МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра програмної інженерії

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА

на тему: **«РОЗРОБКА ЛОГІЧНОЇ ВІДЕОГРИ-ГОЛОВОЛОМКИ ДЛЯ ШКОЛЯРІВ»**

Виконав: студент 2 курсу, групи 8.1212-іпз-1
спеціальності 121 інженерія програмного забезпечення
(шифр і назва спеціальності)
освітньої програми інженерія програмного забезпечення
(назва освітньої програми)
Я.Г. Зябров
(ініціали та прізвище)

Керівник доцент кафедри програмної інженерії, доцент, к.т.н. Мухін В.В.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Рецензент завідувач кафедри фундаментальної та прикладної математики, професор, д.т.н. Гребенюк С.М.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя – 2023

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра програмної інженерії

Рівень вищої освіти магістр

Спеціальність 121 інженерія програмного забезпечення
(шифр і назва)

Освітня програма інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри програмної інженерії, к.ф.-м.н., доцент

_____ Лісняк А.О.
(підпис)

“ ____ ” _____ 2023 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Зяброву Ярославу Георгійовичу
(прізвище, ім’я та по-батькові)

1. Тема роботи Розробка логічної відеогри-головоломки для школярів

керівник роботи Мухін Віталій Вікторович, к.т.н., доцент
(прізвище, ім’я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 01 » травня 2023 року № 642-с

2. Строк подання студентом роботи 27.11.2023 р.

3. Вихідні дані до роботи 1. Постановка задачі.
2. Перелік літератури.
3. Офіційна документація Unity.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Аналіз актуальності та проблематики розроблювальної гри.
2. Аналіз та вибір інструментів для реалізації обраної гри.
3. Проєктування гри.

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)
презентація

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання 03.05.2023 р.

К А Л Е Н Д А Р Н И Й П Л А Н

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Розробка плану роботи.	17.05.2023	
2.	Збір вихідних даних.	07.06.2023	
3.	Обробка методичних та теоретичних	28.06.2023	
	джерел.		
4.	Розробка першого та другого розділу.	30.08.2023	
5.	Розробка третього розділу.	08.11.2023	
6.	Оформлення та нормоконтроль	20.11.2023	
	кваліфікаційної роботи магістра.		
7.	Захист кваліфікаційної роботи.	14.12.2023	

Студент _____ Я.Г. Зябров
(підпис) (ініціали та прізвище)

Керівник роботи _____ В.В. Мухін
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

Нормоконтролер _____ А.В. Столярова
(підпис) (ініціали та прізвище)

РЕФЕРАТ

Кваліфікаційна робота магістра «Розробка логічної відеогри-головоломки для школярів»: 45 с., 36 рис., 1 табл., 14 джерел, 2 додатки.

ВІДЕОГРА, ГОЛОВОЛОМКА, ІГРОВА МЕХАНІКА, ІГРОВИЙ РУШІЙ, СЕРЕДОВИЩЕ РОЗРОБКИ, СЦЕНАРІЙ.

Об’єкт дослідження – геймплей, що базується на виконанні головоломок.

Мета роботи – збільшення зацікавленості потенційних гравців до ігор в жанрі головоломок що реалізується за рахунок геймплею, що базується на виконанні логічних завдань.

У рамках кваліфікаційної роботи було проведено аналіз ринку комп’ютерних ігор у жанрі головоломок. Було розглянуто переваги та недоліки різних програмних інструментів для розробки ігор. Також детально вивчено особливості розробки комп’ютерних ігор жанру головоломки.

Комп’ютерна гра була розроблена за допомогою ігрового рушія Unity, а програмування ігрової логіки проводилося з використанням мови програмування C#.

Ця гра може бути використана як швидкий спосіб ненадовго відволіктися, так і повноцінна гра, яка забезпечує поглиблене занурення у світ гри та відволікає від повсякденних проблем.

SUMMARY

Master’s qualifying paper «Development of a Logical Puzzle Video Game for Schoolchildren»: 45 pages, 36 figures, 1 table, 14 references, 2 supplements.

VIDEO GAME, PUZZLE, GAME ENGINE, GAME MECHANICS, DEVELOPMENT ENVIRONMENT, SCRIPT.

The object of the study is the gameplay based on solving puzzles.

The aim of the study is to increase the interest of potential players in puzzle games through the gameplay based on the execution of logical tasks.

As part of the diploma project, the market for computer games in the puzzle genre was analyzed. The advantages and disadvantages of various software tools for game development were considered. The peculiarities of developing computer games in the puzzle genre were also studied in detail.

The computer game was developed using the Unity game engine, and the game logic was programmed using the C# programming language.

This game can be used as a quick way to distract yourself for a while, as well as a full-fledged game that provides in-depth immersion in the game world and distracts from everyday problems.

ЗМІСТ

ВСТУП

В епоху стрімкого технологічного розвитку та змін у парадигмі навчання, питання створення інноваційних інструментів для підтримки освіти стає більш актуальним ніж будь-коли раніше. Сучасне покоління школярів потребує не тільки знань, але і навичок розвитку критичного мислення, творчості та проблемного вирішення завдань. У цьому контексті важливо створювати інтерактивні ігри, які стимулюють не лише ігровий процес, але й активізують мозкову діяльність учнів.

Актуальність розробки ігор-головоломок для школярів обумовлена сучасними освітніми та соціокультурними тенденціями, що визначають вимоги до формування компетентного та гнучкого покоління. Визнання ролі ігор у процесі навчання та розвитку стає ключовим компонентом педагогічної стратегії, оскільки ці інтерактивні інструменти сприяють не лише засвоєнню конкретних знань, але й активному розвитку розумових, соціальних та творчих навичок.

Однією з головних переваг використання ігор-головоломок в освіті є їхній потенціал стимулювати критичне мислення, логічний аналіз та творчість учнів. Гри, які вимагають розв’язання завдань, розширюють межі звичайного мислення та сприяють формуванню навичок самостійного вирішення проблем. Це особливо важливо в контексті сучасного освітнього підходу, що підкреслює важливість розвитку критичного та творчого потенціалу кожного учня.

Застосування ігор-головоломок в навчальному процесі також сприяє підвищенню мотивації учнів до навчання. Вони створюють атмосферу захоплення та інтриги, що спонукає дітей бути активними учасниками навчання, вдосконалюючи при цьому навички командної роботи, спілкування та вирішення конфліктів.

Об’єктом дослідження є геймплей, що базується на виконанні головоломок.

Предметом дослідження обрані технології розробки відеогри гри за допомогою ігрового рушія Unity.

Метою кваліфікаційної роботи є збільшення зацікавленості потенційних гравців до ігор в жанрі головоломок що реалізується за рахунок геймплею, що базується на виконанні логічних завдань.

Таким чином для досягнення мети в роботі поставлені наступні задачі:

- провести аналіз ринку комп’ютерних ігор у жанрі головоломок;
- розглянути переваги та недоліки різних програмних інструментів для розробки ігор;
- визначити особливості розробки комп’ютерних ігор жанру головоломки.

Наукова новизна кваліфікаційної роботи полягає у впровадженні і дослідженні інноваційних геймплейних механік, створенні унікального сценарію та використанні потужних функціональних можливостей Unity для реалізації цього проєкту.

Ця гра може бути використана як швидкий спосіб ненадовго відволіктися, так і повноцінна гра, яка забезпечує поглиблене занурення у світ гри та відволікає від повсякденних проблем.

1 ОГЛЯД АРХІТЕКТУРИ ІСНУЮЧИХ ІГОР В ЖАНРІ ГОЛОВОЛОМОК ТА АНАЛІЗ ПРОЄКТУ

1.1 Аналіз та загальна характеристика відеоігор та жанру головоломки

Відеогра – це електронна гра, в ігровому процесі якої гравець використовує інтерфейс користувача, щоб отримати зворотну інформацію з відеопристрою. Електронні пристрої, які використовуються для того щоб грати, називаються ігровими платформами. Наприклад, до таких платформ належать персональний комп’ютер та гральна консоль. Пристрій введення, який використовується для керування грою, називається ігровим контролером. Це може бути, наприклад, джойстик, клавіатура та мишка, геймпад або сенсорний екран [3].

Головоломка – назва жанру відеоігор, метою яких є вирішення логічних завдань, що вимагають від гравця задіяння логіки, стратегії, інтуїції та іноді ерудиції й уважності. Головоломки можуть включатися до ігор інших жанрів як ключові елементи ігрового процесу або ж для його урізноманітнення як міні-ігри [4].

Дослідження популярних і успішних ігор в жанрі головоломки є важливим кроком для розуміння основних характеристик, механік та елементів, які визначають цей жанр. Розглянемо деякі з них:

- основна ідея головоломок полягає в тому, щоб представити гравцеві завдання, які вимагають від нього розв’язання логічним шляхом (це може включати в себе розгадування головоломок, вирішення логічних задач або використання креативності для подолання перешкод);
- головоломки можуть включати в себе різноманітні типи завдань, такі як логічні головоломки, кросворди, головоломки на час, завдання на аналіз та розв’язання, ігри з об’єктами та багато іншого;
- деякі головоломки можуть базуватися на фізичних законах або механіці

гри, де гравець повинен використовувати ці закони для вирішення завдань;

– головоломки часто вимагають від гравця ретельного спостереження, уважності до деталей та здатності виявляти шаблони та зв’язки між об’єктами чи елементами гри;
– головоломки часто будуються на системі рівнів, де кожен рівень представляє нові виклики та складність (гравець повинен пройти кожен рівень, розв’язуючи головоломки, щоб просуватися вперед в грі);
– гравець може мати кілька шляхів для розв’язання конкретної головоломки, сприяючи розвитку креативності та вибору власного шляху до успіху;
– деякі головоломки можуть мати загадкову атмосферу, що підкреслюється загадковими сюжетами, арт-стилем чи звуковим супроводом.

Ці елементи допомагають створювати унікальний геймплей та виклики для гравців у світі відеоігор обраного жанру.

1.2 Аналіз вже існуючих ігор в жанрі головоломок

Жанр головоломок у відеоігровій індустрії вражає своєю різноманітністю та творчим підходом до викликів для гравців. Вже існуючі ігри у цьому жанрі не лише втілюють розвагу, але й стають платформою для розвитку логічного та креативного мислення. Нижче будуть розглянуті найвідоміші ігри даного жанру.

Відеогра «2048», написана 19-річним італійським розробником Габріелем Чіруллі мовою програмування JavaScript. Гра є числовою головоломкою з полем у формі квадрата 4х4 клітинки. Мета гри – складаючи плитки з числами, отримати плитку номіналу «2048», звідки й походить назва.

Гра починається на квадратному полі з 16-и клітинок, де дві клітинки зайняті плитками номіналом «2» і «4». Кожну плитку можна перемістити по

горизонталі чи вертикалі. Щоразу як гравець переміщує плитку, на полі з'являється додаткова плитка номіналу «2» (з ймовірністю 90 %) або «4» (з ймовірністю 10 %). Дві плитки одного номіналу, будучи поміщеними на одну клітинку, зливаються в одну, номінал якої дорівнює сумі злитих. Якщо в одній з ліній після руху поряд стоять понад дві плитки одного номіналу, то вони зливаються автоматично. Кожному номіналу відповідає свій колір, що більший номінал, то «гарячіший» колір [2].

За кожне злиття ігрові бали збільшуються на номінал новоутвореної плитки. Метою є отримати плитку «2048», після чого дозволяється продовжити. Гра закінчується, якщо після чергового переміщення неможливо виконати жодне злиття плиток (див. рис. 1.1).

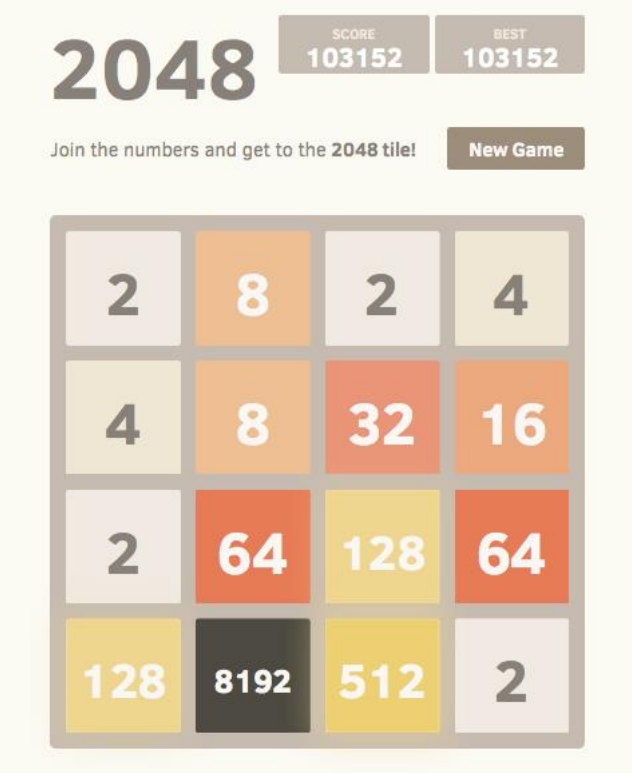


Рисунок 1.1 – Знімок екрану гри «2048»

Zuma – відеогра з родини казуальних головоломок, випущена 2003 року. Розробник – PopCap Games. Існує онлайн-версія гри Zuma та безліч її клонів, версія під назвою Zuma Deluxe для Windows і Mac (див. рис. 1.2).

Гра складається з серії рівнів, виконаних в ацтецькому стилі, де гравець

повинен завадити ланцюжку різнокольорових кульок досягнути лузи у вигляді черепа. Гравець керує фігуркою жаби, що може обертатися навколо своєї осі, та вистрілює кульки випадкового кольору. Індикатор на спині жаби сигналізує про колір наступної кульки. Низки кульок одного кольору (три й більше штук) вибухають, скорочуючи ланцюжок. Тому гравцеві слід вистрілювати кульки так, щоб вони формували подібні низки. Основне завдання кожного рівня – протриматися до заповнення спеціального індикатора (що власне і є Зума), що заповнюється при знищенні кульок. Після того, як він заповнений, нові кульки перестають з’являтися. Якщо ланцюжок потрапляє в лузу, гравець втрачає одне життя і починає рівень заново. Вже пройдені рівні є змога перегравати. Коли всі життя вичерпано, гра закінчується. Кількість кольорів, у які забарвлено кульки, поступово збільшується, ускладнюючи процес гри. За знищення кульок нараховуються бали, спонукаючи гравця повертатися до гри й побити колишній рекорд [6].

Рисунок 1.2 – Знімок екрану гри «Zuma»

Тетріс – відеогра-головоломка, в якій випадкові фігурки тетраміно падають зверху в прямокутний стакан шириною 10 і висотою 20 клітин. У польоті гравець може повертати фігурку та рухати її по горизонталі. Також можна «скидати» фігурку, тобто прискорювати її падіння, коли вже вирішено, куди фігурка

повинна впасти. Фігурка летить, поки не наткнеться на іншу фігурку або на дно склянки. Якщо при цьому заповнився горизонтальний ряд з 10 кліток, він пропадає і все, що вище нього, опускається на одну клітку (див. рис. 1.3).

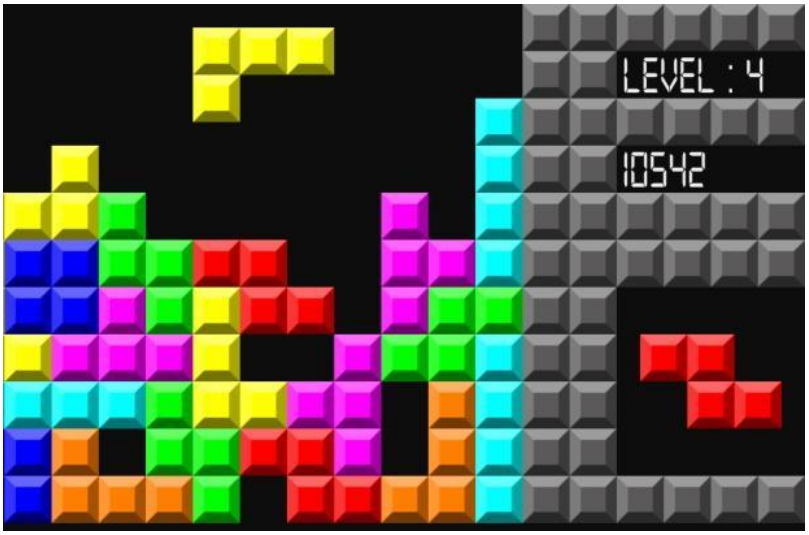


Рисунок 1.3 – Знімок екрану гри «Тетріс»

У спеціальному полі гравець бачить фігурку, яка буде слідувати після поточної – ця підказка дозволяє планувати свої дії. Темп гри поступово збільшується. Назва гри походить від кількості клітин, з яких складається кожна фігура. Гра закінчується, коли нова фігурка не може поміститися в стакан. Гравець отримує бали за кожну фігурку, тому його задача – заповнювати ряди, не заповнюючи саму склянку якомога довше, щоб таким чином отримати якомога більше балів [5].

Усе більше ігор жанру головоломок почали поєднувати елементи з іншими жанрами, такими як виживання, рольові ігри, екшн та інші. Це дозволило створювати більш різноманітні та цікаві ігрові досвіди для гравців.

1.3 Актуальність проєкту

Актуальність проєкту виявляється у декількох аспектах. Розглянемо деякі з них більш детально.

Галузь комп’ютерних ігор є однією з тих, що найшвидше розвивається і є привабливою для користувачів. Ігри є не тільки формою розваги, але й способом відпочинку, навчання та соціальної взаємодії. Високий попит на нові та захопливі ігрові проєкти створює потребу в постійному розробленні та вдосконаленні нових ігор.

Розвиток технологій у галузі комп’ютерної графіки, штучного інтелекту, фізичної симуляції та інших сфер інформаційної технології стимулює створення більш складних і реалістичних ігрових досвідів. Розробка проєкту з використанням популярного ігрового рушія Unity дозволяє використати сучасні технології та створити гру з високоякісною графікою та реалістичною фізикою.

Розробка гри з використанням Unity надає студентам можливість практично застосувати свої знання та навички в галузі розробки програмного забезпечення. Вивчення процесу розробки ігор дозволяє краще розуміти алгоритми, архітектуру програмного забезпечення та використання візуальних інструментів для створення графічного інтерфейсу та геймплею.

Розробка гри відкриває можливості для дослідження ігрових механік, взаємодії з гравцем, наративних структур, ефектів та інших аспектів геймдизайну. Це дозволяє вдосконалювати та вивчати нові підходи до створення ігрового досвіду, впливати на емоційну взаємодію гравця з грою та розробляти нові інноваційні ідеї.

Розробка ігор національними розробниками є важливим чинником розвитку української ігрової індустрії. Проєкт, що має локалізацію на українську мову, сприяє поширенню ігор на внутрішньому ринку та впливає на популяризацію української геймдев-спільноти як на внутрішньому, так і на зовнішньому рівнях.

Розробка власної гри дає можливість реалізувати креативні ідеї, втілити унікальний наратив, створити неповторну атмосферу та дослідити нові жанрові комбінації. В процесі розробки студент може розвинути свої навички у геймдизайні, візуальному мистецтві, звуковому дизайні та інших аспектах гри, що сприятиме його особистому і професійному зростанню.

Отже, актуальність проєкту виявляється у його відповідності сучасним тенденціям і потребам галузі ігрової розробки, можливостям дослідження та творчого розвитку.

1.4 Об’єкт, предмет та мета проєкту

Визначивши актуальність проєкту перейдемо до формулювання об’єкту та предмету дослідження. А також необхідним є визначення мети роботи.

Об’єкт дослідження – геймплей, що базується на виконанні головоломок.

Предмет дослідження – технології розробки відеогри гри за допомогою ігрового рушія Unity.

Мета роботи – збільшення зацікавленості потенційних гравців до ігор в жанрі головоломок що реалізується за рахунок геймплею, що базується на виконанні логічних завдань.

2 ПРОЄКТУВАННЯ ГРИ В ЖАНРІ ГОЛОВОЛОМКИ

В даному розділі буде проведений аналіз існуючих середовищ розробки обраного проєкту. Також на цьому етапі роботи буде досліджений загальний алгоритм реалізації проєкту, який допоможе забезпечити ефективний процес розробки.

2.1 Аналіз та обґрунтування вибору середовища розробки проєкту

У світі комп’ютерних ігор розробка власної гри є захоплюючою та стимулюючою справою. Для успішної реалізації проєкту необхідно обрати відповідне середовище розробки, яке забезпечить необхідні інструменти та можливості для створення професійної гри.

Одними з найпопулярніших та потужних середовищ є Unity, Unreal Engine та GameMaker Studio [8].

Наведемо огляд трьох програмних засобів, їх основні характеристики, переваги та недоліки.

2.1.1 Unity Game Engine

Unity Game Engine є одним з найпопулярніших і потужних програмних засобів для розробки ігор, включаючи жанр головоломки.

Програмне забезпечення Unity включає в себе інтегровану розробку середовища, візуальний редактор, засоби для моделювання 3D-об’єктів, налаштування фізики, розробки штучного інтелекту, створення анімацій, компіляції та розгортання гри на різних платформах (див. рис. 2.1).

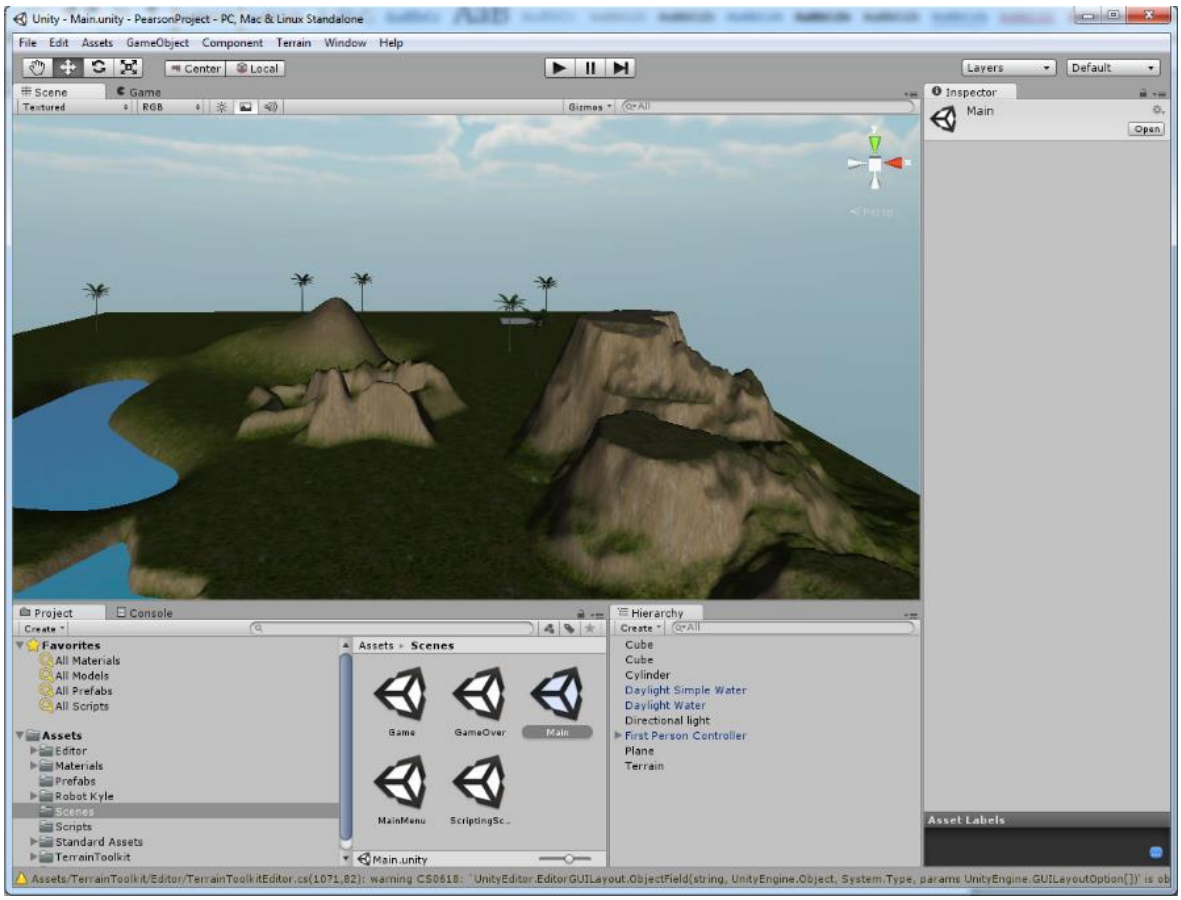


Рисунок 2.1 – Інтерфейс ПЗ «Unity»

Основні характеристики Unity включають:

– Unity надає широкі можливості для створення графічно багатих ігор з різноманітними ефектами і фізикою;
– ігри, розроблені в Unity, можуть бути легко портовані на різні платформи, такі як Windows, macOS, iOS, Android, Xbox, PlayStation та інші;
– Unity забезпечує доступ до багатьох інструментів та ресурсів, які допомагають розробникам створювати гри, включаючи готові компоненти та візуальний редактор [13].

Переваги Unity для розробки ігор-головоломок включають:

– Unity надає розробникам багато інструментів і ресурсів, які допомагають створювати комплексну гру, включаючи графіку, фізику, звук, штучний інтелект та інші аспекти гри;
– Unity має велику спільноту розробників, що означає наявність великої

кількості документації, підручників, форумів та інших ресурсів, які можуть допомогти розробникам у вирішенні проблем та отриманні підтримки.

Серед недоліків Unity можна виділити наступні:

– Unity може бути дещо складним для вивчення і розуміння для новачків, особливо якщо вони не мають попереднього досвіду в розробці ігор;
– Unity має безкоштовну версію для особистого використання, але для комерційної розробки можуть бути необхідні платні пакети або додаткові розширення;
– розробка гри в Unity може обмежуватись функціями та можливостями, які надаються самим Unity та його екосистемою (розробники можуть бути залежні від оновлень та підтримки Unity).

2.1.2 Unreal Engine

Unreal Engine є ще одним потужним програмним засобом для розробки ігор, який також може бути використаний для створення гри жанру головоломки. Основні характеристики Unreal Engine включають [14]:

– Unreal Engine забезпечує високоякісну графіку, фотореалістичні ефекти, фізику і освітлення, що дозволяє створювати деталізовані і вражаючі візуальні ефекти;
– Unreal Engine має вбудовану підтримку віртуальної реальності, що дозволяє створювати ігри, які підтримують VR-пристрої і забезпечують іммерсивний геймплей;
– Unreal Engine надає розробникам доступ до багатьох інструментів, бібліотек та ресурсів, які допомагають створювати складні механіки гри, включаючи штучний інтелект, фізику, звукові ефекти та багато іншого.

Переваги Unreal Engine для розробки ігор включають:

– Unreal Engine забезпечує високоякісну графіку і візуальні ефекти, що

дозволяє створювати реалістичне середовище для гри;

– Unreal Engine має активну спільноту розробників, що означає доступ до великої кількості ресурсів, документації, форумів та підтримки.

Недоліки Unreal Engine:

– Unreal Engine може вимагати потужного комп’ютера для розробки і тестування гри, особливо якщо використовуються високоякісні графічні ефекти;

– Unreal Engine може бути складним для вивчення та освоєння для розробників без попереднього досвіду в галузі розробки ігор або програмування.

Програмне забезпечення Unreal Engine включає в себе інтегроване середовище розробки, візуальний редактор, редактор матеріалів, інструменти для створення 3D-об’єктів, анімації, програмування на мові Blueprint або C++, а також компіляцію та розгортання гри на різних платформах (див. рис. 2.2).

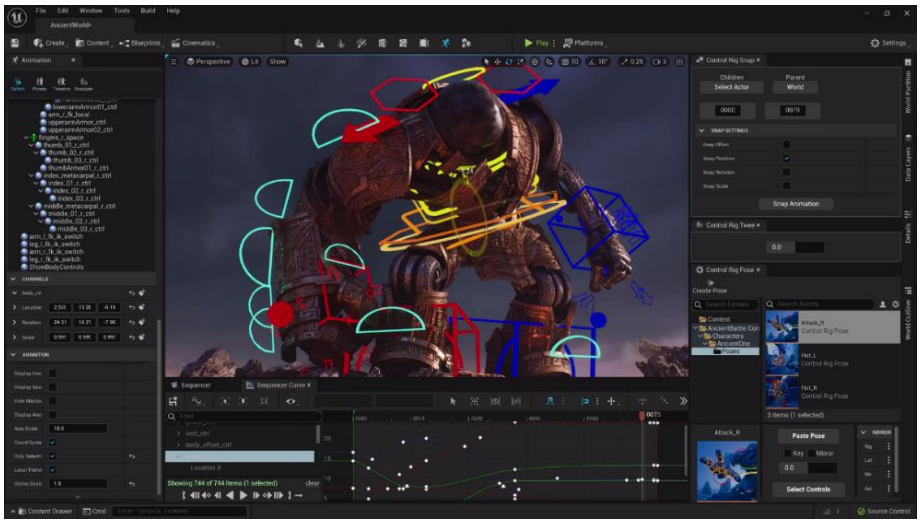


Рисунок 2.2 – Інтерфейс ПЗ Unreal Engine

2.1.3 GameMaker Studio

GameMaker Studio є програмним засобом для розробки 2D-ігор, який також може бути використаний для створення гри жанру головоломки [12]. Основні

характеристики GameMaker Studio включають наступні характеристики:

– GameMaker Studio має інтуїтивний візуальний редактор, що дозволяє створювати гру без необхідності програмування (він також надає мову програмування GML (GameMaker Language) для розширених можливостей);
– GameMaker Studio дозволяє розробникам використовувати власні скрипти та розширення, що дає більшу гнучкість у створенні унікальних механік та функцій гри;
– ігри, розроблені в GameMaker Studio, можуть бути експортовані на різні платформи, включаючи Windows, macOS, iOS, Android та інші.

Переваги GameMaker Studio:

– GameMaker Studio підходить для початківців і розробників без попереднього досвіду в розробці ігор;
– GameMaker Studio має велику кількість вбудованих функцій і засобів, що допомагають створювати базові механіки гри, такі як управління персонажем, ресурси, штучний інтелект і т.д;
– GameMaker Studio дозволяє швидко створювати прототипи гри, що дозволяє швидко перевірити концепцію та геймплей перед детальною розробкою.

Недоліки GameMaker Studio для розробки ігор включають:

– GameMaker Studio більш придатний для 2D-ігор, тому для створення складних 3D-моделей та ефектів можуть знадобитись додаткові інструменти або інтеграція з іншими програмами;
– хоча GameMaker Studio має мову програмування GML для розширеного програмування, вона може бути обмеженою порівняно з мовами програмування, такими як C# або C++.

Програмне забезпечення GameMaker Studio має вбудоване середовище розробки, візуальний редактор для створення рівнів, об’єктів, анімацій, інструменти для налаштування фізики, звуку, колізій та інші ресурси (див. рис. 2.3).

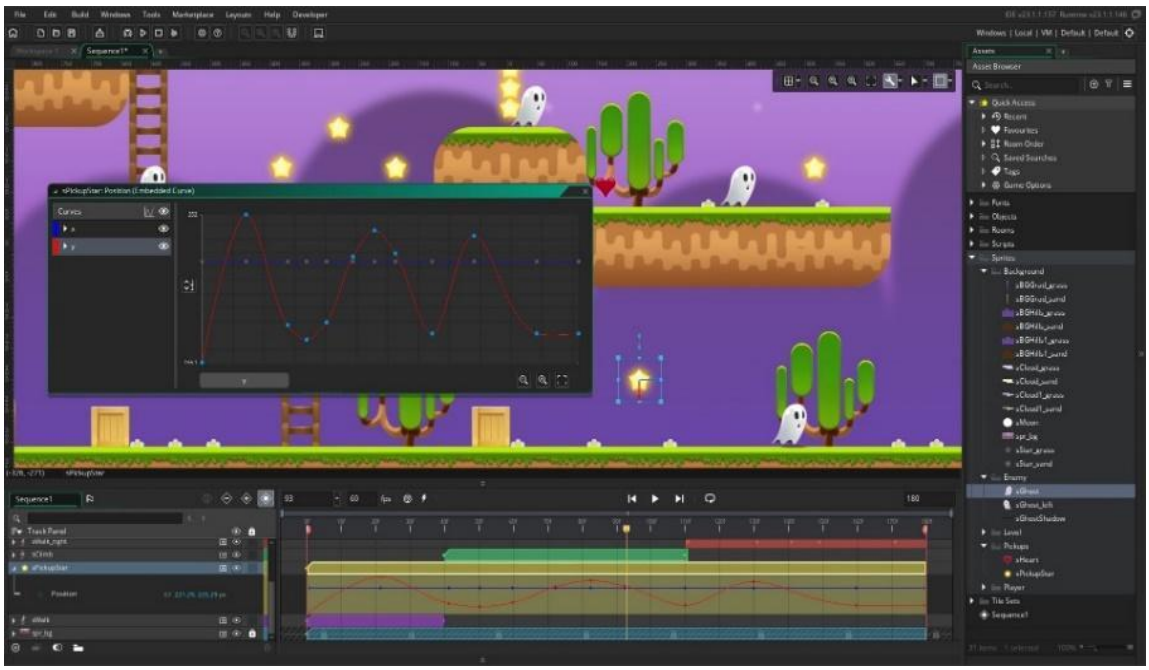


Рисунок 2.3 – Інтерфейс ПЗ GameMaker Studio

Порівняльна характеристика розглянутих додатків наведена у таблиці 2.1.

Таблиця 2.1 – Порівняльна таблиця програмних засобів для розробки гри у жанрі головоломки

Програмний засіб	**Основні характеристики**	**Переваги**	**Недоліки**
Unity Game Engine	Потужність, кросплатформенність, широкі можливості	Широкі можливості розробки, велика спільнота розробників	Висока складність для новачків, можливі проблеми з оптимізацією
Unreal Engine	Висока якість графіки, реалістичні фізичні ефекти	Висока якість графіки, розширені можливості розробки	Вимогливість до апаратного забезпечення, складність для новачків

Продовження табл. 2.1

Програмний засіб	**Основні характеристики**	**Переваги**	**Недоліки**
GameMaker Studio	Простота використання, швидкість розробки	Легкість вивчення, візуальне програмування	Обмежені можливості порівняно з іншими інструментами розробки

2.2 Загальний алгоритм реалізації проєкту

Загальний алгоритм реалізації проєкту може включати деякі кроки, розглянемо їх нижче.

Спочатку необхідно визначити концепцію та основні ідеї гри. Це включає вибір жанру, створення унікального сюжету, геймплею та інших ключових елементів.

Наступним кроком є розробка дизайну гри, включаючи візуальні ефекти, графіку, звукове супроводження та інтерфейс користувача. Це допомагає створити привабливе та зручне для гравців оточення.

Для реалізації гри потрібно визначити ігрові механіки, такі як керування гравцем, фізика об’єктів, штучний інтелект та інші геймплейні елементи;

Для забезпечення гри необхідно розробити рівні та завдання, які будуть виконувати гравці. Це включає створення локацій, завдань та інших елементів, які роблять гру цікавою та викликають захоплення.

Після реалізації гри важливо провести тестування для виявлення помилок, недоліків та вдосконалення геймплею. Налагодження гри допоможе покращити її стабільність та оптимізувати роботу.

Документацією програмного забезпечення вважається супроводжуючі документи з описом деталей необхідних для використання продукту. До документації відносять текст програми, опис програми, тощо.

Останній крок полягає в оптимізації гри, щоб забезпечити її оптимальну продуктивність та сумісність з різними платформами. Важливо виявити та усунути можливі проблеми з продуктивністю, завантаженням ресурсів та іншими технічними аспектами гри. Після завершення оптимізації гра готова до релізу, тобто до випуску на ринок [10].

Загальний алгоритм реалізації проєкту включає кроки від концептуалізації та дизайну до програмування, тестування та оптимізації. Цей підхід допомагає забезпечити ефективний процес розробки та створити якісну та захоплюючу гру для гравців.

2.3 Функціонал продукту

Розроблений продукт являє собою відеогру-головоломку для школярів. Основна мета самої гри – забезпечити гравцю поглиблене занурення у світ гри та відволікти від повсякденних проблем.

Отже основні вимоги до продукту є такими:

- різноманітність ігрових механік задля покращення реіграбельності;
- використання рандомної генерації ресурсів у грі з метою зміни досвіду гравця при кожній новій спробі;
- керування, що дозволяє грати в гру однією рукою;
- відображення всіх візуальних та текстових частин гри.

В першу чергу реалізуются саме ці функції продукту, так як є основними реалізаціями критеріїв створення гри в жанрі головоломки.

2.4 Підготовка матеріалів

Отже розробка гри в жанрі головоломки починається з ідеї сюжету та його написання. Важливу роль відіграє його складність та варіативність. Було обрано реалізовувати лінійний сюжет. Сюжет гри зручний та зрозумілий користувачу: є дві навчальні кімнати (перший клас для вивчення географії, другий для вивчення хімії). В кабінеті хімії учням пропонується здійснити хімічну реакцію (див. рис. 2.4). Вона полягає в тому, щоб змішати між собою запропоновані хімічні елементи та дослідити результат експерименту (див. рис. 2.5).

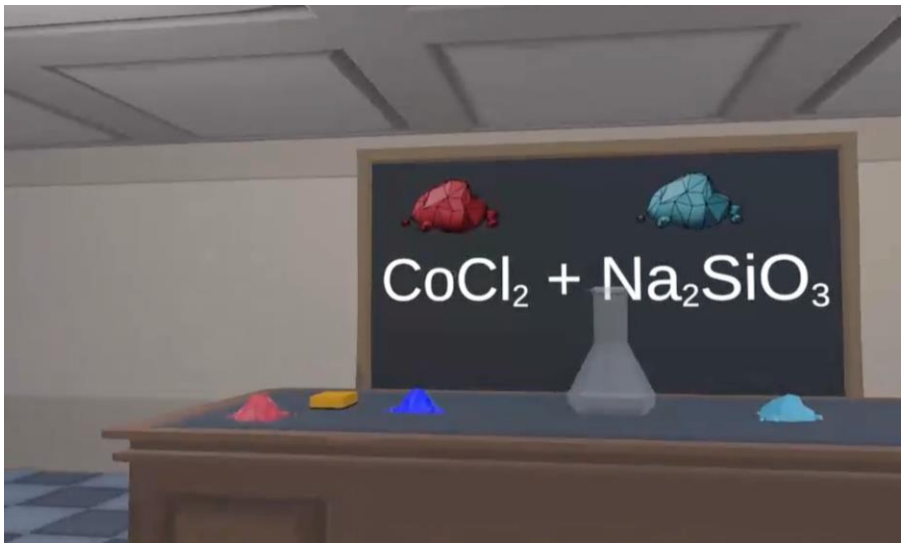


Рисунок 2.4 – Кабінет хімії

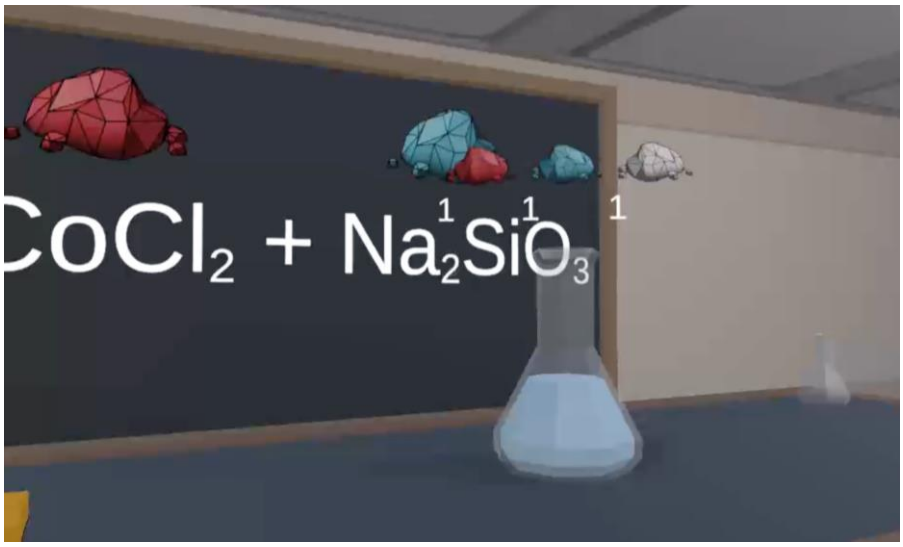


Рисунок 2.5 – Проведення експерименту

Також було дуже важливо реалізувати анімацію того, як гравець обирає хімічний елемент (див. рис. 2.6) та «розбиває» колбу, якщо суміш не вийшла вдалою (див. рис. 2.7).

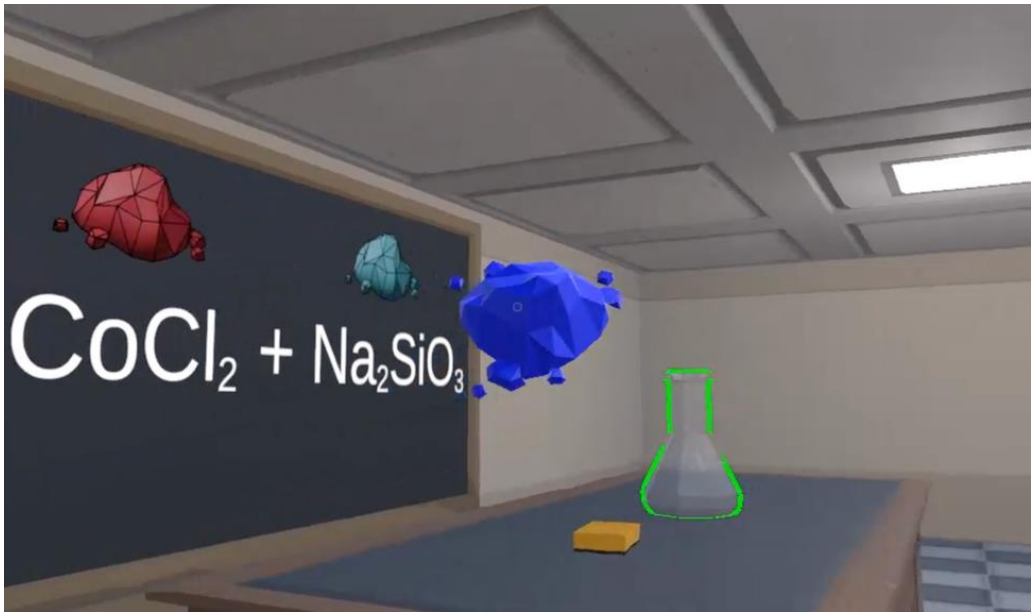


Рисунок 2.6 – Анімація перенесення складових до колби

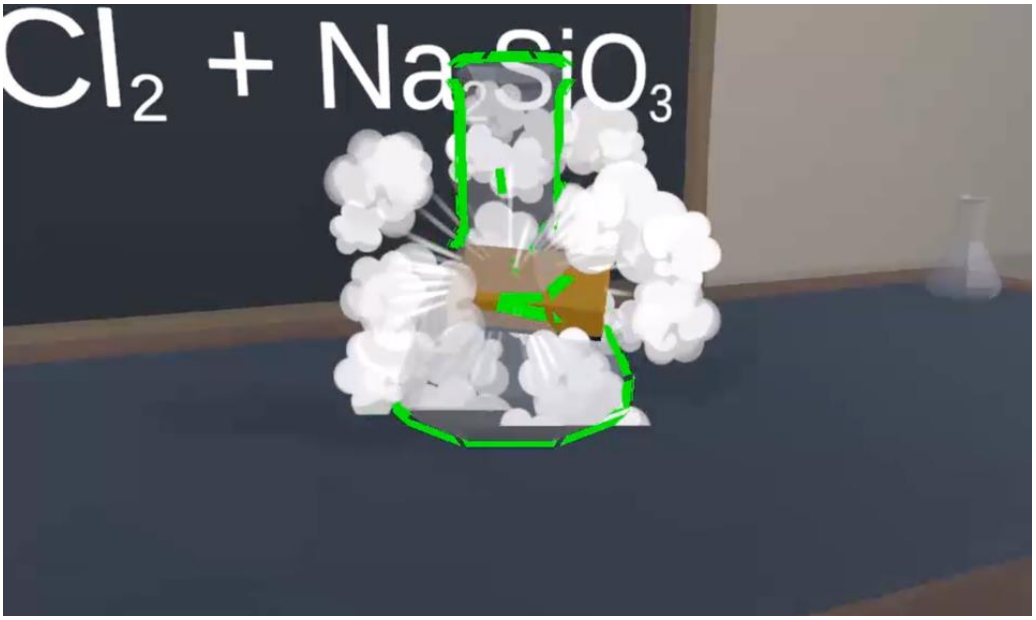


Рисунок 2.7 – Реалізація анімації «розбиття» колби

Про результат виконання експерименту учень зможе дізнатись із відповідних повідомлень (див. рис. 2.8, 2.9).

Рисунок 2.8 – Повідомлення про провалений рівень

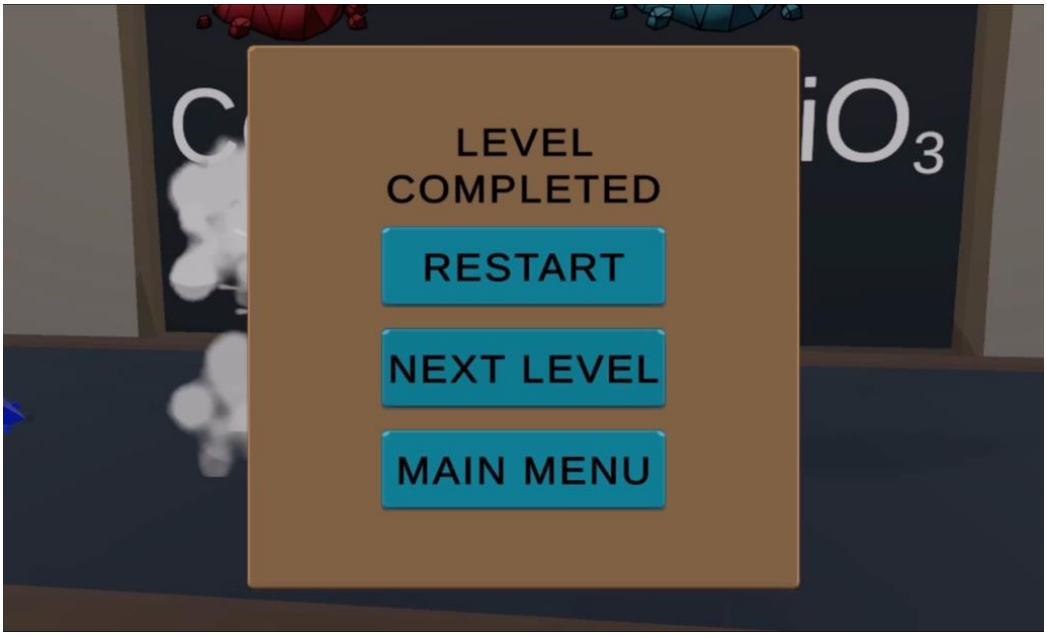


Рисунок 2.9 – Повідомлення про пройдений рівень

В кабінеті географії (див. рис. 2.10) учню необхідно прикріпити на карті зображення визначної пам’ятки відповідно до регіону України.

Тут також реалізована анімація перенесення зображення на карту (див. рис. 2.11, 2.12), а також повідомлення щодо успішності виконання поставленого завдання.

Рисунок 2.10 – Кабінет географії

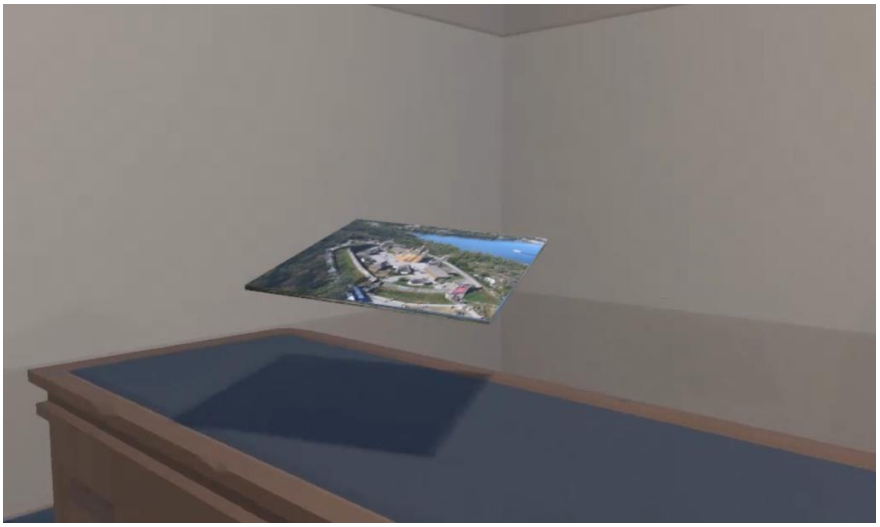


Рисунок 2.11 – Анімація перенесення зображення

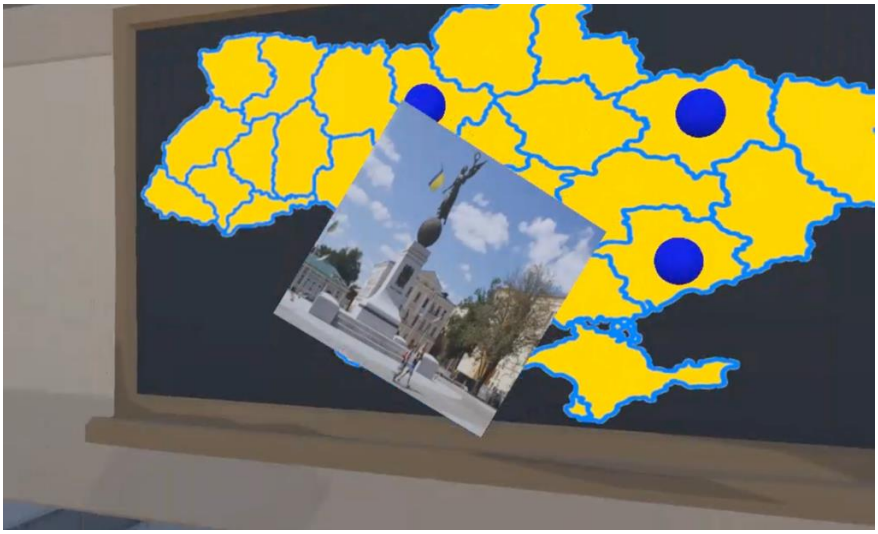


Рисунок 2.12 – Анімація перенесення зображення пам’ятки на карту України

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ГРИ В ЖАНРІ ГОЛОВОЛОМКИ

3.1 Підготовка робочого простору

Підготовка робочого простору є важливим кроком перед початком розробки проєкту. Цей етап включає наступні дії:

– встановлення необхідного програмного забезпечення: для розробки гри з використанням ігрового рушія Unity необхідно було встановити програмне забезпечення Unity;
– налаштування робочого середовища;
– налаштування версійного контролю: для ефективної роботи з розробкою проєкту використовувалась система версійного контролю Git (див. рис. 3.1) – це дозволило зберігати та керувати версіями коду і ресурсів.

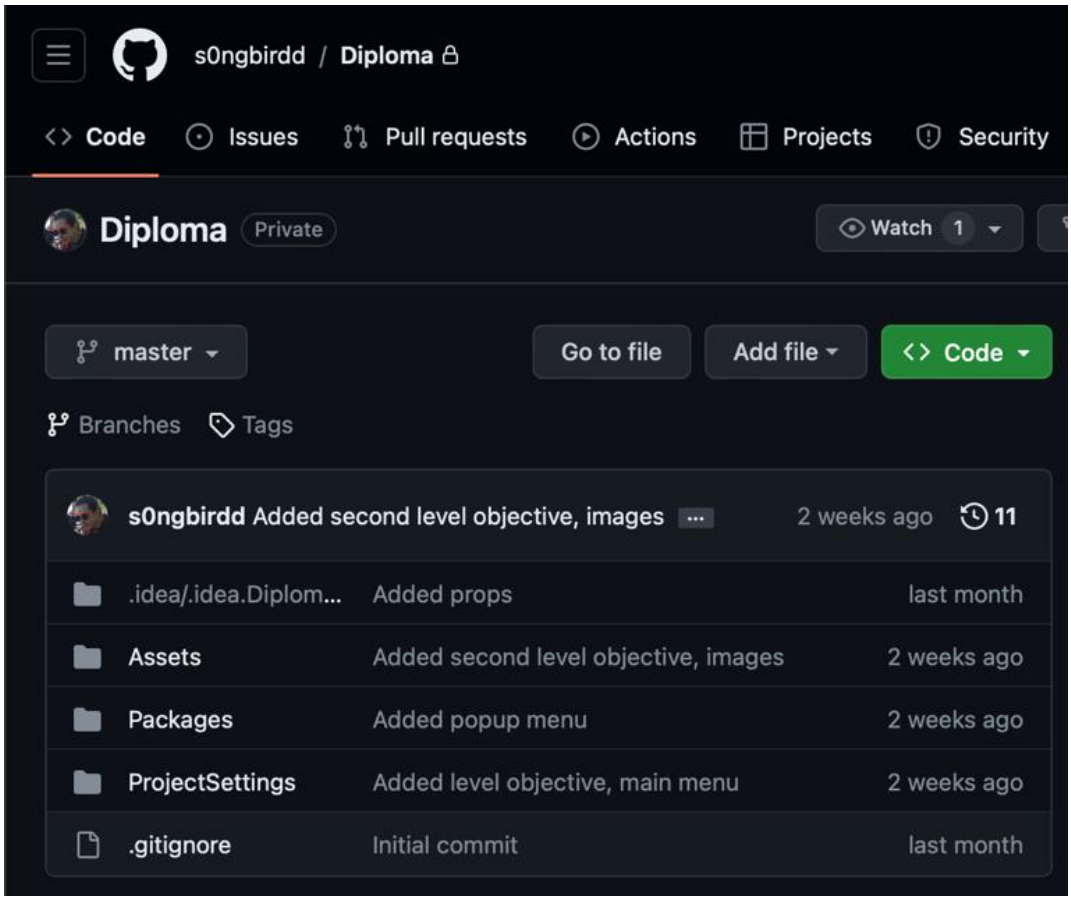


Рисунок 3.1 – Керування версіями коду за допомогою GitHub

Вікно проєкту (див. рис. 3.2) є одним з найважливіших інструментів Unity, адже саме тут розробник має можливість редагувати файли програми.

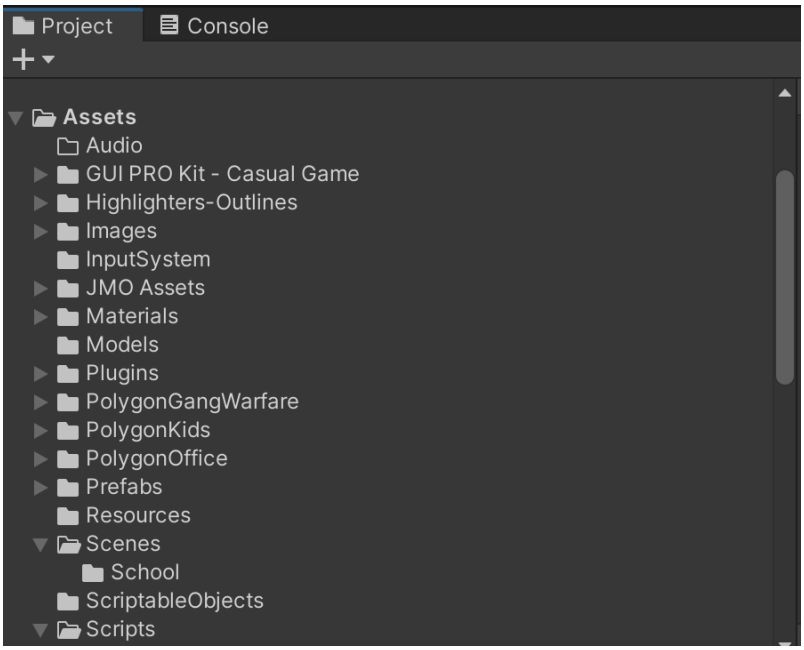


Рисунок 3.2 – Вікно проєкту

Інспектор – це один з основних інструментів, які надає ігровий рушій Unity для огляду та редагування компонентів об’єктів в грі. Це вікно знаходиться в інтегрованій середовищі розробки Unity та відображає список компонентів, прикріплених до вибраного об’єкта [9].

Інспектор надає розробникам можливість переглядати та змінювати властивості компонентів, такі як позиція, розмір, кольори, параметри фізичної взаємодії, анімації та інші. Він також відображає інформацію про скрипти, прикріплені до об’єктів, і дозволяє редагувати їх код безпосередньо в середовищі розробки.

Інспектор є потужним інструментом для швидкого налагодження і тестування гри, оскільки дозволяє розробникам змінювати параметри компонентів в реальному часі та спостерігати за їхніми змінами під час виконання гри. Він полегшує взаємодію з компонентами та дозволяє швидко здійснювати зміни для досягнення бажаних результатів у розробці гри (див. рис. 3.3).

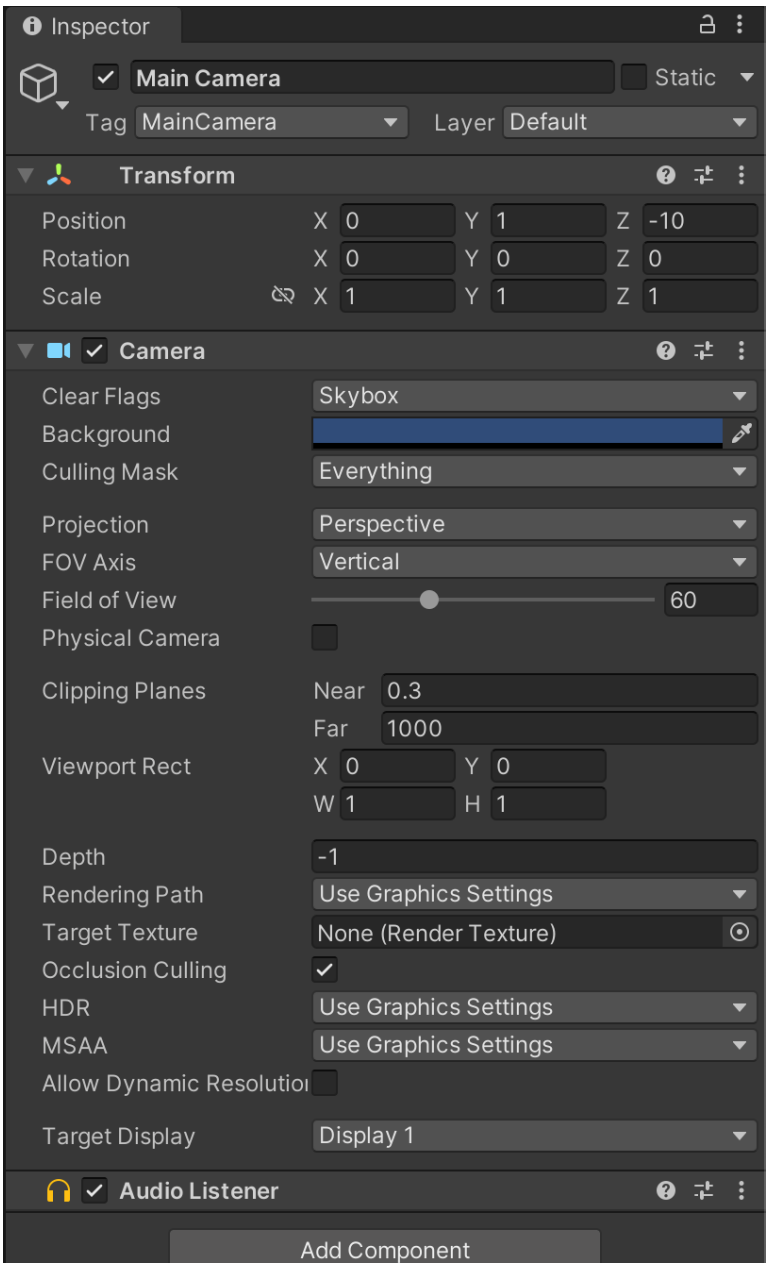


Рисунок 3.3 – Інспектор для огляду компонентів

При проєктуванні різноманітних додатків, а особливо комп’ютерних ігор, розробнику важливо контролювати процес виявлення будь яких помилок. Для цього слугує вікно консолі [12].

У вікні консолі можна побачити повідомлення про роботу скриптів, помилки в коді, інформацію про завантаження ресурсів, статуси підключених модулів та іншу корисну інформацію. Консоль також може відображати результати виконання спеціальних функцій або команд, які виконуються в процесі розробки (див. рис. 3.4).

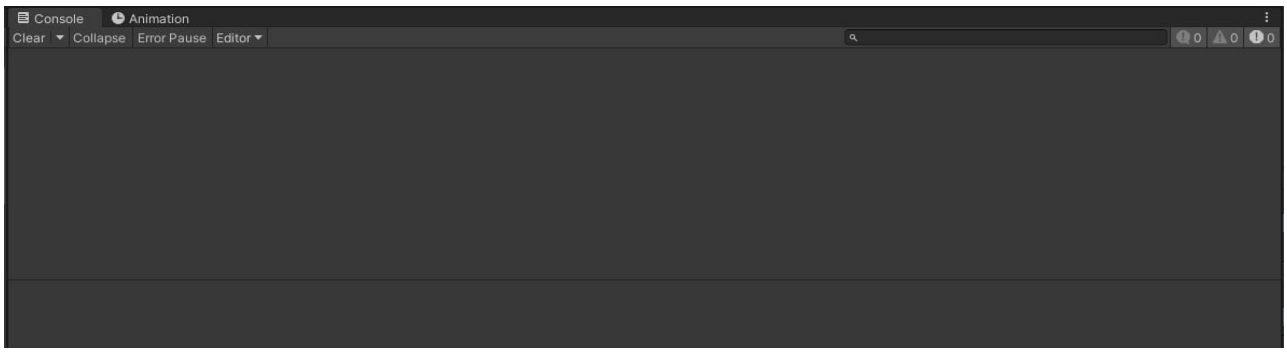


Рисунок 3.4 – Вікно консолі

3.2 Аналіз використаних для розробки проєкту інструментів Unity

Unity – це потужний ігровий рушій, який використовується для розробки комп'ютерних ігор та інтерактивних додатків [1]. Його широко використовують у галузі геймдеву, оскільки він надає розробникам багато можливостей і інструментів для створення гри з нуля.

Основні інструменти, що використовувались для розробки проєкту: редактор сцен; компонентна система; скриптування; графічний редактор; фізична симуляція; анімація; тестування та налагодження.

Ці інструменти Unity допомогли ефективно створювати, тестувати та розгортати комп’ютерні ігри з використанням рушія Unity в рамках даного проєкту. Вони забезпечують широкий спектр функціональності і дозволяють реалізувати різноманітні ідеї та концепції гри.

3.3 Написання коду

Під час розробки гри написання коду відіграє важливу роль у реалізації функціональності та поведінки гри.

Код відповідає за обробку подій, управління персонажем, взаємодію з об’єктами, реалізацію логіки гри та багато іншого. Використання мови

програмування C# у поєднанні з Unity API дозволило ефективно створити та керувати різноманітними аспектами гри [7].

Написання коду є ключовим етапом у розробці гри, оскільки він дозволяє перетворити концепцію гри на функціонуючий продукт з багатою функціональністю та захоплюючим геймплеєм.

Розглянемо більш детально основні елементи коду.

За хімічний експеримент із змішуванням хімічних елементів відповідає клас OrderController (див. рис. 3.5).

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

public class OrderController : MonoBehaviour
{
    [SerializeField] private List<DishType> _dishTypes;
    [SerializeField] private TextMeshProUGUI _orderText;
    [SerializeField] private float _timeBeforeSetOrderText = 1.5f;

    private DishType _orderDish;

    private void Start()
    {
        GenerateRandomOrderDish();
        SetOrderText();
    }

    private void OnTriggerEnter(Collider other)
    {
        if (other.TryGetComponent(out Dish dish))
        {
            if (_orderDish == dish.DishType)
            {
                other.gameObject.SetActive(false);
                _orderText.text = "Correct!";
                Invoke(nameof(SetOrderText), _timeBeforeSetOrderText);
                GenerateRandomOrderDish();
            }
            else
            {
                _orderText.text = "Wrong!";
                Invoke(nameof(SetOrderText), _timeBeforeSetOrderText);
            }
        }
    }

    private void GenerateRandomOrderDish()
    {
        _orderDish = (DishType)Random.Range(0, _dishTypes.Count);
    }

    private void SetOrderText()
    {
        _orderText.text = _orderDish.ToString();
    }
```

Рисунок 3.5 – Реалізація класу OrderController

За всі системні елементи в меню (вихід з гри, початок гри і т.д.) відповідає клас MainMenuController (див. рис. 3.6).

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

public class MainMenuController : MonoBehaviour
{
    [SerializeField] private Button _playButton;
    [SerializeField] private Button _quitButton;

    private void OnEnable()
    {
        _playButton.onClick.AddListener(LoadGame);
        _quitButton.onClick.AddListener(QuitGame);
    }

    private void OnDisable()
    {
        _playButton.onClick.RemoveListener(LoadGame);
        _quitButton.onClick.RemoveListener(QuitGame);
    }

    private void LoadGame()
    {
        SceneManager.LoadScene("School");
    }

    private void QuitGame()
    {
        Application.Quit();
    }
}
```

Рисунок 3.6 – Реалізація класу MainMenuController

За переміщення гравця по сцені відповідає клас BasicRigidBodyPush (див. рис. 3.7, 3.8).

```
using UnityEngine;

public class BasicRigidBodyPush : MonoBehaviour
{
        public LayerMask pushLayers;
        public bool canPush;
        [Range(0.5f, 5f)] public float strength = 1.1f;

        private void OnControllerColliderHit(ControllerColliderHit hit)
        {
                if (canPush) PushRigidBodies(hit);
        }

        private void PushRigidBodies(ControllerColliderHit hit)
        {
```

Рисунок 3.7 – Реалізація класу BasicRigidBodyPush

```
            // https://docs.unity3d.com/ScriptReference/CharacterController.OnControllerColliderHit.html

            // make sure we hit a non kinematic rigidbody
            Rigidbody body = hit.collider.attachedRigidbody;
            if (body == null || body.isKinematic) return;

            // make sure we only push desired layer(s)
            var bodyLayerMask = 1 << body.gameObject.layer;
            if ((bodyLayerMask & pushLayers.value) == 0) return;

            // We dont want to push objects below us
            if (hit.moveDirection.y < -0.3f) return;

            // Calculate push direction from move direction, horizontal motion only
            Vector3 pushDir = new Vector3(hit.moveDirection.x, 0.0f, hit.moveDirection.z);

            // Apply the push and take strength into account
            body.AddForce(pushDir * strength, ForceMode.Impulse);
        }
    }
```

Рисунок 3.8 – Реалізація класу BasicRigidBodyPush

За переміщення об’єктів гравцем відповідає клас DragObject (див. рис. 3.9).

```
using UnityEngine;

public class DragObject : GrabObject
{
    public override void PickObject(Transform pickupPointTransform)
    {
        //
        //_pickupObjectRigidbody.constraints = RigidbodyConstraints.FreezePositionY;
    }

    public override void DropObject()
    {
        /*_pickupObjectRigidbody.useGravity = true;
        _pickupObjectRigidbody.drag = 1;
        _pickupObjectRigidbody.constraints = RigidbodyConstraints.None;

        transform.parent = null;*/
        //_pickupObjectRigidbody.constraints = RigidbodyConstraints.None;
        transform.parent = null;
    }

    public override void MoveObject(Transform pickupPointTransform)
    {
        if (Vector3.Distance(transform.position, pickupPointTransform.position) > 0.1f)
        {
            Vector3 moveDirection = (pickupPointTransform.position - transform.position);
            _pickupObjectRigidbody.AddForce(moveDirection * _pickupForce);

            if (_pickupObjectRigidbody.velocity.magnitude > 1f)
            {
                _pickupObjectRigidbody.velocity = _pickupObjectRigidbody.velocity.normalized * 1f;
            }
        }
    }

    public override void ThrowObject(Transform pickupPointTransform)
    {
        transform.parent = null;
        //_pickupObjectRigidbody.constraints = RigidbodyConstraints.None;
    }

    public override void RotateObject()
    {
        //
    }
}
```

Рисунок 3.9 – Реалізація класу DragObject

3.4 Створення графічних елементів

Також для реалізації повноцінного додатку створеної гри необхідно реалізувати головне меню, та UI елементи керування грою під час її проходження.

Ця сцена буде зустрічати гравця при запуску гри. Отже воно має задачу зрозуміло піднести інформацію того, як почати гру або вийти з додатка (див. рис. 3.10).

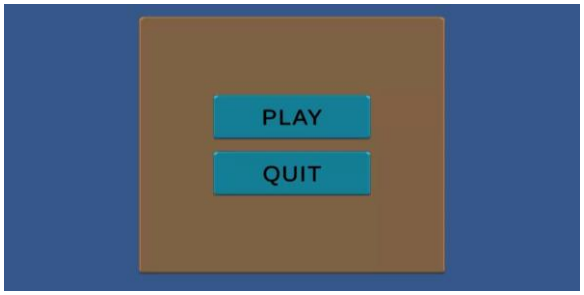


Рисунок 3.10 – Головне меню гри

Отже, це і буде реалізовано в сцені головного меню. Реалізація головного меню являє собою створення кнопок та написання коду, за що ці кнопки відповідають. Отже, необхідно створити окремий файл з кодом, де буде прописані дві задачі: відкриття сцени з самою грою для кнопки «Play», закриття всього додатку для кнопки «Quit».

Також було реалізовано меню паузи (див. рис. 3.11), спливаюче меню в разі провалу рівня (див. рис. 3.12), а також в разі успішного виконання завдання (див. рис. 3.13).

Рисунок 3.11 – Меню паузи

Рисунок 3.12 – Меню в разі провалу рівня

Рисунок 3.13 – Меню в разі проходження рівня

Завдяки графічним елементам меню, гравець має змогу ставити гру на паузу, перепроходити рівні, переходити на наступні рівні або вийти в головне меню.

ВИСНОВКИ

Результатом виконаної кваліфікаційної роботи магістра є комп’ютерна гра в жанрі головоломки для школярів. У роботі описується покрокове створення програмного продукту цього типу починаючи з аналізу поставленої задачі та завершуючи створенням гри.

Проаналізовано ринок комп’ютерних ігор і встановлено основні характеристики майбутнього продукту за вимогами жанру. Для проєктування власної гри було проведено аналіз існуючих ігор в даному жанрі та виділено необхідні риси майбутнього продукту. Розглянуто його актуальність.

Розглянуто основні ресурси для розробки комп’ютерної гри в обраному жанрі. Проаналізовано та обрано для реалізації такий інструмент як ігровий рушій Unity.

Розглянутий алгоритм створення комп’ютерних ігор в жанрі головоломки.

За результатами спроєктовано та розроблено відеогру-головоломку для школярів на основі аналізу потреб користувачів.

Додано суміжні механіки для покращення user experience, такі як система збереження, налаштування UI, покращення управління.

Виконано внутрішнє тестування гри та підготовка до публікації.

Отже результатом проведеної роботи є прототип гри в жанрі головоломки для школярів, що має всі ознаки свого жанру, головну мету, умови проходження і зручний інтерфейс для користування та відповідає поставленим вимогам. Прототип може бути довершений іншими деталями та функціями і офіційно виданий.

ПЕРЕЛІК ПОСИЛАНЬ

1. Крейтон Р. Х. Основи розробки ігор у Unity. Packt Publishing, 2010. 83 с.
2. Учасники проектів Вікімедіа. 2048 (гра) – Вікіпедія. URL: https://uk.wikipedia.org/wiki/2048_(гра) (дата звернення: 10.08.2023).
3. Учасники проектів Вікімедіа. Відеогра – Вікіпедія. URL: https://uk.wikipedia.org/wiki/Відеогра (дата звернення: 24.07.2023).
4. Учасники проектів Вікімедіа. Головоломка (жанр відеоігор) – Вікіпедія. URL: https://uk.wikipedia.org/wiki/Головоломка_(жанр_відеоігор) (дата звернення: 13.09.2023).
5. Учасники проектів Вікімедіа. Тетріс – Вікіпедія. URL: https://uk.wikipedia.org/wiki/Тетріс (дата звернення: 09.08.2023).
6. Учасники проектів Вікімедіа. Zuma (відеогра) – Вікіпедія. URL: https://uk.wikipedia.org/wiki/Zuma_(відеогра) (дата звернення: 08.10.2023).
7. Хокінг Д. М. Unity в дії. Мультиплатформенна розробка на практиці. 2016. 336 с.
8. David B. Hands-On Game Development Patterns with Unity. 2019. 116 p.
9. Ferrone H. Learning C# by Developing Games with Unity 2019: Code in C# and build 3D games with Unity, 4th ed. 2019. 342 p.
10. Goldstone W. Unity Game Development Essentials. Birmingham : Packt Publishing Ltd, 2009. 316 p.
11. Halpern J. Developing 2D Games with Unity: Independent Game Programming with C#. 2018. 408 p.
12. Make 2D Games With GameMaker: Free Video Game Maker. URL: https://gamemaker.io/en (date of access: 08.10.2023).
13. Unity Real-Time Development Platform: 3D, 2D, VR & AR Engine. Unity Real-Time Development Platform: 3D, 2D, VR & AR Engine. URL: https://unity.com (date of access: 04.08.2023).
14. Unreal Engine: The most powerful real-time 3D creation tool. Unreal Engine. URL: https://www.unrealengine.com/en-US (date of access: 08.07.2023).

ДОДАТОК А

GrabController

GrabController відповідає за переміщення об’єктів. На рисунках А.1 – А.4 представлено відповідний код.

```
using System;
using Highlighters;
using UnityEngine;

3 asset usages  8 usages  Yaroslav Ziabrov
public class GrabController : MonoBehaviour
{
    Yaroslav Ziabrov
    public static event Action<string> OnObjectFocus;
    Yaroslav Ziabrov
    public static event Action OnObjectUnfocus;
    Yaroslav Ziabrov
    public static event Action OnPickupObject;
    Yaroslav Ziabrov
    public static event Action OnDropObject;

    [Header("Pickup Parameters")]
    [SerializeField] private Transform _cameraTransform;  Changed in 3 assets
    [SerializeField] private Transform _pickupPointTransform;  Changed in 3 assets
    [SerializeField] private LayerMask _pickupLayerMask;  Serializable
    [SerializeField] private float _pickupDistance = 4f;  2

    private GrabObject _pickupObject;
    private Highlighter _highlighter;
    private Highlighter _highlighterTemp;

    private Color _baseImageColor = Color.red;
    private Color _interactionImageColor = Color.blue;
    private Color _moveImageColor = Color.green;
    private Color _currentImageColor;

    //private RaycastHit _hitPickup;

    Event function  Yaroslav Ziabrov
    private void OnEnable()
    {
        TriggerZoneObject.OnObjectEnterTrigger += DropObject;
        TriggerZoneLever.OnLeverEnterTrigger += DropObject;
        PickupObject.OnObjectCollisionEnter += DropObject;
        CraftController.OnAddCraftingIngredient += DropObject;
```

Рисунок А.1

```
}

Event function  Yaroslav Ziabrov
private void OnDisable()
{
    TriggerZoneObject.OnObjectEnterTrigger -= DropObject;
    TriggerZoneLever.OnLeverEnterTrigger -= DropObject;
    PickupObject.OnObjectCollisionEnter -= DropObject;
    CraftController.OnAddCraftingIngredient -= DropObject;
}

Event function  Yaroslav Ziabrov
private void Start()
{
    _currentImageColor = _baseImageColor;
}

Event function  Yaroslav Ziabrov
private void Update()
{
    if (Physics.Raycast( origin: _cameraTransform.position,  direction: _cameraTransform.forward, out RaycastHit hitPickup, _pickupDistance,  (int) _pickupLayerMask))
    {
        if (!_currentImageColor.Equals(_moveImageColor))
        {
            _currentImageColor = _interactionImageColor;

            if (hitPickup.transform.TryGetComponent(out _highlighter))
            {
                if (_highlighterTemp != null)
                {
                    _highlighterTemp.enabled = false;
                }

                _highlighterTemp = _highlighter;

                _highlighter.enabled = true;

                OnObjectFocus?.Invoke(_highlighter.gameObject.name);
            }
```

Рисунок А.2

```
            }
            else
            {
                if (_highlighter != null)
                {
                    _highlighter.enabled = false;
                }

                OnObjectUnfocus?.Invoke();
            }
        }
        else if (!_currentImageColor.Equals(_moveImageColor))
        {
            _currentImageColor = _baseImageColor;

            if (_highlighter != null)
            {
                _highlighter.enabled = false;

                OnObjectUnfocus?.Invoke();
            }
        }

        if (_pickupObject == null && Input.GetMouseButtonDown(0))
```

Рисунок А.3

```
if (_pickupObject == null && Input.GetMouseButtonDown(0))
{
    try
    {
        if (hitPickup.transform.TryGetComponent(out _pickupObject))
        {
            _pickupObject.PickObject(_pickupPointTransform);

            Debug.Log( message: "Pickup");
            OnPickupObject?.Invoke();

            _currentImageColor = _moveImageColor;
        }
    }
    catch (Exception e)
    {
        Debug.Log(e.Message);
    }
}
else if (_pickupObject != null && Input.GetMouseButtonDown(0))
{
    _pickupObject.DropObject();

    Debug.Log( message: "Drop");
    OnDropObject?.Invoke();

    _pickupObject = null;

    _currentImageColor = _baseImageColor;
}
else if (_pickupObject != null && Input.GetMouseButton(1))
{
    _pickupObject.ThrowObject(_pickupPointTransform);

    Debug.Log( message: "Drop");
    OnDropObject?.Invoke();

    _pickupObject = null;

    _currentImageColor = _baseImageColor;
}
else if (_pickupObject != null && Input.GetMouseButton(2))
{
    _pickupObject.RotateObject();
}
```

Рисунок А.4

ДОДАТОК Б

FirstPersonController

FirstPersonController реалізує переміщення і поворот камери. На рисунках Б.1 – Б.4 представлено відповідний код.

```
using System;
using UnityEngine;
#if ENABLE_INPUT_SYSTEM
using UnityEngine.InputSystem;
#endif

namespace StarterAssets
{
    [RequireComponent(typeof(CharacterController))]
#if ENABLE_INPUT_SYSTEM
    [RequireComponent(typeof(PlayerInput))]
#endif
    1 asset usage  Yaroslav Ziabrov
    public class FirstPersonController : MonoBehaviour
    {
        [Header("Player")]
        [Tooltip("Move speed of the character in m/s")]
        public float MoveSpeed = 4.0f;  Unchanged
        [Tooltip("Sprint speed of the character in m/s")]
        public float SprintSpeed = 6.0f;  Unchanged
        [Tooltip("Rotation speed of the character")]
        public float RotationSpeed = 1.0f;  Unchanged
        [Tooltip("Acceleration and deceleration")]
        public float SpeedChangeRate = 10.0f;  Unchanged

        [Space(height: 10)]
        [Tooltip("The height the player can jump")]
        public float JumpHeight = 1.2f;  Unchanged
        [Tooltip("The character uses its own gravity value. The engine default is -9.81f")]
        public float Gravity = -15.0f;  Unchanged

        [Space(height: 10)]
        [Tooltip("Time required to pass before being able to jump again. Set to 0f to instantly jump again")]
        public float JumpTimeout = 0.1f;  Unchanged
        [Tooltip("Time required to pass before entering the fall state. Useful for walking down stairs")]
        public float FallTimeout = 0.15f;  Unchanged

        [Header("Player Grounded")]
        [Tooltip("If the character is grounded or not. Not part of the CharacterController built in grounded check")]
        public bool Grounded = true;  Unchanged
        [Tooltip("Useful for rough ground")]
        public float GroundedOffset = -0.14f;  Unchanged
        [Tooltip("The radius of the grounded check. Should match the radius of the CharacterController")]
        public float GroundedRadius = 0.5f;  Unchanged
        [Tooltip("What layers the character uses as ground")]
        public LayerMask GroundLayers;  Serializable

        [Header("Cinemachine")]
        [Tooltip("The follow target set in the Cinemachine Virtual Camera that the camera will follow")]
        public GameObject CinemachineCameraTarget;  Changed in 0+ assets
        [Tooltip("How far in degrees can you move the camera up")]
        public float TopClamp = 90.0f;  Unchanged
        [Tooltip("How far in degrees can you move the camera down")]
        public float BottomClamp = -90.0f;  Unchanged

        // cinemachine
        private float _cinemachineTargetPitch;
```

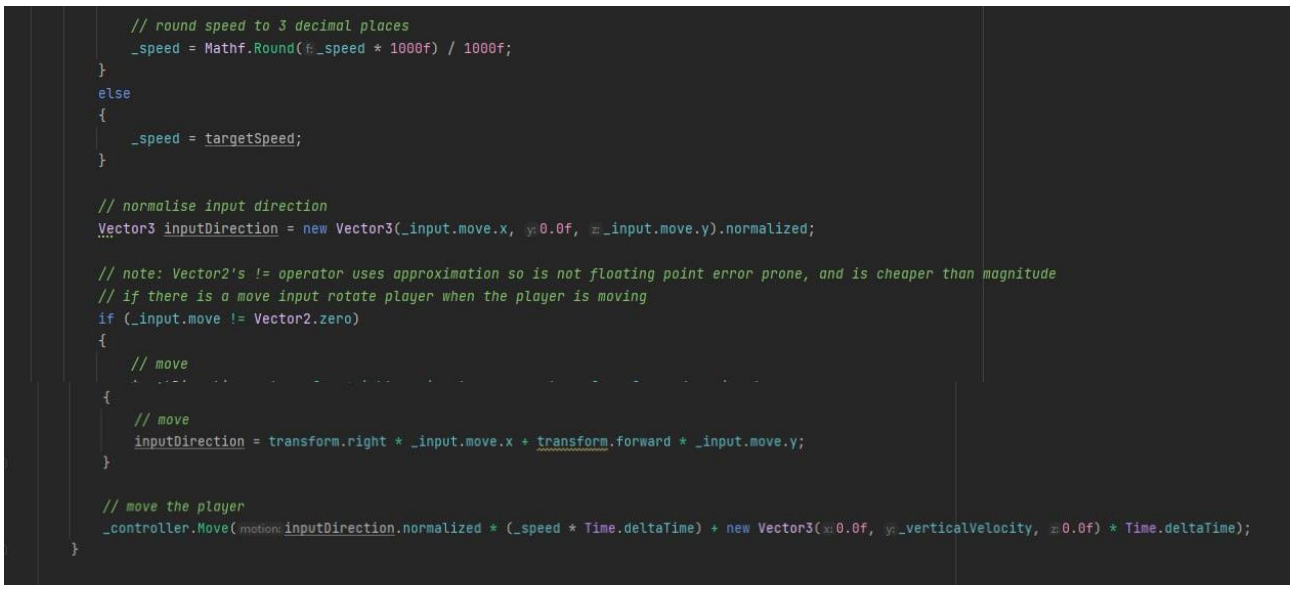
Рисунок Б.1

```
        // cinemachine
        private float _cinemachineTargetPitch;

        // player
        private float _speed;
        private float _rotationVelocity;
        private float _verticalVelocity;
        private float _terminalVelocity = 53.0f;

        // timeout deltatime
        private float _jumpTimeoutDelta;
        private float _fallTimeoutDelta;

        //
        private bool _canRotate = true;
        //


#if ENABLE_INPUT_SYSTEM
        private PlayerInput _playerInput;
#endif
        private CharacterController _controller;
        private StarterAssetsInputs _input;
        private GameObject _mainCamera;

        private const float _threshold = 0.01f;

        1 usage  Yaroslav Ziabrov
        private bool IsCurrentDeviceMouse
        {
            Frequently called
            get
            {
                #if ENABLE_INPUT_SYSTEM
                return _playerInput.currentControlScheme == "KeyboardMouse";
                #else
                return false;
                #endif
            }
        }

        Event function  Yaroslav Ziabrov
        private void Awake()
        {
            // get a reference to our main camera
            if (_mainCamera == null)
            {
                _mainCamera = GameObject.FindGameObjectWithTag("MainCamera");
            }
        }

        Event function  Yaroslav Ziabrov
        private void OnEnable()
        {
            MenuController.OnPause += LockRotation;
            MenuController.OnUnpause += UnlockRotation;
            ObjectiveController.OnPause += LockRotation;
            ObjectiveController1.OnPause += LockRotation;
        }

        Event function  Yaroslav Ziabrov
        private void Start()
        {
            _controller = GetComponent<CharacterController>();
            _input = GetComponent<StarterAssetsInputs>();
#if ENABLE_INPUT_SYSTEM
            _playerInput = GetComponent<PlayerInput>();
#else
            Debug.LogError( "Starter Assets package is missing dependencies. Please use Tools/Starter Assets/Reinstall Dependencies to fix it");
#endif

            // reset our timeouts on start
            _jumpTimeoutDelta = JumpTimeout;
            _fallTimeoutDelta = FallTimeout;
        }

        Event function  Yaroslav Ziabrov
        private void Update()
        {
```

Рисунок Б.2

```
Event function  Yaroslav Ziabrov
private void Update()
{
    /*JumpAndGravity();
    GroundedCheck();*/
    Move();
}

Event function  Yaroslav Ziabrov
private void LateUpdate()
{
    if (_canRotate)
    {
        CameraRotation();
    }
}

Yaroslav Ziabrov
private void GroundedCheck()
{
    // set sphere position, with offset
    Vector3 spherePosition = new Vector3(transform.position.x, y: transform.position.y - GroundedOffset, transform.position.z);
    Grounded = Physics.CheckSphere(spherePosition, GroundedRadius, (int) GroundLayers, QueryTriggerInteraction.Ignore);
}

Frequently called  1 usage  Yaroslav Ziabrov
private void CameraRotation()
{
    // if there is an input
    if (_input.look.sqrMagnitude >= _threshold)
    {
        //Don't multiply mouse input by Time.deltaTime
        float deltaTimeMultiplier = IsCurrentDeviceMouse ? 1.0f : Time.deltaTime;

        _cinemachineTargetPitch += _input.look.y * RotationSpeed * deltaTimeMultiplier;
        _rotationVelocity = _input.look.x * RotationSpeed * deltaTimeMultiplier;

        // clamp our pitch rotation
        _cinemachineTargetPitch = ClampAngle(_cinemachineTargetPitch, lfMin: BottomClamp, lfMax: TopClamp);

        // Update Cinemachine camera target pitch
        CinemachineCameraTarget.transform.localRotation = Quaternion.Euler(x: _cinemachineTargetPitch, y: 0.0f, z: 0.0f);

        // rotate the player left and right
        transform.Rotate(eulers: Vector3.up * _rotationVelocity);
    }
}

Frequently called  1 usage  Yaroslav Ziabrov
private void Move()
{
    // set target speed based on move speed, sprint speed and if sprint is pressed
    float targetSpeed = _input.sprint ? SprintSpeed : MoveSpeed;

    // a simplistic acceleration and deceleration designed to be easy to remove, replace, or iterate upon

    // note: Vector2's == operator uses approximation so is not floating point error prone, and is cheaper than magnitude
    // if there is no input, set the target speed to 0
    if (_input.move == Vector2.zero) targetSpeed = 0.0f;

    // a reference to the players current horizontal velocity
    float currentHorizontalSpeed = new Vector3(_controller.velocity.x, y: 0.0f, _controller.velocity.z).magnitude;

    float speedOffset = 0.1f;
    float inputMagnitude = _input.analogMovement ? _input.move.magnitude : 1f;

    // accelerate or decelerate to target speed
    if (currentHorizontalSpeed < targetSpeed - speedOffset || currentHorizontalSpeed > targetSpeed + speedOffset)
    {
        // creates curved result rather than a linear one giving a more organic speed change
        // note T in Lerp is clamped, so we don't need to clamp our speed
        _speed = Mathf.Lerp(a: currentHorizontalSpeed, b: targetSpeed * inputMagnitude, t: Time.deltaTime * SpeedChangeRate);

        // round speed to 3 decimal places
        _speed = Mathf.Round(f: _speed * 1000f) / 1000f;
    }
    else
    {
        _speed = targetSpeed;
```

Рисунок Б.3



```
            // round speed to 3 decimal places
            _speed = Mathf.Round(f: _speed * 1000f) / 1000f;
        }
        else
        {
            _speed = targetSpeed;
        }

        // normalise input direction
        Vector3 inputDirection = new Vector3(_input.move.x, y: 0.0f, z: _input.move.y).normalized;

        // note: Vector2's != operator uses approximation so is not floating point error prone, and is cheaper than magnitude
        // if there is a move input rotate player when the player is moving
        if (_input.move != Vector2.zero)
        {
            // move
        {
            // move
            inputDirection = transform.right * _input.move.x + transform.forward * _input.move.y;
        }

        // move the player
        _controller.Move( motion: inputDirection.normalized * (_speed * Time.deltaTime) + new Vector3(x: 0.0f, y: _verticalVelocity, z: 0.0f) * Time.deltaTime);
    }
```

Рисунок Б.4