResultOfCombination);

         _combinationImage.sprite = currentImage.sprite;
         _text.text = item.itemCombinationText;

         _item.takenItems.Add(item.itemResultOfCombination);
         _item.takenItems.Remove(item.itemName);
         _item.takenItems.Remove(itemName);

         _combinationWindow.SetActive(true);
      }
      else
      {
         _screenLogic.OpenScreenOrWindow(110);
      }

      combinationState = false;
   }

}

public void ShowItemsProperties(int cellsID)
{
   if(!combinationState)
```

```
        {
            for (int i = 0; i < _itemsPropertiesCells.Count; i++)
            {
                if (_itemsPropertiesCells[i].activeSelf)
                    _itemsPropertiesCells[i].SetActive(false);
            }

            _itemsPropertiesCells[cellsID].SetActive(true);
        }
    }

    public void HideItemsProperties(int cellsID)
    {
        _itemsPropertiesCells[cellsID].SetActive(false);
    }
}
```

Файл StoryManager.cs:

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

public class StoryManager : Singleton<MonoBehaviour>
{
    [SerializeField]
    private Story _story;
    [SerializeField]
    private DialogSystem _dialog;
    [SerializeField]
    private ItemManager _item;

    [SerializeField]
    private List<GameObject> _locationsList;
    [SerializeField]
    private GameObject _itemsList;
    [SerializeField]
    private GameObject _itemListPut;
    [SerializeField]
    private GameObject _barier;

    [SerializeField]
    private List<Animation> _animations;

    private void Start()
    {
        Initialization();
    }

    private void Initialization()
```

```
{
    DontDestroyOnLoad(this.gameObject);
    LoadLocation();
    LoadItems();
    SetCurrentStoryLine();
}

public void SetCurrentStoryLine()
{
    CheckStoryLine();
}

private void LoadItems()
{
    for (int i = _itemsList.transform.childCount; i > 0; i--)
    {
        if (_itemsList.transform.GetChild(i).name == _item.takenItems[i])
            Destroy(_itemsList.transform.GetChild(i).gameObject);
    }

    for (int i = _itemListPut.transform.childCount; i > 0; i--)
    {
        if (_itemListPut.transform.GetChild(i).name != _item.putItems[i])
            Destroy(_itemListPut.transform.GetChild(i).gameObject);
    }
}

private void LoadLocation()
{
    StoryLine storyLine = _story.GetCurrentStoryLine();
    SceneManager.LoadScene(storyLine.location);
}

private void CheckStoryLine()
{
    StoryLine storyLine = _story.GetCurrentStoryLine();

    if (storyLine.action != " ")
    {
        DoAction(storyLine.action);
    }
    else if (storyLine.dialog != " ")
    {
        _barier.SetActive(true);
        _dialog.SetCurrentDialog(storyLine);
    }
}

private void DoAction(string actionName)
{
    switch (actionName)
    {
```

```
case "Awake":
    _animations[0].Play();
    break;

case "Climb":
    _animations[1].Play();
    break;

case "StopClock":
    _animations[2].Play();
    break;

case "Shake":
    _animations[3].Play();
    break;

case "ControlPanel1Work":
    _animations[4].Play();
    break;

case "MoveCameraLeft":
    _animations[5].Play();
    _animations[3].Play();
    break;

case "MoveCameraRight":
    _animations[6].Play();
    _animations[3].Play();
    break;

case "SetTag":
    _animations[7].Play();
    break;

case "OpenSmallDoor":
    _animations[8].Play();
    break;

case "OpenBigDoor":
    _animations[9].Play();
    _animations[10].Play();
    break;

case "MoveDoor":
    _animations[11].Play();
    break;

case "MoveBox":
    _animations[12].Play();
    break;

case "FixRobot":
```

```
            _animations[13].Play();
            _animations[14].Play();
            _animations[15].Play();
            break;

        case "OpenVentilation":
            _animations[16].Play();
            _animations[17].Play();
            break;
    }
  }

  private void CheckEndOfStoryLine()
  {
    StoryLine storyLine = _story.GetCurrentStoryLine();

    if (storyLine.dialog != " ")
    {
      _barier.SetActive(false);
      _dialog.HideCurrentDialog(storyLine);
    }
  }

  public void NextStoryAction()
  {
    CheckEndOfStoryLine();
    _story.NextStoryLine();
    CheckStoryLine();
  }
}
```

Файл AnimationController.cs:

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

public class AnimationController : MonoBehaviour
{
  [SerializeField]
   private List<Animation> _animations;

  private GameObject currentAnimation;

  public void StartAnimation(string nameOfAnimation)
  {
    GameObject animation = GetAnimationByName(nameOfAnimation);
    if (animation != null)
    {
      if (currentAnimation != null)
        StopCurrentAnimation();

      currentAnimation = animation;
```

```
            currentAnimation.SetActive(true);
        }
    }

    public void StopAnimation(string nameOfAnimation)
    {
        GameObject animation = GetAnimationByName(nameOfAnimation);
        if (animation != null && animation == currentAnimation)
        {
            StopCurrentAnimation();
        }
    }

    public void ChangeAnimation(string nameOfAnimation)
    {
        StopCurrentAnimation();
        StartAnimation(nameOfAnimation);
    }

    private void StopCurrentAnimation()
    {
        if (currentAnimation != null)
        {
            currentAnimation.SetActive(false);
            currentAnimation = null;
        }
    }

    private GameObject GetAnimationByName(string name)
    {
        foreach (GameObject animation in _animations)
        {
            if (animation.name == name)
                return animation;
        }

        return null;
    }
}
```

Файл AreaOfMoving.cs:

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

public class AreaOfMoving : MonoBehaviour
{
    [SerializeField]
    private GameObject _character;
```

```
    private void OnMouseDown()
    {
        Vector3 position = Camera.main.ScreenToWorldPoint(Input.mousePosition);
        float time = (position - _character.transform.position).magnitude/5.5f;
        _character.transform.DOMove(new Vector3(position.x,position.y,0f), time);
    }
}
```

Файл Buttons.cs:

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

public class Buttons : MonoBehaviour
{
    [SerializeField]
    private GameObject _eyeButton, ,;
    [SerializeField]
    private GameObject _book1;
    [SerializeField]
    private GameObject _book2;
    [SerializeField]
    private Sprite _eye;
    [SerializeField]
    private Sprite _eyeHide;

    private GameObject description;

    private ScreenLogic screenLogic;

    private void Start()
    {
        screenLogic = GameObject.Find("GameManager").GetComponent<ScreenLogic>();
    }

    public void OpenWindow(int nameID)
    {
        screenLogic.OpenScreenOrWindow(nameID);
    }

    public void CloseWindow(int nameID)
    {
        screenLogic.CloseScreenOrWindow(nameID);
    }

    public void HideButtons(GameObject button)
    {
        button.SetActive(false);
    }
```

```
    public void ActivateButton(GameObject button)
    {
        button.SetActive(true);
    }

    public void OpenBook()
    {
        if (PlayerPrefs.GetString("language") == "eng")
            OpenWindow(_book2);
        else
            OpenWindow(_book1);
    }

    public void HideInterface(GameObject intr)
    {
        intr.SetActive(!intr.activeSelf);
        if(intr.activeSelf) _eyeButton.GetComponent<Image>().sprite = _eye;
        else _eyeButton.GetComponent<Image>().sprite = _eyeHide;
    }

    public void HideDescription()
    {
        if (GameObject.Find("DescriptionsOfBones") != null)
            description = GameObject.Find("DescriptionsOfBones");

        description.SetActive(!description.activeSelf);
    }

    public void ChoseLocation(string name_of_scene)
    {
        SceneManager.LoadScene(name_of_scene);
    }
}
```

Файл Cell.cs:

```
using UnityEngine;
using UnityEngine.UI;

public class Cell : MonoBehaviour
{
    [SerializeField]
    private bool trueImageIs;
    [SerializeField]
    private Sprite trueCellImage;

    public void CheckOnCorrectImage(Sprite cellImage)
    {
        if (trueCellImage == cellImage) trueImageIs = true;
        else trueImageIs = false;
    }
```

```
    public bool GetTrueImage()
    {
        return trueImageIs;
    }
}
```

Файл PazzleGame.cs:

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

public class PazzleGame : MonoBehaviour
{
    [SerializeField] private List<GameObject> _cells;
    [SerializeField] private List<Sprite> _cellsImages;
    [SerializeField] private Sprite _voidCell;

    private int voidCellID;

    private void Start()
    {
        voidCellID = 10;
    }

    private void CheckAllCells()
    {
        int count = 0;

        for (int i = 0; i < _cells.Count; i++)
        {
            Cell currentCell = _cells[i].transform.GetComponent<Cell>();
            if (currentCell.GetTrueImage()) count++;
        }

        if(count == 15)
        {
            Debug.Log("Game over");
            _cellsImages[voidCellID].sprite = _cellsImages[voidCellID ];
        }
    }

    private void CheckVoidCell(int cellID)
    {
        Cell currentVoidCell = _cells[voidCellID].transform.GetComponent<Cell>();

currentVoidCell.CheckOnCorrectImage(_cells[cellID].transform.GetComponent<Image>().sprite);
        MoveCell(cellID);
    }
```

```
    private void MoveCell(int cellID)
    {
        _cells[voidCellID].transform.GetComponent<Image>().sprite =
_cells[cellID].transform.GetComponent<Image>().sprite;
        _cells[cellID].transform.GetComponent<Image>().sprite = _voidCell;
        voidCellID = cellID;
    }

    public void ChooseCell(int cellID)
    {
        CheckVoidCell(cellID);
        CheckAllCells();
    }
}
using UnityEngine;

public class AudioManager : MonoBehaviour
{
    [SerializeField]
    private Sound[] sounds;

    private void Start()
    {
        foreach (Sound sound in sounds)
        {
            sound.source = gameObject.AddComponent<AudioSource>();
        }
    }

    private Sound FindSound(string nameOfSound)
    {
        return System.Array.Find(sounds, sound => sound.name == nameOfSound);
    }

    public void PlaySound(string nameOfSound)
    {
        Sound sound = FindSound(nameOfSound);
        if (sound != null)
        {
            sound.source.Play();
        }
    }

    public void StopAudio(string nameOfSound)
    {
        Sound sound = FindSound(nameOfSound);
        if (sound != null)
        {
            sound.source.Stop();
        }
    }
```

```
}

public void Save()
{
      _story.SaveCurrentStoryLine();
 }
```

Файл SubMenu.cs:

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

public class SubMenuPanel : MonoBehaviour
{
   [SerializeField]
   private Story _story;

   private ScreenLogic screenLogic;

   private void Start()
   {
      screenLogic = GameObject.Find("GameManager").GetComponent<ScreenLogic>();
   }

   public void ContinueGame()
   {
      _story.SaveCurrentStoryLine();
      screenLogic.CloseScreenOrWindow(9);
   }

   public void OpenSettings()
   {
      screenLogic.OpenScreenOrWindow(8);
   }

   public void ReturnToMenu()
   {
      _story.SaveCurrentStoryLine();
      SceneManager.LoadScene("MainMenu");
   }
}
```

ДОДАТОК Б

ВІДГУК
керівника економічного розділу
на кваліфікаційну роботу бакалавра
на тему:
"Розробка багаторівневої відеогри на основі кроссплатформного рушія Unity"
студентки групи 122-19-4 Пилипенко Кароліни Дмитрівни

ДОДАТОК В

ПЕРЕЛІК ДОКУМЕНТІВ НА ОПТИЧНОМУ НОСІЇ

Ім’я файлу	**Опис**
Пояснювальні документи	
ПЗ_Пилипенко.doc	Пояснювальна записка до кваліфікаційної роботи. Документ Word.
ПЗ_Пилипенко.pdf	Пояснювальна записка до кваліфікаційної роботи в форматі PDF
Програма	
HorsDeCombat.zip	Архів. Містить коди програми і откомпільовану програму
Презентація	
Презентація_Пилипенко.ppt	Презентація кваліфікаційної роботи