Хмельницький національний університет
Факультет інформаційних технологій
Кафедра інженерії програмного забезпечення

КВАЛІФІКАЦІЙНА РОБОТА

Ігровий застосунок у жанрах "Платформер" та "Головоломка" з використанням інструментів Unity для розробки відеоігор
Назва теми

Рівень вищої освіти Перший (бакалаврський)

Галузь знань 12 «Інформаційні технології»

Спеціальність 121 «Інженерія програмного забезпечення»

Освітня програма Освітньо-професійна програма «Інженерія програмного забезпечення»

Шифр КвРІПЗ.190125.01.01.ПЗ

Виконав студент IV курсу, група ІПЗ-19-1 _____ Боцяновський О.В.
Підпис Ініціали, прізвище

Керівник канд. пед. наук, доцент _____ Праворська Н.І.
Науковий ступінь, звання Підпис Ініціали, прізвище

Нормоконтролер канд. тех. наук, доцент _____ Яшина О.М.
Науковий ступінь, звання Підпис Ініціали, прізвище

До захисту допускаю:

Завідувач кафедри інженерії програмного забезпечення _____ Л. П. Бедратюк
Підпис Ініціали, прізвище

6 червня 2023 р.

Хмельницький 2023

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет Інформаційних технологій

Кафедра Інженерії програмного забезпечення

Рівень вищої освіти Перший (бакалаврський)

Галузь знань 12 «Інформаційні технології»

Спеціальність 121 «Інженерія програмного забезпечення»

Освітня програма Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ

Завідувач кафедри ІПЗ

Л. П. Бедратюк

05.02.2023 р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

Боцяновський Олексій Віталійович

Прізвище, ім'я, по батькові студента

1. Тема кваліфікаційної роботи Ігровий застосунок у жанрах "Платформер" та "Головоломка" з використанням інструментів Unity для розробки відеоігор

Керівник кваліфікаційної роботи канд. пед. наук, доцент Праворська Н. І.

Прізвище, ім'я, по батькові, науковий ступінь, вчене звання

Затверджена наказом ректора університету від 01.03.2023 р. № 5

2. Строк подання студентом роботи на кафедру 01.06.2023 р.

3. Вихідні дані до роботи Матеріали переддипломної практики

4. Зміст пояснювальної записки (перелік питань, які потрібно розробити) *Характеристика предметної області та постановка задачі, проектування програмного забезпечення, програмна реалізація.*

5. Перелік графічного матеріалу (із зазначенням обов'язкових креслень) Три креслення формату А3

6. Консультанти розділів кваліфікаційної роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	Яшина О.М., к.т.н, доцент	25.05.23	06.06.23
Антиплагіат	Гурман Т.В., к.т.н, доцент	1.06.23	1.06.23

7. Дата видачі завдання «02» січня 2023 р.

КАЛЕНДАРНИЙ ПЛАН

Назва етапів (розділів) кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1. Збір матеріалу за темою кваліфікаційної роботи (КвР), дослідження предметної області, в якій планується використання програмного забезпечення (ПЗ), визначення здач та вимог, розробка технічного завдання	02.01–31.01 2023	
2. Проектування програмного забезпечення	01.02–28.02 2023	
3. Програмна реалізація програмного забезпечення	01.03–10.04 2023	
4. Тестування програмного забезпечення	11.04–30.04 2023	
5. Написання вступу, загальних висновків, оформлення переліку джерел та посилання на додатків. Оформлення пояснювальної записки КвР згідно вимог	01.05–25.05 2023	

Студент _____ Боцяновський О. В.
Підпис Ініціали, прізвище

Керівник роботи _____ Праворська Н. І.
Підпис Ініціали, прізвище

АНОТАЦІЯ

Тема кваліфікаційної роботи «Ігровий застосунок у жанрах “Платформер” та “Головоломка” з використанням інструментів Unity для розробки відеоігор».

Автор роботи: Боцяновський Олексій Віталійович.

Керівник роботи: Праворська Наталія Іванівна

Пояснювальна записка: 66 с., 39 рис., 2 табл., 2 дод., 44 джерел.

Графічна частина: 3 креслення формату А3.

Об’єктом дослідження кваліфікаційної роботи є процес створення ігрового застосунк3 у жанрах «Платформер» та «Головоломка» з використанням інструментів Unity для розробки відеоігор, а також розробка архітектури, компонентів і візуальної складової застосунку.

Предметом дослідження кваліфікаційної роботи є методи проектування та розробки ігрового застосунку з використанням інструментів Unity.

Метою кваліфікаційної роботи є створення ігрового застосунку у жанрах «Платформер» та «Головоломка» з використанням інструментів Unity для розробки відеоігор, а також розробка архітектури, компонентів і візуальної складової застосунку.

У кваліфікаційній роботі було проведено аналіз предметної області, вивчено вже наявні на ринку ігрові застосунки, їх переваги та недоліки. На основі отриманої інформації було визначено вимоги до програмного забезпечення та здійснено його детальне проектування з вибором технологій розробки.

Використовуючи дані проектування, було розроблене програмне забезпечення, яке відповідає усім поставленим вимогам та за потреби може бути швидко модифікована у майбутньому.

6.06
Дата

Підпис

ВІДОМІСТЬ ДОКУМЕНТІВ

№ рядка	Формат	Позначення документа	Найменування документа	К-сть аркушів	№ екз.	Примітка
			Текстові документи			
1	А4	КвРІПЗ.190125.01.01.ПЗ	Пояснювальна записка	66		
2	А4		Завдання на кваліфіка-	1		
			ційну роботу			
3	А4		Анотація	1		
			Графічні документи			
4	А3	КвРІПЗ.190125.01.01.Е8	Діаграма декомпозиції	1		
			IDEF0 процесу			
			реалізації застосунку			
5	А3	КвРІПЗ.190125.01.01.Е8	Діаграма декомпозиції	1		
			IDEF0 алгоритму			
			проходження рівня			
6	А3	КвРІПЗ.190125.01.01.Е8	Діаграма варіантів	1		
			використання			

Змн.	Арк.	№ докум.	Підпис	Дата	КвРІПЗ.190125.01.01.ВД		
Виконав		Боцяновський О.В.		6.06	Ігровий застосунок у жанрах "Платформер" та "Головоломка" з використанням інструментів Unity для розробки відеоігор	Літ.	Арк. 1 / Аркушів 1
Керівник.		Праворська Н. І.		6.06			
						ХНУ, ІПЗ-19-1	
Н. контр		Яшина О. М.		6.06			
Зав.кафедри		Бедратюк Л. П.		6.06			

ЗМІСТ

ВСТУП

Ігрова індустрія на сьогоднішній день займає одну із передових позицій на ринку цифрових технологій. Комп'ютерні ігри є одними із найдоступніших видів розваг і з розвитком технологій мобільних приладів і персональних комп'ютерів стали одними із найдоступніших.

Розробка відеоігор є важливим джерелом технологічного прогресу. Розробники відеоігор постійно вдосконалюють графічні та інші технології, що дозволяє створювати більш складні та реалістичні світи. Це може мати приклади у більш широкому спектрі індустрій, від медицини та автомобілебудування до наукових досліджень та військових технологій. Розробка передових комплектуючих для персональних комп'ютерів і частково для мобільних приладів пов'язана із великою популярністю відеоігор, що в свою чергу дозволяє розробникам відеоігор розробляти все кращі і реалістичніші ігри.

Також розвивається програмне забезпечення для розробки відеоігор і таким чином стає доступнішим і зрозумілішим для початківців у сфері розробки відеоігор, що в свою чергу збільшує асортимент відеоігор для користувачів різних платформ. Окрім цього, для розробників відеоігор майже не існує перешкод для реалізації своєї продукції по всьому світу.

Розробка відеоігор є важливим галузевим сектором, який генерує значні прибутки. Глобальна індустрія відеоігор оцінюється в мільярди доларів, а це означає, що вона має значний економічний вплив.

Успішність відеогри залежить лише від її якості і актуальності для світової повістки, саме тому, незважаючи на слабкий розвиток індустрії відеоігор у пострадянських країнах, індустрія відеоігор є дуже прибутковою для розробників різних масштабів.

Зважаючи на значну кількість доступних джерел, доступність програмного забезпечення для розробки дана індустрія є актуальною для її розвитку і працевлаштуванню у ній.

Двовимірна відеоігри у жанрі «Платформер» є досить популярними серед гравців і розробників відеоігор. Це жанр, в якому гравець керує персонажем, який має переміщуватися від однієї точки до іншої, перестрибуючи перешкоди, та взаємодіяти з різними об'єктами в грі.

Однією з причин популярності цього жанру є його простота. Відеоігри у жанрі «Платформер» досить легкі для розуміння та грати, що робить їх привабливими для новачків та для тих, хто не хоче витрачати багато часу на навчання складніших механік гри.

Підсумовуючи, розробка відеоігор є актуальною через їх культурну та економічну вагу, технологічний прогрес, можливості навчання та розвитку, а також популярність серед широкої аудиторії. А обрані жанри для відеогри є простими і популярними для великої кількості користувачів, що зробить її орієнтованої на велику аудиторію.

Мета кваліфікаційної роботи: створення ігрового застосунку у жанрах «Платформер» та «Головоломка» з використанням інструментів Unity для розробки відеоігор, а також розробка архітектури, компонентів і візуальної складової застосунку.

Об’єкт дослідження кваліфікаційної роботи: процес створення ігрового застосунку у жанрах «Платформер» та «Головоломка» з використанням інструментів Unity для розробки відеоігор, а також розробка архітектури, компонентів і візуальної складової застосунку.

Предмет дослідження кваліфікаційної роботи: методи проектування та розробки ігрового застосунку з використанням інструментів Unity.

Задача кваліфікаційної роботи: розробити архітектуру, компоненти і візуальну складову ігрового застосунку у жанрах «Платформер» та «Головоломка».

1 ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Характеристика функціональної структури предметної області

Ігровий застосунок – це комп'ютерна програма, яка здійснює організацію ігрового процесу з іншими гравцями або сама з собою з використанням користувальницького інтерфейсу та пристроїв введення. Під час ігрового процесу за допомогою ігрового рушія у віртуальному просторі здійснюється імітація прямої взаємодії між ігровим персонажем або внутрішньо ігровим об'єктом з користувачем за допомогою програмного коду.

Головними елементами для здійснення маніпуляції у грі є клавіатура або її аналоги та «миша». Також користувач може використовувати інші прилади для контролю над ігровим процесом, наприклад джойстики, контролери та VR обладнання, які дозволяють по іншому отримувати задоволення від гри.

Основним елементом виведення інформації для всіх ігор є візуальний елемент, таким чином користувач на екрані монітора спостерігає за ігровою ситуацією і може на неї впливати за допомогою засобів введення. Окрім цього, у кожній грі можуть бути реалізовані і інші елементи виведення інформації такі як: звук, вібрація, зворотній механічний зв'язок у кнопок. Для цього у користувача повинні бути прилади і контролери, які можуть підтримувати реалізовані функції у самій грі.

Через стрімкий розвиток ігрової індустрії класифікація ігрових застосунків є неточною і описати гру одним жанром, інколи, майже неможливо. Через це, у певній частині випадків, різних джерелах відеогру можуть класифікувати за різними жанрами. Саме через це, як розробники, так і користувачі прийшли до певного консенсусу у даному питанні і виділили п'ятнадцять основних геймплейних елементів з яких може складатися гра. Завдяки цим критеріям і розробники, і користувачі можуть визначити як мінімум один жанр відеогри.

Для його визначення необхідно взяти одну з п’ятнадцяти дій, яка найчастіше повторюється і звірити її з таблицею. Приклад даної таблиці представлено в таблиці 1.1.

Таблиця 1.1 – Таблиця класифікації ігрових застосунків по жанрам

Категорія	Ігри контролю	Ігри інформації	Ігри дій
Базові геймлейні елементи	Піклування, Створення, Контроль, Тактика, Планування	Навчання, Загадки, Взаємодія, Роль, Вивчення	Збирання, Ухилення, Знищення, Змагання, Водіння
Сім елементів	Real Time Strategy (RTS)		Action Role Play Game
П’ять елементів	Global strategy	Open RPG	Open action
Три елемента	Strategy Sim strategy Global wargame	RPG, MMORPG, MUD	Action, Slasher, Battle racing
Два елемента	Economic, Tower defense, Wargame, Card game	Puzzle, Quest, Browser`s, Adventure	Platformer, Stealth, Fighting, Racing
Один елемент	Arcade, Horror, Shooter, Sport, Simulator	Education, Test, Contact, Hero, Tour	Life sim Build, Control, Tactic, Logic

Також вищеописані жанри поділяються на три основних класи, а саме на: ігри контролю, ігри інформації та ігри дій. Окрім цього, жанри які описані в таблиці 1.1 в основному пишуться на англійській мові і певна частина з них не може бути перекладена на інші мови правильно, оскільки деякі з них є термінами.

Дана ситуації склалася через те, що центром розвитку технологій і самих ігрових застосунків були країни заходу, де англійська мова була універсальної для всіх. Приклад даних, які знаходяться в таблиці 1.1 можна побачити на рисунку 1.1.

Рисунок 1.1 – Графічне зображення жанрової класифікації ігрових застосунків

Головною рисою для категорії ігор контролю є управління якимось ігровим об'єктом або групою об'єктів. В основному в даній категорії значну кількість займають ігри стратегії і їх різновиди.

Ігри інформації наповнені ігровими елементами, які заставляються користувача взаємодіяти з інформацією, її аналізувати і використовувати у наступних ігрових ситуаціях.

Ігри дій пов'язані з головним героєм і надають величезний функціонал для головного героя. В даних ігрових застосунках користувач керує лише головним героєм, який може бути ким завгодно.

Сучасна ігрова індустрія одна із найбільш прибуткових і передових галузей у сфері розваг. По даним проведених досліджень, глобальний ринок відеоігор у 2020 році сягав більше ста шістдесяти мільярдів доларів США, що свідчить про його значну популярність як виду розваг серед гравців усього світу.

Сьогодні, відеоігри доступні на більшості платформ: від персональних комп'ютерів та консолей до мобільних пристроїв та браузерів. На ринку відеоігор присутня велика кількість компаній-виробників, серед яких можна виділити такі відомі назви, як Electronic Arts, Activision Blizzard, Ubisoft, Nintendo, Sony та Microsoft.

Індустрія відеоігор включає в себе значний рівень конкуренції серед гігантів ігрової індустрії та великий рівень інновацій. Розробники постійно вдосконалюють графічні рушії та використовують найновіші технології, щоб створювати більш реалістичні та іммерсивні ігрові світи.

Популярними жанрами відеоігор на сьогоднішній день є перестрілки від першої та третьої особи, рольові ігри, пригодницькі ігри, спортивні ігри, гонки, стратегії та ігри-головоломки.

Однак, індустрія відеоігор стикається з рядом проблем, таких як кібербуллінг, залежність від гри та інші. У зв'язку з цим, розробники стежать за тим, щоб їх ігри відповідали вимогам етики та моралі.

Також, відеоігри стають все популярнішими в електронному спорті, де вони використовуються як засіб змагань та розвитку командної співпраці.

Існує багато плюсів відеоігор, які роблять їх популярними серед гравців усього світу. Деякі з цих переваг включають в себе наступні аргументи.

Відеоігри можуть розвивати когнітивні здібності, такі як увагу, концентрацію, пам'ять та спритність. Ігри-головоломки, наприклад, допомагають розвивати логіку та розв'язувати складні проблеми.

Багато відеоігор мають мультиплеєрний режим, що дозволяє гравцям грати разом з іншими гравцями з усього світу та взаємодіяти з ними. Це допомагає вдосконалювати соціальні навички, такі як комунікація, співпраця та взаємодія з іншими людьми.

Грати в відеоігри може бути джерелом розваг та розважальних відчуттів. Це може допомогти підвищити настрій та зменшити стрес.

Відеоігри можуть стати джерелом творчого натхнення та допомогти розвивати творчі навички, наприклад, шляхом вільного експериментування з геймплеєм.

Деякі відеоігри, зокрема ігри-бойовики, можуть допомогти розвивати реакцію та координацію рухів.

Деякі відеоігри можуть допомогти поглиблювати знання в певних областях, таких як історія, наука та математика, а також можуть бути спеціалізованими для навчання.

Жанр відеоігор «Платформер» це жанр ігор, де головний герой повинен пройти через серію рівнів, стрибаючи від платформи до платформи, уникаючи перешкод, збираючи бонуси та знищуючи ворогів.

Ігри цього жанру зазвичай розробляються у двовимірному або тривимірному просторі, де камера зазвичай розташовується в бічному або діагональному ракурсі.

Основна мета – це гравця досягти кінцевої точки рівня, яка зазвичай позначена прапорцем або іншим символом. Зазвичай, на кожному рівні є багато секретів та прихованих місць, що відкриваються при виконанні певних умов або завдань.

У платформерах головними рухами є біг, стрибок та атака. Часто гравці повинні зіграти роль атлета, щоб долати складні перешкоди та пройти весь рівень без помилок. При цьому важливо правильно визначити момент для стрибка, адже недостатньо вміти стрибати на велику відстань, потрібно ще й точно приземлятись.

Платформери можуть бути як одиночними, так і мультиплеєрними, де гравці можуть спільно проходити рівні, допомагаючи один одному.

Деякі з найвідоміших платформерів – це Super Mario Bros., Sonic the Hedgehog, Donkey Kong, Rayman, Crash Bandicoot та Braid. Ці ігри зайняли своє місце в історії відеоігор, тому що вони надихали та захоплювали гравців у всьому світі, і до цього дня залишаються надзвичайно популярними.

Жанр відеоігор «Головоломка» це жанр ігор, де гравець повинен вирішувати різноманітні логічні завдання та головоломки. У цих іграх головна мета полягає в тому, щоб знайти рішення проблеми, яка часто є складною та неочевидною на перший погляд.

Відеоігри цього жанру можуть бути різноманітними: від класичних головоломок, де гравець повинен розгадати складну послідовність дій, до більш складних та інноваційних проектів, які вимагають від гравця вирішувати завдання на основі фізики, логіки та механіки.

До класичних представників жанру головоломок належать такі ігри, як Tetris, Sudoku, Rubik's Cube, а також гра з в'язанням вузлів - Untangle. До більш складних головоломок можна віднести ігри серії Portal, The Witness та Monument Valley, де гравцю потрібно не тільки розгадувати складні завдання, але й маніпулювати простором та змінювати перспективу.

У жанрі головоломок дуже популярним є також використання елементів гри віртуальної реальності, що дає можливість гравцям взаємодіяти з ігровим світом на більш іммерсивному рівні.

Головоломки відомі своєю високою складністю, що часто приводить до використання різноманітних візуальних та звукових ефектів, які можуть допомогти гравцям у процесі вирішення завдань. Жанр головоломок завжди викликає інтерес серед гравців, які люблять викликати свій розум та вирішувати складні завдання.

Ігровим рушієм називають програмне забезпечення для розробки відеоігор. Ігрові рушії містять у собі набір інструментів, бібліотек та API(Application Programming Interface), які дозволяють розробникам створювати відеоігри за допомогою написання скриптів, або використовуючи наявні інструменти.

Ігровий рушій забезпечує базові функції, такі як рендер графічних елементів, фізику у об’єктів, звук, штучний інтелект та управління об’єктами за допомогою приладу для введення. Він також дозволяє розробникам створювати та налаштовувати ігрові об'єкти, рівні, персонажів та інші елементи гри.

Ігрові рушії мають різну функціональність та спрямованість. Деякі рушії призначені для розробки конкретних жанрів відеоігор, таких як аркади або стратегії. Інші ж рушії надають розробникам більш широкий набір інструментів, що дозволяє створювати відеоігри різних масштабів.

Використання ігрових рушіїв дозволяє значно спростити та прискорити процес розробки відеоігор, а також забезпечити якісний графічний та звуковий дизайн. Також розробники можуть вибрати підходящий рушій для їх проекту.

1.2 Аналіз наявного програмно-технічного забезпечення предметної області

Жанри «Платформер» і «Головоломка» є популярними серед малих студій розробників відеоігор, а також існує багато популярних ігрових застосунків, які відносяться до предметної області. Одним із таких ігрових застосунків є популярна відеогра «PICO PARK». Зовнішній вигляд відеогри зображений на рисунку 1.2 [26].

Рисунок 1.2 – Зовнішній вигляд відеогри «PICO PARK»

Гра розрахована на кількох гравців, а саме мінімум на двох і максимум на вісьмох. Але свою популярність вона здобула не просто наявністю кооперативного режиму, а завдяки різноманітній кількості механік, інтерактивних дій з навколишнім середовищем і тісній взаємодії між гравцями. Дана відеогра є двовимірною з мінімалістичними інтерфейсом і графікою.

Ігровий процес ставить перед гравцями задачу пройти рівень, а для цього їм необхідно знайти і отримати ключ, а потім відкрити двері і пройти рівень. Для того, щоб здобути ключ гравцям потрібно в кооперативі зі своїми друзями розгадувати загадки і взаємодіяти з навколишніми предметами, наприклад рухати рухомі блоки, чи змінювати розмір свого персонажа. Всього у грі є сорок вісім рівнів зі зміною розділів рівнів, де у кожного розділу свої механіки і головоломки. Зовнішній вигляд взаємодії гравців з навколишнім середовищем і один з одним, а також ключ і двері зображено на рисунку 1.3 [26].

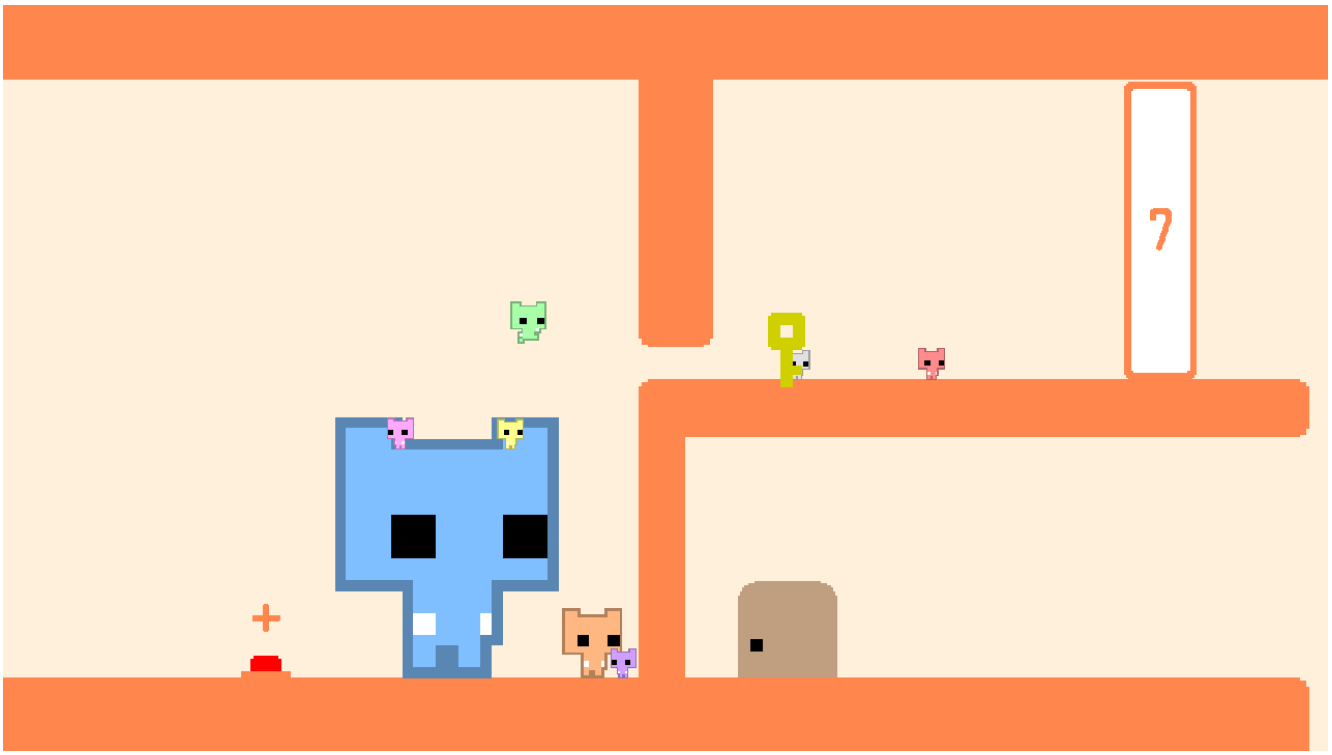


Рисунок 1.3 – Зовнішній вигляд елементів відеогри «PICO PARK»

Як видно з рисунку один з гравців несе ключ для того, аби відкрити двері, частина гравців змінила розмір свого персонажа, а також на рисунку присутній блок з цифрою сім, що означає що для його руху необхідно сім гравців.

При аналізі переваг і недоліків даної відеогри, можна констатувати той факт, що у даній відеогрі відсутні провальні рішень, а також немає і недоліків. Щодо переваг, то дана гра привертає свою увагу варіативністю загадок, які реалізовані у грі, легким для сприйняття інтерфейсом і дизайном, а також можливістю проходити дану відеогру разом зі своїми друзями. Окрім цього, однією з переваг є можливість запускати даний ігровий застосунок не тільки з операційної системи Windows, а й з платформ Nintendo і Steam Deck, і керування можна здійснювати за допомогою клавіатури, геймпада чи з телефону за допомогою мобільного застосунку TECOGAMEPAD.

Наступним ігровим застосунком для аналізу є відеогра «Fez». Дана відеогра розрахована на одного гравця і містить певний лінійний сюжет, з додатковими гілками, які не впливають на основний. Зовнішній вигляд відеогри і одного з рівнів зображено на рисунку 1.4 [27].

Рисунок 1.4 – Зовнішній вигляд відеогри «Fez»

Вона є двовимірною, але при цьому гравець може прокручувати камеру на дев’яносто градусів, тим самим персонаж яким керує користувач переміщується в тривимірному просторі. Дана механіка і головною є головною особливістю

відеогри, і дозволяє здійснювати дії, які були би неможливими у тривимірному просторі і на перший погляд виглядають як оптичні ілюзії. Наприклад гравець може перемістити персонажа на іншу частину карти при правильній зміні проекцій камери. Для даної відеогри один із розробників написав власний ігровий рушій Trixel, особливістю якого є те, що він перетворює двовимірний куб на тривимірний.

Відеогра поділена на рівні, де для проходження на наступний рівень гравець повинен зібрати один куб з восьми фрагментів. Сумарно гравець за перший раз проходження сюжету повинен зібрати тридцять два куба, але в кінці гри користувач отримує нову механіки гри. Дана механіка дозволяє гравцю бачити потаємні ходи на карті і рухатися по цих ходах в яких знаходяться додаткові куби, тим самим гра дозволяє ще раз пройти сюжет, але в цьому випадку гравцю потрібно буде зібрати шістдесят чотири куба. Зовнішній вигляд геймплею даної відеогри зображено на рисунку 1.5 [27].

Рисунок 1.5 – Геймплей відеогри «Fez»

Аналізуючи переваги і недоліки, варто зазначити, що явних недоліків у цієї гри немає, натомість негативним аспектом можна вважати довгий сюжет, який потрібно проходити повторно. Перевагою даного застосунку є унікальна механіка

повороту камери, яка і є основною для даної відеогри, а також при проходженні певного етапу відеогри відкриття нової механіки відображення карт. Як і в попередній відеогрі, притаманною рисою даного платформера-головоломки є система рівнів, а також наявність абстрактного ключа для переходу на наступний рівень.

Останнім ігровим застосунком для аналізу є тривимірна відеогра «Unravel» з фіксованою позицією камери, а саме збоку, що в свою чергу робить її ізометричним двовимірним платформером. Дана відеогра розрахована на одного гравця і містить атмосферний лінійний сюжет з додатковими завданнями на кожному з рівнів, які є необов'язковими. Відеогра поділена на рівні і на кожному з рівнів гравець повинен знайти ключовий предмет, а саме один із предметів із альбому. Гра підкріплена чудово підібраним музичним супроводом, що робить сюжет ще більш захоплюючим. Відеогра доступна на різних платформах і гравець може керувати головним персонажем різними пристроями для введення. Відеогра використовує ігровий рушій PhyreEngine, який орієнтований на ігрові приставки компанії Sony, але також дозволяє створювати ігри на операційні системи Windows та Linux Зовнішній вигляд відеогри зображено на рисунку 1.6 [28].

Рисунок 1.6 – Зовнішній вигляд відеогри «Unravel»

Основна механіка даного ігрового застосунку це обмеження дальності переміщення ниткою. Головний персонаж являє собою клубок ниток і при переміщенні персонажа користувачем він втрачає довжину нитки, що в свою чергу і є лімітом на переміщення. Гравець може змотувати нитку, але для цього йому необхідно рухатися в напрямку звідки він прийшов. Також гравець повинен використовувати нитку для вирішення задач, які є на рівні, наприклад він може побувати міст, закріпивши нитку на ключових позиціях, забиратися вверх по нитці, якщо вона закріплена зверху. Також ігровий персонаж може взаємодіяти з іншими незакріпленими тілами і їх рухати. Відеогра поділена на контрольні точки на яких ліміт нитки відновлюється на стільки на скільки необхідно для досягнення наступної контрольної точки, тому гравець не може отримати певний запас для майбутнього. У відеогрі реалізовані неігрові персонажі, які налаштовані агресивно щодо головного героя і можуть його атакувати, якщо вони все таки ударять головного персонажа, то він повернеться на попередню контрольну точку. Як виглядають вищеописані механіки гри можна побачити на рисунку 1.7 [28].

Рисунок 1.7 – Геймплей відеогри «Unravel»

Констатуючи переваги і недоліки даної відеогри можна сказати, що у неї відсутні недоліки, оскільки гра не охоплює велику кількість механік. Безумовно, основною перевагою даного ігрового застосунку є його унікальна механіка, яка обмежує дії користувача і заставляє його наперед обдумувати свої дії. Другою перевагою відеогри є її пророблений сюжет, який стає кращим завдяки ізометричній графіці і музичному супроводу. Як і в інших проаналізованих відеоіграх притаманною рисою є система рівнів з наявністю ключового предмета.

Підсумовуючи аналіз даних трьох відеоігор, які відповідають предметній області, можна зазначити, що рішення про використання системи рівнів і пошук певного предмета для проходження рівня є вдалим, і його слід використати для майбутнього ігрового застосунку. Також при аналізі виявлено, що для майбутніх користувачів слід нарощувати складність рівнів і механік по мірі проходження відеогри, що буде підтримувати інтерес користувача до майбутнього ігрового застосунку, але при цьому не потрібно їх робити дуже важкими аби користувач не вирішив закінчити її проходження. Пряма і непряма взаємодія з навколишніми об'єктами і середовищем також є вдалим рішенням, що дозволяє робити рівні не такими легкими. Варіативними рішеннями є наявність сюжету і ворогів на рівнях, оскільки це не є головним аспектом для жанрів «Платформер» і «Головоломка», але однозначно це є плюсом для гри. Звукові ефекти є обов'язковими, але повноцінна музика є необов'язковою, таким чином при наявності звуків користувач матиме можливість відчувати зворотній зв'язок від відеогри.

1.3 Визначення функціональних вимог до програмного забезпечення

Розроблюваний застосунок являє собою двовимірну відеогру-платформер для персонального комп'ютера. Основною метою є розробка даного застосунку для його реалізація у сфері розваг. На основі аналізу предметної області ігровим рушієм на якому буде розроблений майбутній застосунок стане Unity, а також будуть використовуватися фреймворки, які є інтегрованими в даний рушій.

Для написання коду для ігрового застосунку було обрано інтегроване середовище розробки Microsoft Visual Studio, оскільки дане середовище для розробки містить плагіни, які можуть пришвидшувати написання коду для скриптів Unity і аналізувати його на помилки, а також через те, що воно безкоштовне. Для написання скриптів у Unity використовують об'єктно-орієнтовану мову програмування C# або візуальну мову для написання скриптів Bolt.

Для майбутнього ігрового застосунку основними функціональними вимогами можна визначити саме такі:

– побудування застосунку для операційної системи Windows;

– реалізація схем контролю ігровим процесом за допомогою клавіатури і миші та геймпадів;

– використання двовимірної графіки у ігровому застосунку;

– використання звукових ефектів для супроводу ігрового процесу.

Опис ігрового процесу і механік:

– у відеогрі буде відсутній сюжет;

– у відеогрі буде реалізована система рівнів з відкриттям наступного рівня при завершенні попереднього і можливістю запустити любий доступний рівень;

– у відеогрі на кожному з рівнів буде реалізований об'єкт, який є попередньо заблокованим, в який гравцю необхідно буде зайти, щоб завершити рівень;

– у відеогрі для розблокування об'єкту виходу з рівня буде реалізований інший об'єкт в якому гравець повинен буде вирішити задачу-головоломку, використовуючи необхідні предмети;

– у відеогрі гравець повинен буде знайти на рівні необхідні для вирішення задачі-головоломки предмети і яких буде рівно стільки, скільки необхідно для її вирішення;

– у відеогрі на рівнях будуть передбачені об'єкти-пастки при доторканні до яких головний персонаж буде повертатися на контрольну точку;

– у відеогрі об’єкти-пастки будуть активними з самого початку гри, або активуватимуться і деактивуватимуться по мірі проходження рівня гравцем;

– у відеогрі по мірі проходження рівнів для гравця стануть доступна нові механіки для виявлення прихованих об’єктів і активації нових об’єктів на певний час для того аби пройти рівень.

Дані вимоги були сформовані основуючись на аналізі наявних рішень, а також на основі аналізу їх переваг і недоліків. Використовуючи ці сформовані вимоги, буде проводитися розробка програмного застосунку відповідно до теми кваліфікаційної роботи .

Також для майбутнього ігрового застосунку необхідно визначити модель життєвого циклу. Необхідно вибрати одну з чотирьох найкращих моделей, а саме: каскадна модель, спіральна модель, RAD модель та інкрементна модель.

Найкращою моделлю для розробки майбутнього ігрового застосунку є модель інкрементна модель тому що вона дозволяє розробляти відеогру поетапно, додаючи нові функції на кожному етапі, що дозволяє швидко реагувати на зміни вимог та швидко створювати ігровий контент. Відеогри зазвичай мають складну функціональність, тому інкрементна модель дозволяє поетапно розробляти та тестувати окремі частини гри, що дозволяє збільшувати якість та швидкість розробки.

Каскадна модель життєвого циклу підходить для проектів зі статичними вимогами та чітко визначеною функціональністю, що не є характерним для відеоігор.

Спіральна модель та RAD модель підходять для проектів з високим ризиком та необхідністю швидкої реакції на зміни вимог, але майбутній ігровий продукт має чіткі функціональні вимоги і не вважається проектом з високим ризиком, тому дані моделі життєвого циклу менше підходять для розробки майбутньої відеогри.

Інкрементна модель передбачає, що вимоги не завжди є добре відомими і їх визначення не є легким, але вони можуть бути визначеними у кожному циклі, але при цьому вони не змінюються дуже часто.

Також дана модель передбачає, що сформовані вимоги демонструються складність програмної системи і, що вимоги володіють функціональними властивостями на ранньому етапі циклу.

Також дана модель життєвого циклу має характеристики щодо команди розробників, а саме те, що розробники добре володіють предметною областю і вона не є новинкою для них, але враховуючи той факт, що деякі технології, які передбачає майбутній ігровий застосунок можуть бути новими для деяких розробників.

При цьому розробники повинні добре володіти інструментами для розробки програмного продукту і керівник проекту може змінювати їх ролі під час розробки, а також виділяти час для того аби розробник вивчив нову технологію. Також при даній моделі менеджер або керівник повинен суворо відслідковувати прогрес розробки програмного продукту і орієнтуватися на думки інших розробників для покращення якості програмного продукту.

При цьому даній моделі життєвого циклу притаманно, що користувачі не зможуть відслідковувати хід розробки програмного забезпечення і їх присутність буде обмеженою, але вони будуть ознайомлені з проблемами предметної області програмного продукту. Також користувачі не будуть задіяні на всіх етапах життєвого циклу.

На рисунку 1.8 можна побачити як виглядає інкрементна модель життєвого циклу програмного продукту [44].

Рисунок 1.8 – Інкрементна модель життєвого циклу програмного продукту

Також варто відзначити, що при даній моделі життєвого циклу фінансування проекту не завжди може бути стабільним, а тому і графік розробки програмного продукту є обмеженим. Програмний продукт при інкрементній моделі розрахований на довгострокову експлуатацію, що в свою чергу означає, що у програмного рішення повинен бути високий ступінь надійності і можливості для розширення функціоналу.

1.4 Висновки. Постановка задачі

У даному розділі було проведено детальний аналіз сфери розробки ігрових застосунків. Були докладно проаналізовані різноманітні жанри відеоігор, а також була створена таблиця, яка включає всі ці жанри. Зокрема, були розглянуті і проаналізовані два основних жанри, які будуть використовуватися в розроблюваному ігровому застосунку – «Платформер» і «Головоломка».

Також були детально проаналізовані наявні програмні рішення. Для цього було взято три відеоігри, що мають схожі жанри між собою і з майбутнім розроблюваним додатком. В ході аналізу були розглянуті переваги та недоліки

цих ігор, а також були вивчені успішні рішення, що були реалізовані в цих ігрових застосунках.

Завдяки усій отриманій інформації при аналізі функціональної структури, предметної області та аналізу наявних програмних рішень були визначені основні вимоги на розробку майбутнього програмного продукту, його функціональні , вимоги до інтерфейсу користувача та інше. Основними задачами на реалізацію є:

– реалізувати перехід між рівнями;

– реалізувати головні механіки відеогри;

– реалізувати графічну складову ігрового застосунку;

– реалізувати логіку взаємодії компонентів ігрового рушія.

В даному розділі, були визначені вимоги до майбутнього ігрового застосунку. Були проведені докладні аналізи різних моделей життєвих циклів, серед яких була обрана інкрементна модель життєвого циклу як найбільш відповідна для даного проекту.

2 ПРОЕКТУВАННЯ ІГРОВОГО ЗАСТОСУНКУ

2.1 Архітектурна та функціональна структура ігрового застосунку

При розробці простих ігрових застосунків розробники часто використовують якусь одну архітектуру з додаванням різних патернів, які є зручними для реалізації необхідного компонента. Для розробки невеликих проектів розробники в основному використовують або архітектуру MVC або ECS, але при цьому не дотримуються їх суворо.

Архітектура MVC або Model-View-Controller розділяє програму на три компоненти: модель, вид і контролер. Модель відповідає за збереження даних, що використовуються в застосунку. У випадку ігрових застосунків це може бути стан гри, дані про об'єкти, з якими взаємодіє гравець, а також різноманітні налаштування та параметри гри. Модель не має жодних залежностей від інших компонентів системи.

Представлення відповідає за те, як дані з моделі відображаються на екрані. В ігровій розробці це може бути графічний інтерфейс користувача, який відображає гру та інформацію для гравця. Представлення отримує дані з моделі та відображає їх, не маючи жодних залежностей від контролера. В основному у Unity представленнями є елементи Canvas і його складові.

Контролер відповідає за обробку дій користувача та оновлення моделі та представлення. У випадку ігрових застосунків контролер може обробляти події, які виникають в грі, наприклад, введення гравця з клавіатури або натискання кнопок на ігровому контролері. Контролер оновлює модель, щоб зберегти новий стан гри, а також відправляє повідомлення до представлення для оновлення відображення даних у грі.

Ця архітектура забезпечує гнучкість і можливість повторного використання коду. Проте дана архітектура збільшує складність коду, збільшує об’єм контролеру до розмір при яких з ним стає важко працювати і знижує швидкодію ігрового застосунку, тобто робить процес оптимізації важчим.

Архітектура ECS або Entity-Component-System заснована на розділенні компонентів логіки гри та їх управління системами, що дозволяє підвищити швидкодію та гнучкість гри. ECS складається з трьох основних компонентів: сутність (Entity), компонент (Component) та система (System). Сутність – це простий об'єкт в грі, який може бути складений з різних компонентів. Сутності не містять логіки, вони просто володіють набором компонентів. Компонент – це конкретна частина логіки гри, яка додається до сутності. Компоненти можуть містити різні дані та функції, такі як графіку, фізику, поведінку та інші. Кожен компонент має унікальний ідентифікатор, який дозволяє системі легко знайти та звернутися до нього. Система – це компонент, який керує взаємодією між сутностями та їх компонентами. Системи виконують всю необхідну логіку, де кожна система відповідає за певний набір компонентів та виконує потрібні дії з кожним компонентом.

У ECS взаємодія між компонентами відбувається через сутності, які містять ці компоненти. Системи виконують дії з цими компонентами, а сутності виступають лише як контейнери для них. Це дозволяє легко додавати, видаляти та змінювати компоненти без зміни сутності. Також для зручності компонент можна зробити ознакою для визначення сутності поміж інших і при цьому компонент залишається пустим. При даній архітектурі файли групуються не за типом, а за функціоналом який вони виконують. Проте дана архітектура має свої недоліки, такі як високий поріг входження, велика кількість файлів і класів і передбачає майже повну ізольованість систем одна від одної, що в свою чергу змушує розробника за допомогою інших методів здійснювати взаємодію між системами.

Також при розробці ігрових застосунків об’єднують компонент і систему в одне ціле таким чином це перетворює ECS в архітектуру ЕС, яка являє собою ООП з модульністю і використанням патерну Composition Over Inheritance. При такому підході компонент може сам обробляти і змінювати дані всередині себе.

При розробці майбутнього ігрового застосунку архітектура ЕС, оскільки вони забезпечує модульність і не накладає таких обмежень при розробці як ECS.

Також при розробці будуть використовуватися такі патерни як Singleton, State, Composition Over Inheritance.

Патерн Singleton є одним з найбільш популярних патернів програмування в ігровій розробці. Він використовується для забезпечення того, щоб у програмі було лише один екземпляр певного класу, і до цього екземпляру можна було звертатися з будь-якого місця програми.

У ігровій розробці патерн Singleton може бути корисним для створення глобальних об'єктів, таких як менеджери ресурсів, головний персонаж гравця або контролери гри. Наприклад, можна створити клас, який буде відповідати за управління станом гри, зберігання даних гри та інші подібні функції. Щоб забезпечити те, що програма буде працювати з лише одним екземпляром цього класу, необхідно зробити конструктор приватним та створити метод, який буде повертати єдиний екземпляр класу.

Проте при використанні патерну Singleton можливі проблеми зі зберіганням стану гри та його відновленням у випадку збоїв. Також потрібно бути обережними з використанням Singleton у великих проектах, де може бути багато залежностей між класами, оскільки це може призвести до складності розуміння та тестування коду.

Патерн State є корисним для розробки складних ігрових систем, які мають багато станів і переходів між ними. В основі патерну State лежить ідея, що об'єкт може змінювати свій стан в залежності від зовнішніх подій або дій користувача. При цьому кожен стан повинен мати свою власну поведінку та можливі переходи до інших станів.

У ігровій розробці патерн State може бути використаний для управління різними станами гри, такими як меню, гра, пауза, екран кінця гри та інших об’єктів гри. Кожен стан може мати свої власні методи та властивості, які відповідають за його функціональність. Наприклад, перший клас з даним патерном може мати методи для відображення меню та обробки вибраних елементів, а другий клас може мати методи для керування рухом персонажа та обробки взаємодії з ігровими об'єктами.

Для реалізації патерну State можна створити абстрактний клас State, який містить основні методи та властивості, що використовуються в кожному стані. Крім того, можна створити окремі класи для кожного стану, які будуть наслідувати від класу State та реалізовувати свою власну логіку.

Один з прикладів використання патерну State в ігровій розробці – реалізація станів персонажа у платформері. Прикладом може бути клас, який містить всі можливі стани персонажа: крокування, біг, присідання, стрибок та інші. Кожен з цих станів може мати свої методи та властивості, які відповідають за поведінку персонажа у даному стані.

У ігровій розробці патерн Composition Over Inheritance дозволяє створювати складні об'єкти, з різними функціях, шляхом комбінування компонентів замість успадкування властивостей батьківських класів.

Наприклад, при розробці ігрового застосунку з різними типами ворогів, то замість створення окремого класу для кожного типу ворога з успадкуванням властивостей від базового класу ворога, розробник може створити компоненти, які представляють різні властивості ворога, наприклад, компонент очків здоров’я, пошкодження, руху, і тому подібне. Далі, розробники можуть додавати ці компоненти до об'єктів ворогів у редакторі, щоб створювати різні типи ворогів, комбінуючи різні компоненти відповідно до потреб гри.

Даний підхід дозволяє створювати більш гнучкі та масштабовані системи, оскільки він дозволяє динамічно додавати та видаляти компоненти з об'єктів в процесі редагування гри у інспекторі ігрового рушія. Крім того, це сприяє повторному використанню коду та зменшенню зв'язності між класами, що робить код більш чистим та легким для змін.

Наприклад, в Unity, підхід Composition Over Inheritance можна застосувати при створенні об’єкта Game Object, який складається з різних готових компонентів з даними таких як Transform, Rigidbody, Collider, Script Components, тощо, замість успадкування властивостей від базового класу.

2.2 Проектування інтерфейсу користувача

Проектування інтерфейсу є важливим етапом розробки відеогри. Даний етап включає в себе багато різних аспектів.

Першим аспектом є користувацький досвід. Інтерфейс визначає, як гравці будуть взаємодіяти з грою. Його якість безпосередньо впливає на задоволення гравців та їхній загальний користувацький досвід. Грати в гру зручно та інтуїтивно зрозуміло допомагає гравцям більш повно насолоджуватися її геймплеєм.

Наступним є інформаційний потік. Інтерфейс відображає різноманітну інформацію гравцям, таку як статус героя, рівень здоров'я, запаси або карта гри. Його правильне проектування дозволяє забезпечити зручну та швидку передачу цієї інформації, не заважаючи гравцям головному екрану гри.

Інтерфейс дозволяє гравцям виконувати різні функції, такі як управління персонажем, вибір зброї, взаємодія з оточенням, виконання завдань тощо. Його добре продумане проектування допомагає забезпечити ефективність та зручність виконання цих дій, підвищуючи геймплейну якість гри.

Інтерфейс також впливає на візуальну привабливість гри. Дизайн інтерфейсу має бути згуртованим з загальним стилем гри, щоб створити захоплюючу та іммерсивну атмосферу. Це допомагає створити єдиний візуальний світ, в якому гравці будуть повністю занурені.

Усі ці аспекти роблять проектування інтерфейсу необхідним етапом розробки відеогри, який впливає на загальний успіх та прийняття гри гравцями.

За допомогою веб-сервісу Figma було спроектовано шаблони інтерфейсу відеогри. Даний сервіс дозволяє створювати також прототип інтерфейсу з переходами між макетами, а також отримувати відстані між об’єктами.

Figma є веб-сервісом для дизайну і прототипування інтерфейсів користувача. Він набув популярності серед дизайнерів та команд розробки,

оскільки надає потужні можливості для спільної роботи, простоти використання та інтуїтивного інтерфейсу.

Figma дозволяє дизайнерам та командам працювати над проектами одночасно та спільно. Користувачі можуть спостерігати за змінами, вносити власні правки та коментувати роботу інших учасників в реальному часі.

Даний веб-сервіс працює в браузері, що дозволяє легко доступатися до проектів з будь-якого комп'ютера без необхідності встановлення додаткового програмного забезпечення. Він також підтримує різні операційні системи, включаючи Windows, macOS і Linux.

Також він має потужні інструменти для створення векторного графічного контенту, таких як фігури, шляхи, текст та інші елементи дизайну. Він дозволяє розташовувати, масштабувати, редагувати та стилізувати елементи для створення зручного та привабливого інтерфейсу.

Окрім цього, є можливість створювати та використовувати бібліотеки компонентів, що спрощує повторне використання елементів дизайну та забезпечує єдність стилю у всьому проекті. Зміни, внесені в компоненти, автоматично оновлюють всі використовувані екземпляри цих компонентів.

Прототипування та анімація дозволяє створювати інтерактивні прототипи, що демонструють взаємодію елементів дизайну. Користувачі можуть створювати переходи між сторінками, додавати анімацію та встановлювати взаємодію елементів на основі різних подій.

Figma надає зручні інструменти для збереження та керування проектами. Користувачі можуть легко ділитися проектами з колегами, клієнтами та іншими зацікавленими сторонами, давати права доступу та коментувати роботу.

Інтеграція з іншими інструментами у даному сервісі дозволяє підтримувати інтеграцію з різними інструментами для розробки, такими як Jira, Slack, Zeplin, та інші. Це полегшує спільну роботу з командою розробки та передачу дизайну в процес реалізації.

На рисунку 2.1 зображено макет головної сторінки відеогри. Сірим прямокутником зображено назву відеогри, а чорними прямокутниками – кнопки які здійснюють перехід на інші екрани.

Рисунок 2.1 – Макет головного меню

На рисунку 2.2 зображено список ігрових рівнів відеогри. Як і в минулому макеті чорними прямокутниками – кнопки які здійснюють перехід на інші екрани, а саме перехід на інші рівні відеогри. На рисунку 2.3 зображено макет, яке буде висвітлюватися при завершенні рівня відеогри.

Рисунок 2.2 – Макет вибору рівнів меню

Рисунок 2.3 – Макет завершення проходження рівня

Як і в минулому макеті чорними прямокутниками – кнопки які здійснюють перехід на інші екрани. Одна з кнопок відповідатиме за вихід в меню, наступна за повтор проходження даного рівня і третя за перехід до наступного рівня. На рисунку 2.4 зображено макет фінальної головоломки для гравця.

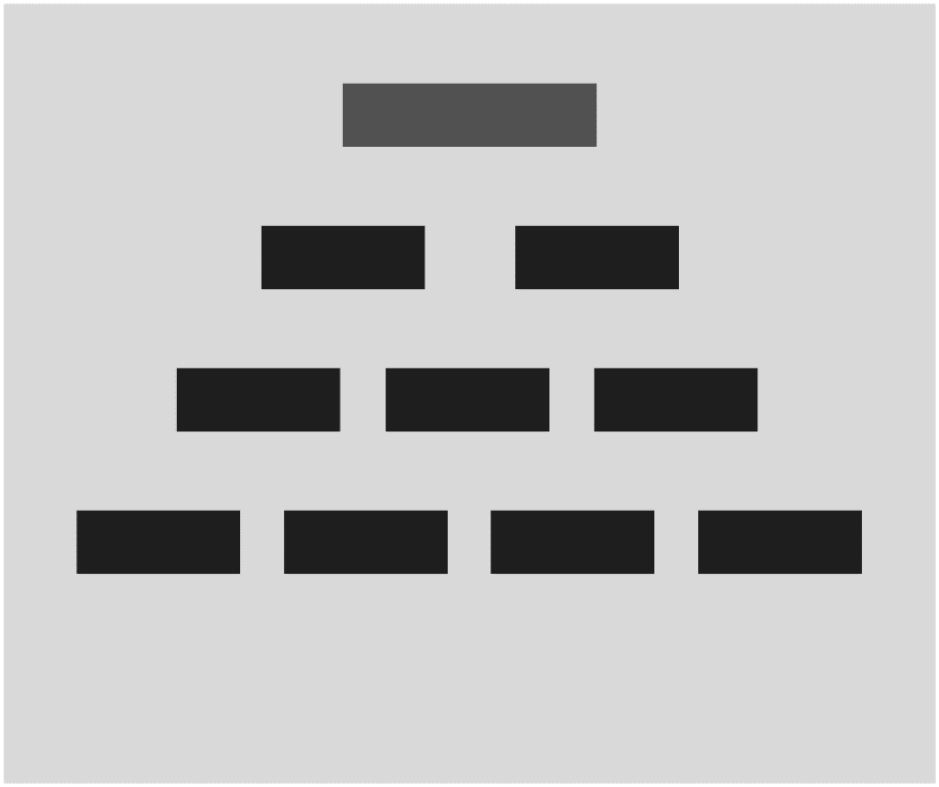


Рисунок 2.4 – Макет головоломки в кінці рівня

Головоломка, яка створена на основі даного макета відкривається під час ігрового процесу і займає лише певну частину ігрового простору на екрані. Чорними прямокутниками на даному макеті позначено кнопки. При завершені проходження головоломки вікно закривається.

2.3 Розробка алгоритму роботи ігрового застосунку

Для проектування алгоритму роботи ігрового застосунку була використана діаграма IDEF0. Даною діаграмою є графічний метод, який використовується для моделювання функцій та процесів у системах. IDEF0 діаграми використовуються для аналізу, проектування та вдосконалення бізнес-процесів, систем управління та інших складних систем.

У даних діаграмах використовуються позначки, які представляють різні елементи системи, такі як функції, процеси, вхідні та вихідні дані, ресурси та контроль. Ці символи з'єднані стрілками, які показують залежності між ними та порядок виконання дій.

IDEF0 діаграми є корисним інструментом для розуміння того, як працює система та її складові частини. Вони допомагають виявляти проблеми в системі та розробляти стратегії для їх вирішення.

На рисунку 2.5 зображено контекстну діаграму проходження рівня. Вхідними даними для проходження рівня є початкові позиції всіх об’єктів на сцені і об’єкт головного персонажа. Також вхідними даними є умови при дотриманні яких гравець зможе пройти рівень і схема керування головним персонажем. Керування і маніпуляції здійснює гравець і при виконанні всіх умов і задач рівень буде вважатися пройденим.

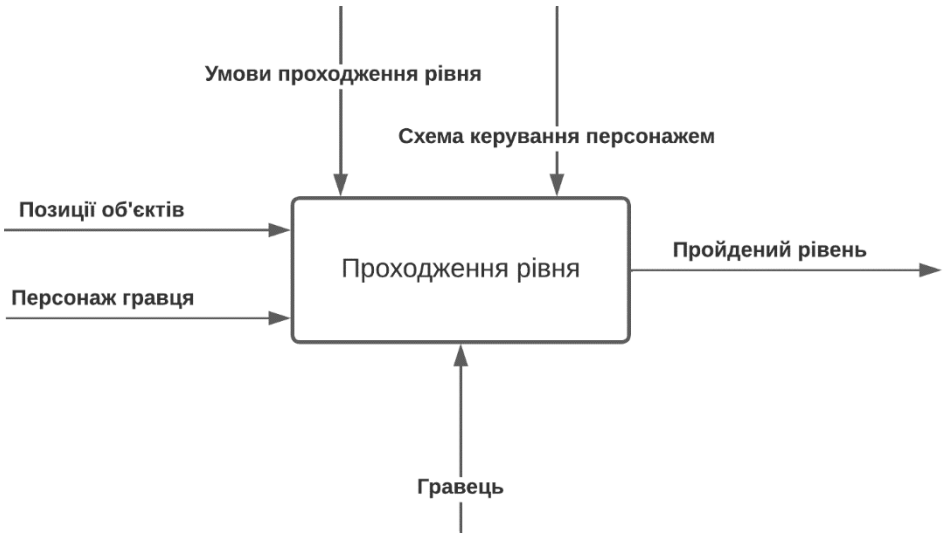


Рисунок 2.5 – Контекстна діаграма IDEF0 алгоритму проходження рівня

На рисунку 2.6 зображено діаграму декомпозицію контекстної діаграми. Як видно з рисунку при декомпозиції виділено чотири блоки. Першим блоком є запуск рівня під час якого розставляються об’єкти на рівні.

Наступними двома блоками є пошук предметів і вирішення додаткових завдань. При пошуку необхідних предметів гравець керує персонажем за допомогою приладу введення і схеми керування персонажем. Третім блоком є вирішення додаткових завдання на рівні.

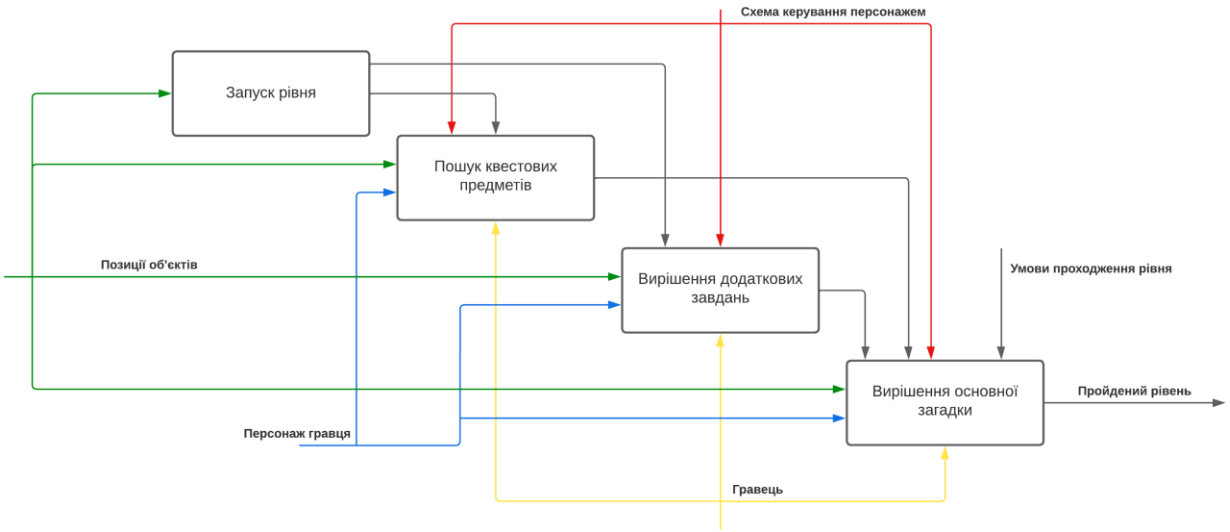


Рисунок 2.6 – Діаграма декомпозиції IDEF0 алгоритму проходження рівня

Гравець керує своїм персонажем і спочатку шукає додаткові завдання на рівні, а потім вже їх вирішує. Обидва блоки після їх виконання приводять гравця до останнього блоку, а саме вирішення основної загадки рівня.

Для того, щоб гравець мав можливість розпочати вирішувати загадку, він повинен виконати попередні умови, які будуть встановлені. Щоб успішно пройти рівень, гравець має сконцентруватись на графічному інтерфейсі, що відображається на екрані, і використовувати квестові предмети, які будуть доступні для нього. У кожному рівні присутня загадка, яка залишається незмінною. Щоб продовжити до наступного етапу гри, гравець повинен успішно розв'язати цю загадку. Після того, як всі умови проходження рівня будуть виконані, гравець завершує його і може перейти до наступного рівня або продовжити свою пригоду в грі. На рисунку 2.7 зображена діаграма варіантів використання.

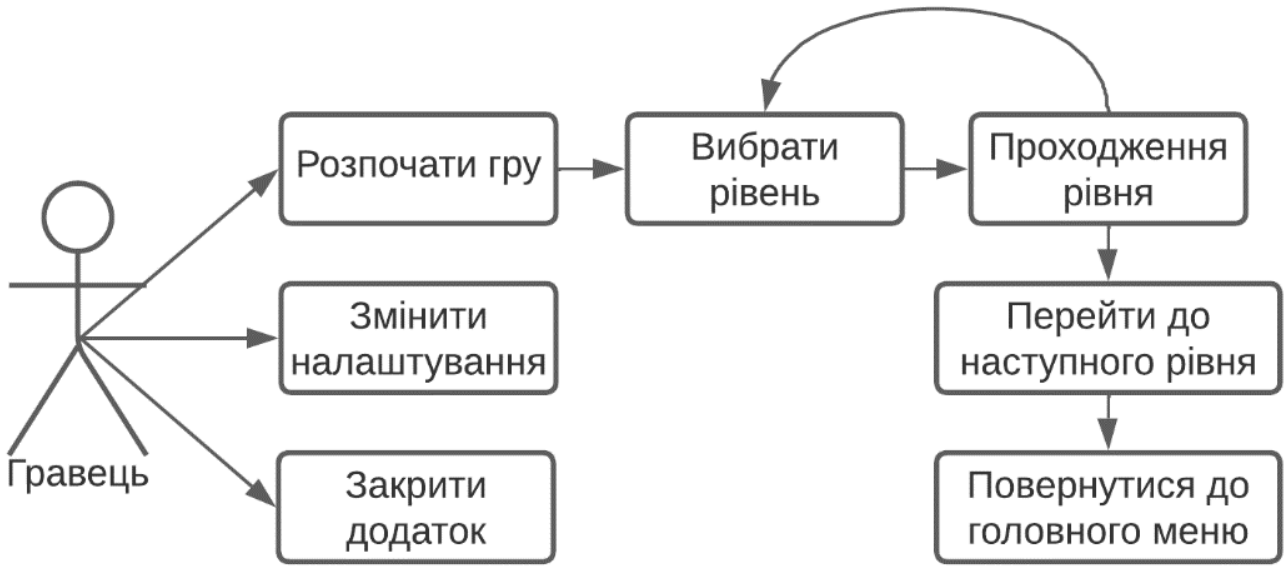


Рисунок 2.7 – Діаграма варіантів використання

При запуску ігрового застосунку користувачу буде доступний вибір з трьох пунктів. Він зможе вийти з застосунку, змінити налаштування ігрового застосунку або розпочати гру. Якщо користувач вирішить почати грати, йому потрібно буде обрати рівень.

Після вибору рівня він буде відкритий, і користувач матиме можливість пройти його і перейти до наступного рівня, або повернутися до вибору рівнів .

2.4 Аналіз та вибір технологій і методів реалізації застосунку

Unity – це багатоплатформовий інструмент для розробки відеоігор, застосунків, анімацій і анімованих фільмів, а також рушій на якому вони працюють. Створені у Unity застосунки запускаються на платформах ж операційними системами Windows, Android, Linux, iOS та MacOS, а також розробник може створити за допомогою Unity застосунку для ігрових приставок PlayStation, Xbox, Nintendo та для смарт приставок для телевізорів. Створені

застосунку підтримують такі технології як DirectX та OpenGL і можуть бути запущеними у браузері.

Головним елементом Unity є його редактор, а саме Unity Editor. Він є графічним інструментом розробки, який використовується для створення відеоігор. Графічний редактор надає можливість взаємодії зі своїм проектом в режимі реального часу, візуально налаштовувати графічний вигляд, фізику, анімацію, звуки та інші елементи гри без необхідності писати код. Зовнішній вигляд графічного редактора зображено на рисунку 2.8

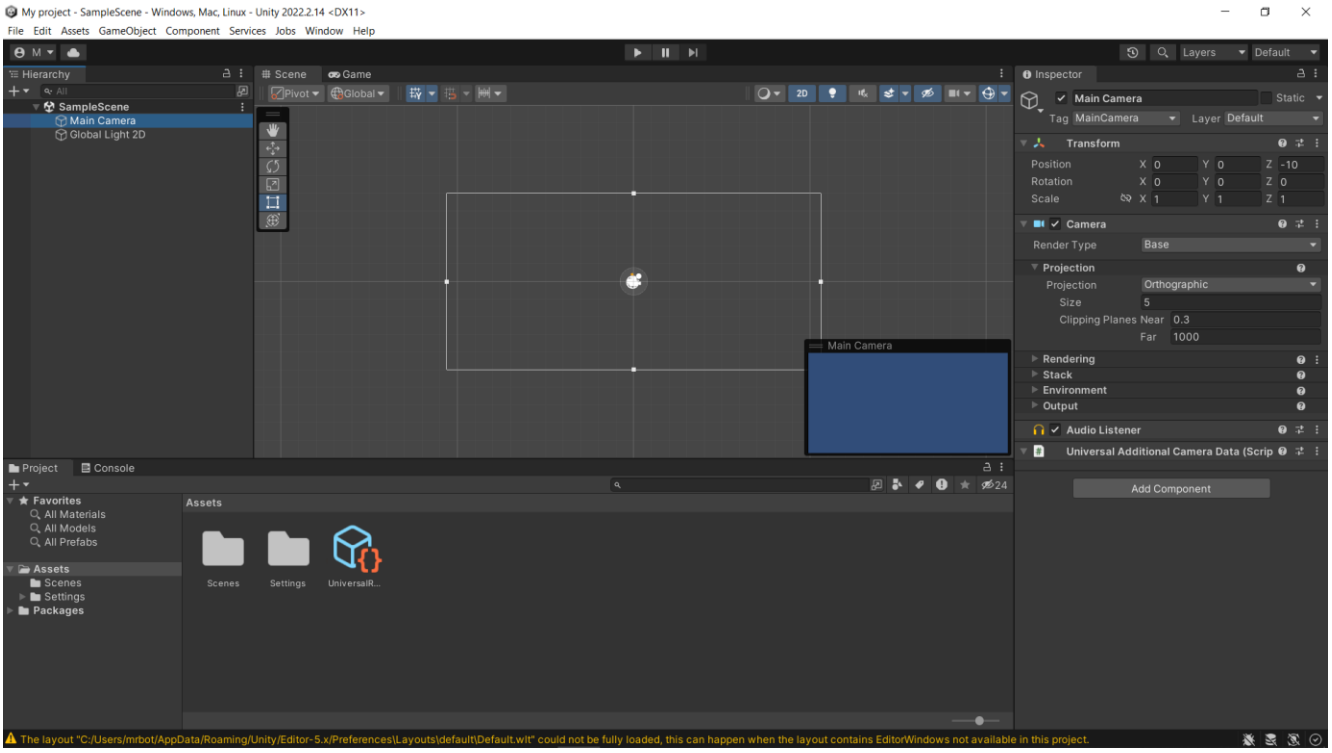


Рисунок 2.8 – Зовнішній вигляд редактора Unity

В даному графічному редакторі розробники можуть створювати ту частину відеогри, яка не потребує написання коду, а саме створення анімацій, добавляння елементів фізики і колізій на об’єкти, налаштування звуку і тому подібне. В даному графічному редакторі розробники можуть оптимізувати відеогру наскільки це можливо шляхом розподілення ресурсів, визначенням поведінки об’єктів і налаштуванням графіки.

Для написання скриптів Unity використовує об’єктно-орієнтовану мову програмування C#. Дані скрипти керують поведінкою ігрових об’єктів та

реалізують логіку гри. У Unity є свої бібліотеки, фреймворки для C#, а також спільнота додає до ігрового рушія свої готові рішення і бібліотеки, які є корисними для тих чи інших проектів.

Розробники мають змогу прямо з графічного редактора відкривати Unity Asset Store, де знаходиться велика кількість різноманітних елементів для розробки відеоігор. В Asset Store розробники можуть безкоштовно або за гроші отримати готові моделі, анімації, звук, графічні елементи, інтерфейс та інші ігрові компоненти. Це дозволяє пришвидшити процес розробки і зекономити ресурси для розробників відеоігор. Також це є площадка для реалізації своїх різноманітних продуктів, що дозволяє або заробляти, або поповнювати бюджет свого проекту шляхом продажі елементів своєї гри.

Unity має вбудовану фізичну систему, яка дозволяє створювати реалістичну фізичну поведінку у об’єктів, колізії та взаємодії між ними. Розробники самі можуть змінювати параметри і налаштовувати фізичну систему для свого проекту. Також у Unity є вбудована система створення анімацій для двовимірних і тривимірних об’єктів. Проте Unity так само сумісна і з анімаціями, які були створені в інших програмних продуктах.

Розробники можуть використовувати у графічному редакторі візуальну мову програмування Bolt. Дана технологія дозволяє розробникам створювати логіку гри без використання традиційного текстового кодування, а замість цього використовувати графічний інтерфейс з блоками та з'єднаннями між ними.

Bolt має велику бібліотеку вбудованих блоків, які включають різні операції, такі як математика, логіка, робота зі змінними, робота з фізикою, анімація та багато іншого. Це дозволяє розробникам ефективно створювати різноманітну логіку гри без необхідності писати код вручну.

Він має відкритий API, що дозволяє розробникам створювати власні блоки та функції, розширюючи можливості системи. Це дає можливість налаштовувати Bolt під власні потреби та розробляти власні розширення для конкретних проектів. Також він має ряд розширених можливостей, таких як можливість використовувати власні скрипти на C# у візуальному середовищі, робота з

векторами, матрицями та іншими складними типами даних, підтримка зовнішніх бібліотек та API, інтеграція з системами контролю версій та багато іншого. Це дозволяє розробникам розширювати можливості Bolt та створювати складні логічні конструкції в своїх проектах.

Проте у даної технології є ряд недоліків і головний з них це обмеженість у порівнянні із традиційним методом програмування. Розробникам важче або неможливо реалізувати складні логічні конструкції за допомогою Bolt. Також Bolt є відносно новою технологією, тому може бути менше доступної документації, ресурсів та підтримки порівняно зі стандартним C# програмуванням в Unity. Це може зробити процес навчання складнішим для деяких розробників.

При використанні візуальної технології програмування можливі деякі втрати продуктивності, наприклад, як витрачення більше часу на створення складних логічних конструкцій або збільшення обсягу роботи при великому проекті. Деякі розробники можуть віддавати перевагу писанню коду вручну, щоб досягти кращої продуктивності.

Компанія власник Unity використовує просунути систему монетизації свого продукту, а саме компанія надає безкоштовну версію із простим функціоналом і з покращеним функціоналом по підписці. Окрім цього розробники залишають весь прибуток собі, якщо їх місячний дохід менший за суму, яка вказана у правилах користуванням ігровим рушієм.

Головним конкурентом Unity є Unreal Engine. Unreal Engine – це популярний ігровий рушій, який був розроблений компанією Epic Games. Він використовується для розробки високоякісних відеоігор, віртуальної реальності, анімації, а також для створення інтерактивних візуальних застосунків, симуляцій, тренажерів та інших проектів.

Unreal Engine має високоякісний графічний рушій, який дозволяє створювати візуально захоплюючі ігри з реалістичною графікою, включаючи підтримку різних ефектів освітлення, тіней, води, частинок та інших візуальних ефектів. Unreal Engine надає величезний функціонал для розробки відеоігор, такий як вбудовані редактори для створення рівнів, штучного інтелекту, фізики,

анімації, звуку, розвитку ігрових персонажів та багато іншого. Він також має розширювану систему плагінів, що дозволяє розробникам розширювати функціональність рушію за своїми потребами. Зовнішній вигляд головного редактора Unreal Engine зображено на рисунку 2.9

Рисунок 2.9 – Зовнішній вигляд головного редактора Unreal Engine

Даний ігровий рушій використовує мови програмування C++ та Blueprints. C++ надає потужні можливості для розробки високопродуктивних ігрових функцій, тоді як Blueprints – це візуальна система програмування, яка дозволяє створювати логіку гри за допомогою візуального інтерфейсу без необхідності писати код. Дана технологія є аналогом технології Bolt. Unreal Engine так само як і Unity підтримує кросплатформену розробку.

Проте Unreal Engine є досить складним рушієм, особливо для початківців, які не мають досвіду в розробці відеоігор або в програмуванні. Вивчення C++ та інших інструментів Unreal Engine може вимагати часу та зусиль, що може становити виклик для новачків. Порівняння ігрових рушіїв по основним параметрам наведено в таблиці 2.1.

Таблиця 2.1 – Таблиця порівняння основних параметрів ігрових рушіїв

Назва рушія	Unity	Unreal Engine
Мова програмування	C#	C++
Роялті	При досягненні певної суми доходів від продуктів необхідно купити ліцензію	Певний відсоток від прибутку з продаж продукту
Складність опановування	Середня	Висока
Наявність Asset Store	Так	Так
Наявність інтегрованого засобу створення анімацій	Так	Так
Наявність візуальної мови програмування	Так	Так
Наявність кросплатформеності	Так	Так, але обмежено
Високі вимоги щодо апаратного забезпечення розробників	Ні	Так

Розробка в Unreal Engine може вимагати потужних комп'ютерних систем з високопродуктивними графічними картами, процесорами та великим обсягом оперативної пам'яті. Це може бути вартісним обмеженням для деяких розробників з обмеженим бюджетом або менш потужним обладнанням.

Окрім цього, відеоігри на Unreal Engine мають деякі проблеми під час ігрового процесу на платформах операційна система яких відрізняється від Windows.

Хоча Unreal Engine безкоштовний для використання в особистих проектах, комерційна розробка вимагає певних ліцензійних витрат, зокрема відсотка від

прибутку від продажу гри або її роялті. Це може вплинути на бюджет розробника, зокрема в малих студіях або самостійних розробниках.

Загалом порівняння, двох найбільш релевантних для розробки відеоігор, ігрових рушіїв для можна здійснити за допомогою таблиці.

Як видно з таблиці Unreal Engine ставить перед розробникам вищі ризики і вимоги. Загалом вибір між Unity та Unreal Engine залежить від потреб та вимог майбутнього проекту. Unity є більш підходящим для простих проектів, початківців або розробки для різних платформ, в той час як Unreal Engine може бути кращим варіантом для високоякісних 3D-графічних проектів з великою кількістю розробників у команді.

2.5 Висновки

В даному розділі були розглянуті та проаналізовані дві архітектури для майбутнього ігрового застосунку, а саме MVC та ECS. Також були розглянуті, проаналізовані та описані необхідні патерни проектування, які будуть використані у майбутньому програмному рішенні.

Для ігрового застосунку були спроектовані макети інтерфейсу. Дані макети будуть використані при розробці відеогри. Також були розроблені діаграми, а саме контекстна діаграма та діаграма декомпозиції стандарту IDEF0. Також була створена діаграма варіантів використання.

Було здійснено аналіз переваг та недоліків двох ігрових рушіїв, на яких можна розробляти ігровий застосунок. Даними рушіями були Unity та Unreal Engine. Було проаналізовано їх інструменти, переваги та недоліки. На основі аналізу була сформована таблиця порівняння і було сформовано пояснення переваги одного рушія над іншим для даної предметної області кваліфікаційної роботи. Також були проаналізовані інші програмні засоби, які будуть використовуватися при розробці відеогри.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ

3.1 Реалізація логіки застосунку

Процес розробки ігрового застосунку описаний за допомогою діаграми IDEF0. Першою діаграмою є контекстна діаграма. Вхідними даними для реалізації застосунку є завдання на розробку. Також вхідними даними є умови, а саме технічне завдання, при дотриманні яких можна вважати ігровий застосунок завершеним. До розробки ігрового застосунку залучені розробник, художник та тестувальник. На рисунку 3.1 зображено контекстну діаграму IDEF0 процесу реалізації застосунку.

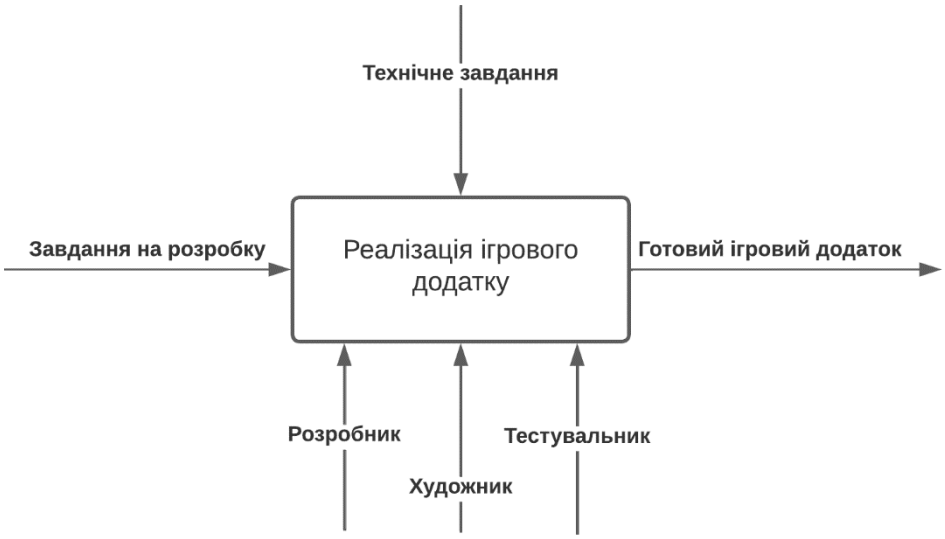


Рисунок 3.1 – Контекстна діаграма IDEF0 процесу реалізації застосунку

На рисунку 3.2 зображено діаграму декомпозицію контекстної діаграми на попередньому рисунку. Як видно з рисунку при декомпозиції виділено чотири блоки. Першим блоком є реалізація логічної частини ігрового застосунку розробником. Наступними блоком є реалізація графічної частини застосунку художником. Обидва перших процеси виконуються паралельно один одному Третім блоком є об’єднання результатів роботи при попередніх двох процесах. Кінцевим процесом на даній діаграмі є тестування ігрового застосунку.

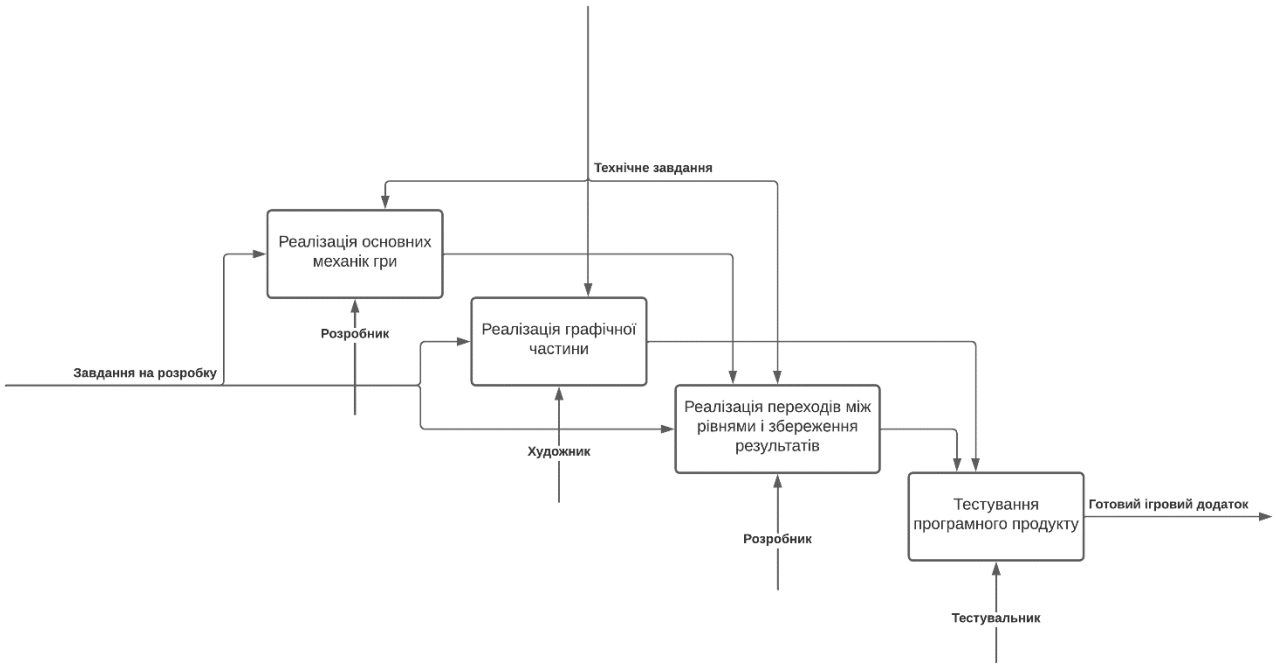


Рисунок 3.2 – Діаграма декомпозиції IDEF0 процесу реалізації застосунку

При розробці ігрового застосунку була використана нова система введення Unity. Нова система введення (New Input System) у Unity є потужним і багатофункціональним інструментом для обробки вводу в ігрових проектах. Ця система була розроблена з метою полегшити процес розробки та управління введенням у грі, забезпечуючи більш гнучкий та продуктивний підхід до обробки вводу користувача.

New Input System надає багато функцій для обробки різноманітного вводу, включаючи клавіатуру, мишу, геймпади, джойстики, сенсорні екрани, віртуальні реальність та багато іншого. Це дозволяє розробникам легко працювати з різноманітними типами вводу без потреби в кастомній логіці для кожного пристрою окремо.

В новій системі введення події використовуються для визначення введення в грі, наприклад, натискання кнопки, рух курсору тощо. Акції дають можливість об'єднувати декілька подій в одну логічну дію, що спрощує обробку складних комбінацій вводу.

New Input System підтримує багато платформ, включаючи Windows, macOS, Linux, Android, iOS, консолі Xbox та PlayStation, а також різноманітні віртуальні та доповнені реальності. Це дозволяє розробникам легко імпортувати свої ігри на різні платформи без необхідності великих змін у коді.

New Input System має зручний інструментарій для налаштування введення, що дозволяє розробникам змінювати параметри вводу, такі як чутливість, діапазон значень, фільтрацію та інші налаштування, в зручний спосіб.

New Input System інтегрується з Unity Editor, що дозволяє розробникам тестувати введення безпосередньо у редакторі, спрощуючи процес розробки та налагодження.

Дана система надає розробнику багато інструментів та можливостей для зручного та потужного керування введенням у проекті. На рисунку 3.3. зображено вікно для налаштування схеми керування головним персонажем.

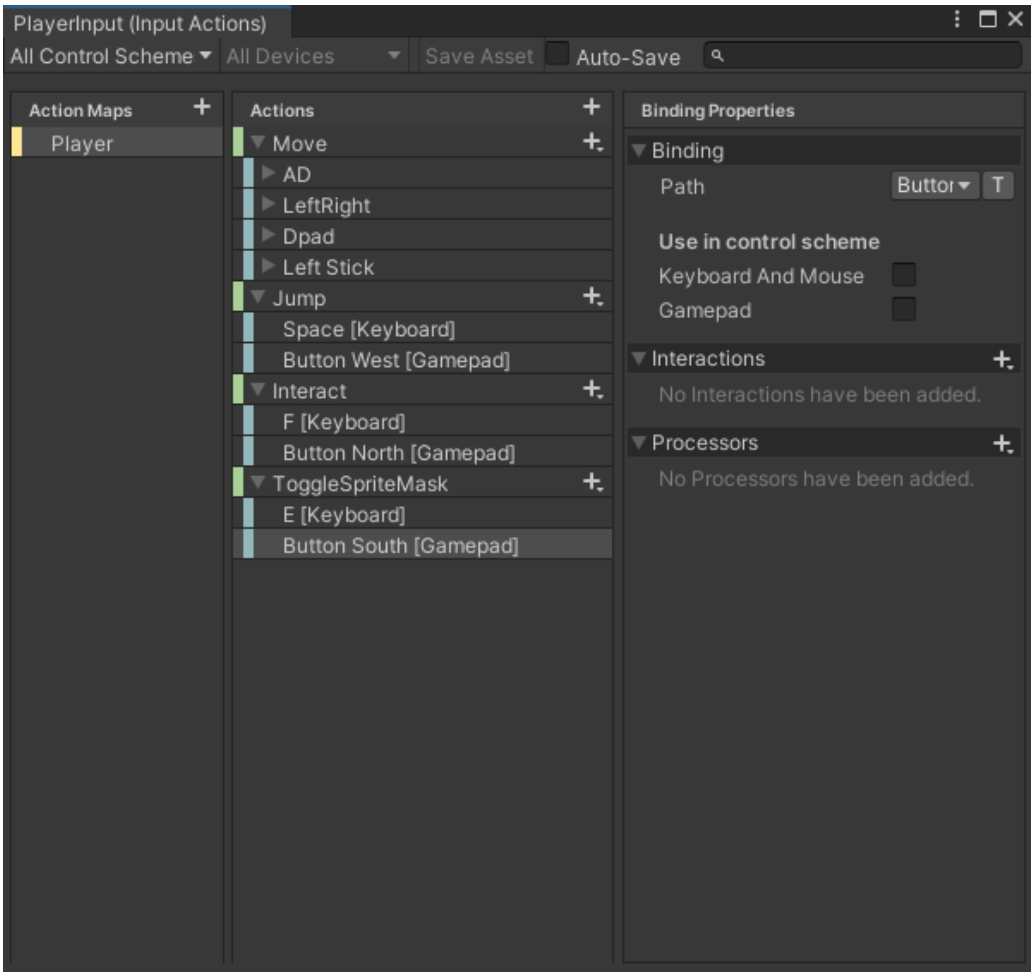


Рисунок 3.3 – Вікно для налаштування схеми керування головним персонажем гри

Дане вікно спрощує розробку, особливо для великих проектів, де використовується велика кількість взаємодій гравця з відеогру через прилади введення. Як видно з рисунку було виділено чотири основних функцій для взаємодії з приладами введення. Всі функції мають можливість бути виконаними як з клавіатури, так і з ігрового контролера.

Три останніх функції спрацьовують лише при натисканні на відповідну клавішу гравцем і в коді відеогри відбуваються прив'язані до цієї функції методи. Функція Move спрацьовує при натисканні однієї з двох клавіш і повертає в код двовимірний вектор. Також вона має набагато більше резервних клавіш для її активації.

Ігровий рушій Unity має свій основний клас – MonoBehaviour. Даний клас є одним з основних класів у Unity, який дозволяє створювати скрипти, що прикріплюються до об'єктів у сцені.

Він є базовим класом для реалізації логіки поведінки об'єктів у відеогрі. MonoBehaviour містить набір функцій подій, які викликаються автоматично в різні моменти життєвого циклу об'єкта. Основні функції подій виконуються по порядку.

Метод Awake викликається один раз при створенні об'єкта та встановлює початкові параметри. Даний метод викликається перед будь-якими іншими функціями подій.

Метод OnEnable викликається, коли компонент стає активним. Він викликається після методу Awake та кожного разу, коли компонент активується після вимкнення.

Метод Start викликається після Awake та OnEnable і виконується один раз перед першим оновленням кадру. Він використовується для початкових налаштування об'єктів або початку процесів. Метод FixedUpdate викликається з фіксованим інтервалом часу і використовується для обробки фізики. Він виконується певну кількість разів на секунду, яку розробник може налаштувати в параметрах фізики проекту.

Метод Update викликається кожен кадр та використовується для оновлення стану об'єктів та логіки гри. Він виконується після FixedUpdate.

Метод LateUpdate викликається після всіх викликів методу Update і використовується для вирішення останніх змін у стані об'єктів після оновлення кадру. Він корисний для керування камерою та іншими об'єктами, які залежать від позицій інших об'єктів.

Метод OnDisable викликається, коли компонент вимикається. Він викликається перед тим, як компонент стає неактивним.

Метод OnDestroy викликається, коли об'єкт або компонент знищується. Він викликається перед тим, як об'єкт більше не існує.

Цей порядок виконання функцій подій застосовується до скриптів, що прикріплені до об'єктів в сцені Unity. Кожен метод подій має своє призначення та виконується відповідно до життєвого циклу об'єктів.

Першим класом для опису є клас PlayerMovement. Даний клас є прикріпленим до головного героя і в його зоні відповідповідальності знаходяться всі дії пов’язані з рухом головного героя. На початку класу оголошуються поля класу в основному з модифікаторами доступу private. Функція SerializeField використовуються для відображення у інспектора, а також для зміни значень без використання для цього середовища розробки. Окрім використання стандартних класів Unity, також даний клас використовує класи AudioManager, який здійснює запуск аудіо файлів, і GroundSensor, який перевіряє положення тіла персонажа відносно поверхні. Клас Rigidbody2D є стандартним і дозволяє керувати поведінкою фізики об’єкта до якого він прикріплений. Серед полів також є і аніматор, який відповідає за запуск і зміну анімацій, а також базові типи даних.

Перші чотири методи виконуються лише один раз самим ігровим рушієм при запуску скрипта. Першим виконується метод Awake, в якому здійснюється підписка на події нової системи введення. В методі Start ініціалізуються поля класу. Метод OnEnable та OnDisable здійснюють увімкнення і вимкнення нової системи введення при включенні і виключенні скрипта відповідно під час ігрового процесу.

На рисунку 3.4 Зображено першу частину класу PlayerMovement.

```
4   public class PlayerMovement : MonoBehaviour
5   {
6       [Header("Variables")]
7       [SerializeField] float _jumpForce = 7.5f;
8       [SerializeField] float _moveSpeed = 5f;
9
10      private PlayerInput _input;
11      private Animator _animator;
12      private Rigidbody2D _rigidbody;
13      private GroundSensor _groundSensor;
14      private AudioSource _audioSource;
15      private AudioManager _audioManager;
16      private bool _isGrounded = false;
17      public bool permissionToMove = true;
18      private float _moveDirection;
19
20
21      private void Awake()
22      {
23          _input = new PlayerInput();
24          _input.Player.Jump.performed += context => Jump(_jumpForce);
25      }
26
27      private void OnEnable()
28      {
29          _input.Enable();
30      }
31
32      private void OnDisable()
33      {
34          _input.Enable();
35      }
36
37      private void Start()
38      {
39          _rigidbody = GetComponent<Rigidbody2D>();
40          _animator = GetComponent<Animator>();
41          _audioSource = GetComponent<AudioSource>();
42          _audioManager = AudioManager.instance;
43          _groundSensor = transform.Find("GroundSensor").GetComponent<GroundSensor>();
44      }
```

Рисунок 3.4 – Перша частина класу PlayerMovement

На рисунку 3.5 зображено другу частину класу PlayerMovement. На даному рисунку зображено метод Update класу PlayerMovement. Даний метод виконується ігровим рушієм кожен кадр. В даному методі здійснюється перевірка положення об'єкта персонажа за допомогою методів класу GroundSensor відносно поверхні, здійснюється зміна значень аніматора і виконання Animation Event, які в даному класі здійснюють запуск внутрішньо ігрових звуків. Також з даного метода здійснюється виклик методу Move, який здійснює рух персонажа, і передається вектор руху.

```
46      private void Update()
47      {
48          if (!_isGrounded && _groundSensor.State())
49          {
50              _isGrounded = true;
51              _animator.SetBool("Grounded", _isGrounded);
52          }
53
54          if (_isGrounded && !_groundSensor.State())
55          {
56              _isGrounded = false;
57              _animator.SetBool("Grounded", _isGrounded);
58          }
59
60          _animator.SetFloat("VerticalVelocity", _rigidbody.velocity.y);
61
62          _moveDirection = _input.Player.Move.ReadValue<float>();
63
64          if(_moveDirection!=0)
65              Move(_moveDirection);
66          else
67              _animator.SetInteger("AnimState", 0);
68
69
70          void AE_RunStop()
71          {
72              _audioManager.PlaySound("RunStop");
73          }
74
75          void AE_Footstep()
76          {
77              _audioManager.PlaySound("Run");
78          }
79
80          void AE_Jump()
81          {
82              _audioManager.PlaySound("Jump");
83          }
84
85          void AE_Landing()
86          {
87              _audioManager.PlaySound("Landing");
88          }
89      }
```

Рисунок 3.5 – Друга частина класу PlayerMovement

На рисунку 3.6 зображено третю частину класу PlayerMovement. Метод Jump виконується при натисканні клавіші для стрибка. При виконанні даного методу здійснюється перевірка чи знаходиться персонаж на поверхні. Якщо перевірка проходить успішно, то метод запускає анімацію стрибка, змінює

значення поля класу і за допомогою методу класу Rigidbody2D надає об'єкту швидкість.

Метод Move виконується при натисканні клавіш для руху з методу Update. В даному випадку здійснюється множення напрямку руху на його швидкість на атрибут класу Time для того, аби швидкість персонажа не залежала від кількості кадрів за секунду. Рух персонажа здійснюється за допомогою його зміщення по координатам. Також в даному методі здійснюється зміна напрямку спрайта і одного зі значень аніматора.

```
91      private void Jump(float jumpForce)
92      {
93          if (_isGrounded)
94          {
95              _animator.SetTrigger("Jump");
96              _isGrounded = false;
97              _animator.SetBool("Grounded", _isGrounded);
98              _rigidbody.velocity = new Vector2(0, jumpForce);
99              _groundSensor.Disable(0.2f);
100         }
101     }
102
103     private void Move(float direction)
104     {
105         if (permissionToMove)
106         {
107             float scaledTransformOffset = direction * _moveSpeed * Time.deltaTime;
108             transform.position += new Vector3(scaledTransformOffset, 0f, 0f);
109
110             if (_moveDirection > 0)
111                 GetComponent<SpriteRenderer>().flipX = false;
112
113             else if (_moveDirection < 0)
114                 GetComponent<SpriteRenderer>().flipX = true;
115
116             _animator.SetInteger("AnimState", 1);
117
```

Рисунок 3.6 – Третя частина класу PlayerMovement

Наступним класом є клас Player, який також є прикріпленим до головного героя. Даний клас використовується для взаємодії з навколишніми об'єктами. На початку класу оголошуються його поля, три з яких мають функцію серіалізації для їх налаштування з інспектора. Даний клас також використовує три базових типи даних.

Клас LayerMask необхідний для взаємодії лише з певними об’єктами на ігровій сцені. Клас Puzzle інший написаний клас, який відповідає за логіку виконання фінальної головоломки. Клас SpriteMask необхідний для того аби взаємодіяти зі спрайтами, які є приховані. Клас AudioManager, який є синглтоном використовується для відтворення звуків взаємодії головного героя з навколишніми об’єктами. На рисунку 3.7 зображено першу частину класу Player, який є основним у розроблюваній відеогрі.

```
5   public class Player : MonoBehaviour
6   {
7       [Header("GameObjects")]
8       [SerializeField] private LayerMask _interactLayerMask;
9       [SerializeField] private Puzzle _puzzle;
10      [SerializeField] private SpriteMask _spriteMask;
11
12      private FuseBox _fuseBox;
13      private BrokenCable _brokenCable;
14      private PlayerInput _input;
15      private PlayerMovement _playerMovement;
16      private AudioManager _audioManager;
17      private bool _isPuzzleCompleted = false;
18      private int _fuseQuantity;
19      private int _fuseCount = 0;
20
21      private void Awake()
22      {
23          _input = new PlayerInput();
24          _input.Player.Interact.performed += context => Interact();
25          _input.Player.ToggleSpriteMask.performed += context => ToggleSpriteMask();
26      }
27
28      private void OnEnable()
29      {
30          _input.Enable();
31          _puzzle.puzzleCompleted += PuzzleCompleted;
32      }
33
34      private void OnDisable()
35      {
36          _input.Enable();
37          _puzzle.puzzleCompleted -= PuzzleCompleted;
38      }
39
40      private void Start()
41      {
42          _playerMovement = GetComponent<PlayerMovement>();
43          _audioManager = GameObject.FindObjectOfType<AudioManager>();
44          _fuseQuantity = GameObject.FindObjectsOfType<Fuse>().Length;
45      }
```

Рисунок 3.7 – Перша частина класу Player

Перші чотири методи є стандартними методами батьківського класу MonoBehaivour. В методі Awake відбувається підписка на дві функції схеми керування головним героєм. В методі OnEnable відбувається підписка на подію, яка відбувається у класі Puzzle. В методі Start здійснюється ініціалізація трьох полів даного класу. На рисунку 3.8 зображено другу частину класу Player.

```
47      private void Interact()
48      {
49          RaycastHit2D raycastHit = Physics2D.Raycast(transform.position, new Vector2(0, 2), 2f, _interactLayerMask);
50          if (raycastHit)
51          {
52              if (_fuseBox != null && !_isPuzzleCompleted && _fuseQuantity == _fuseCount
53                  && GameObject.FindObjectsOfType<BrokenCable>().Length == 0)
54              {
55                  _playerMovement.permissionToMove = !_playerMovement.permissionToMove;
56                  _fuseBox.TogglePuzzleCanvas();
57              }
58              if(_brokenCable != null)
59              {
60                  _brokenCable.FixCable();
61                  _audioManager.PlaySound("PickUp");
62              }
63          }
64      }
```

Рисунок 3.8 – Друга частина класу Player

Метод Interact спрацьовує при натисканні відповідної кнопки схеми керування. Він використовує клас Unity RaycastHit2D для знаходження об’єкта з колізією, який має значення LayerMask, таке ж саме яке було оголошено в даному класі в полі на початку. RaycastHit2D – це структура в Unity, яка містить інформацію про перетин променя з коллайдерами 2D об'єктів у сцені. Використовуючи RaycastHit2D, можна отримати доступ до різних даних про перетин, таких як точка перетину, нормаль до поверхні, відстань до перетину та інші. Основне використання RaycastHit2D пов'язане з взаємодією з фізичними об'єктами у двовимірній сцені. Також екземпляр даного класу може слугувати логічним значенням для логічного оператора.

Головний персонаж може взаємодіяти з двома типами об’єктів, а саме електричним щитком і пошкодженим кабелем. При взаємодії з електричним щитком відбувається перевірка чи була зібрана необхідна кількість запобіжників, чи були відремонтовані всі кабелі і чи не пройшов користувач вже фінальну головоломку. У випадку якщо проходить перевірка, то у користувача на екрані відкривається головоломка.

Якщо об’єктом для взаємодії є пошкоджений кабель, то здійснюється його ремонт за допомогою методів класу BrokenCable. Також відтворюється звук ремонту кабелю. На рисунку 3.9 зображено третю частину класу Player.

```
66      private void IncreaseFuseCount()
67      {
68          _fuseCount++;
69          Debug.Log(_fuseCount);
70      }
71
72      private void ToggleSpriteMask()
73      {
74          _spriteMask.enabled = !_spriteMask.enabled;
75      }
76
77      private void PuzzleCompleted()
78      {
79          _isPuzzleCompleted = true;
80          _playerMovement.permissionToMove = true;
81          _audioManager.PlaySound("FuseBox");
82      }
```

Рисунок 3.9 – Третя частина класу Player

Перший метод з рисунку збільшує лічильник підібраних запобіжників, який потім проходить перевірку при взаємодії зі щитком. Наступний метод включає маску, яка дозволяє бачити приховані об’єкти на сцені. Третій метод на рисунку виконується, якщо користувач пройшов фінальну головоломку і викликається за допомогою делегата з іншого класу.

На рисунку 3.10 зображено четверту частину класу Player. Метод, який зображений на рисунку є стандартним для Unity і виконується кожен кадр ігровим рушієм. Якщо колізія головного персонажа заходить в тригер, то спрацьовує внутрішня частина даного методу. Далі здійснюються три перевірки тригера на наявність в нього певного класу. В даному випадку клас Fuse слугує класом-маркером, що є особливістю архітектури Entity Component System і похідної від неї Entity Component. Якщо перевірка проходить успішно, до об’єкт з даним класом знищується, а лічильник запобіжників збільшується завдяки виклику метода IncreaseFuseCount.

```
84      private void OnTriggerEnter2D(Collider2D other)
85      {
86          if (other.gameObject.TryGetComponent(out Fuse fuse))
87          {
88              Destroy(other.gameObject);
89              IncreaseFuseCount();
90              _audioManager.PlaySound("PickUp");
91          }
92          if (other.gameObject.TryGetComponent(out FuseBox fuseBox))
93          {
94              if (fuseBox != null)
95                  _fuseBox = fuseBox;
96          }
97          if (other.gameObject.TryGetComponent(out BrokenCable brokenCable))
98          {
99              _brokenCable = brokenCable;
100         }
101         else
102         {
103             _brokenCable = null;
104         }
105     }
106 }
107
```

Рисунок 3.10 – Четверта частина класу Player

Наступна перевірка здійснюється на наявність у об’єкта класу FuseBox. Якщо він проходить перевірку, то клас Player має посилання на об’єкт щитка для подальшої взаємодії з ним.

Остання перевірка здійснюється на наявність у об’єкта класу BrokenCable. Дана перевірка необхідна для взаємодії з пошкодженим кабелем і його ремонтом.

Collider2D називають компонент в Unity, який дозволяє визначати області колізій для 2D об'єктів у сцені. Він використовується для взаємодії та знаходження колізій між різними об'єктами у двовимірному просторі. Collider2D встановлюється на графічний об'єкт та визначає його фізичну форму та розмір. Він може бути прямокутним, круглим, полігональним або іншими формами.

Основна функціональність Collider2D включає в себе знаходження колізій. Collider2D дозволяє виявляти колізії з іншими об'єктами, які мають Collider2D або інші колайдери, такі як Rigidbody2D. Також є можливість налаштувати різні режими детекції колізій, такі як зіткнення з краями колайдерів, колізії з певними шарами об'єктів тощо. Також Collider2D взаємодіє з іншими фізичними компонентами, наприклад, Rigidbody2D, щоб забезпечити реалістичну поведінку

об'єктів у сцені. Він може впливати на рух, сили, масштабування та інші аспекти фізичної симуляції. Окрім цього Collider2D може бути використаний як тригер, що дозволяє виявляти колізії, але без впливу на фізичне поведінку об'єктів. Тригер Collider2D спрацьовує, коли інший об'єкт зіткнувся з ним, і тоді він можете виконувати певні дії в коді гри під час спрацювання тригера. На рисунку 3.11 зображено зовнішній вигляд інспектора об’єкта Player.

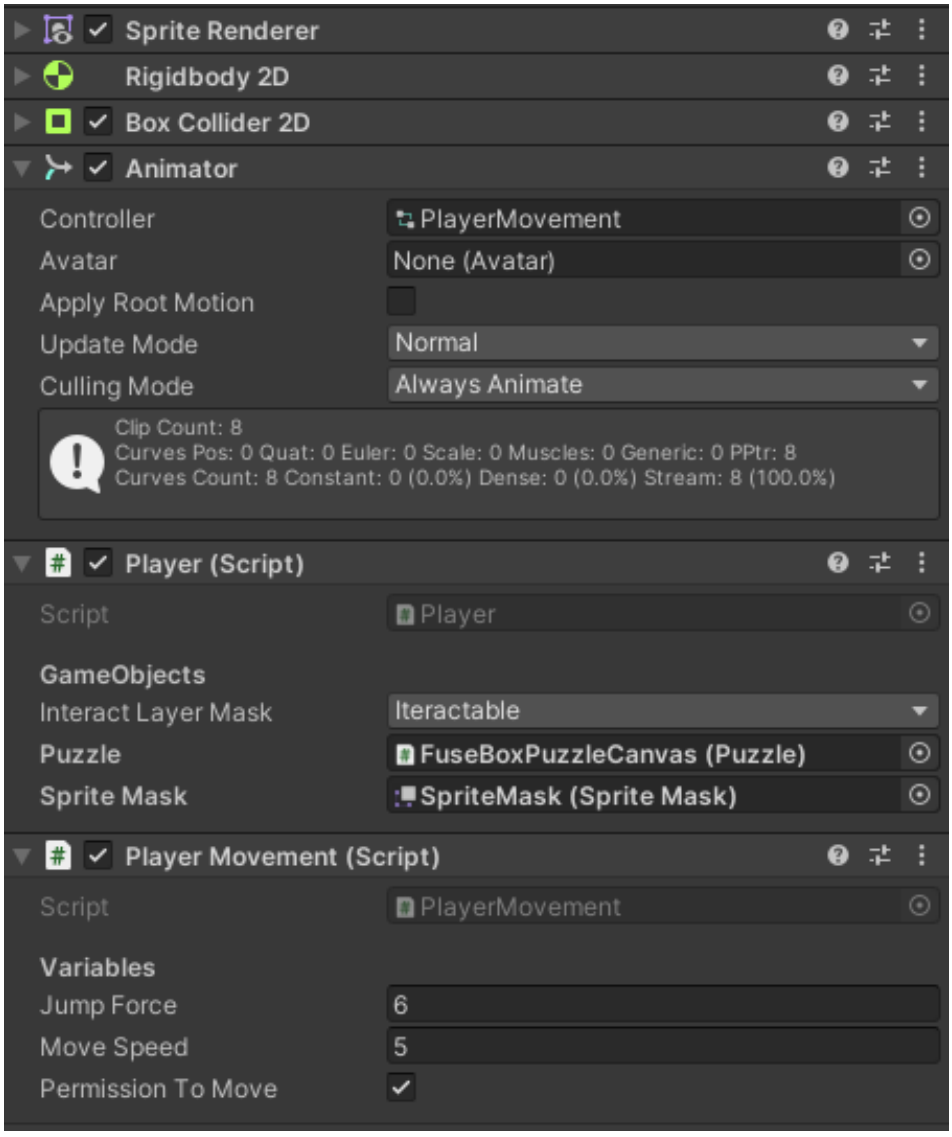


Рисунок 3.11 – Зовнішній вигляд інспектора об’єкта Player

На даному рисунку зображено компоненти, які прив’язані до об’єкта головного героя у інспекторі. Як видно з рисунку серіалізовані поля відображаються у інспекторі і дозволяють вносити зміни в код, не відкриваючи середовища розробки. Перших чотири елементи є інтегрованими компонентами

ігрового рушія. Sprite Renderer відповідає за зовнішній вигляд об'єкта, Rigidbody2D – за його фізичні властивості, Collider2D – за колізію об'єкта. Animator необхідний для відтворення анімацій головного героя на основі спрайтів. Два останніх компонента і є класами Player та PlayerMovement. Також можна побачити значення серіалізованих полів. На рисунку 3.12 зображено першу частину класу Puzzle.

```
6   public class Puzzle : MonoBehaviour
7   {
8       [Header("Variables")]
9       [SerializeField] private float _startScore;
10      [SerializeField] private float _finalScore;
11      [SerializeField] private int _nodeStep = 1;
12      [SerializeField] private int _maxNodeStep = 4;
13
14      [Header("Objects")]
15      [SerializeField] private TextMeshProUGUI _actualScoreTMP;
16      [SerializeField] private TextMeshProUGUI _finalScoreTMP;
17      [SerializeField] private Image _cursorImage;
18
19      private Canvas _canvas;
20      private float _currentScore;
21      private int[] _selectedNodes = new int[4] {1, 0, 0, 0};
22      private int _buttonStep = 1;
23      private bool _permissionToMultiply = true;
24      public event Action puzzleCompleted;
25
```

Рисунок 3.12 – Перша частина класу Puzzle

Наступним класом для опису є клас головоломки, який називається Puzzle. Даний клас використовується класом Player для того аби відкрити графічний елемент Canvas. У даного класу присутні сім полів з функцією серіалізації, які необхідні для налаштування головоломки з інспектора. Даний клас має посилання на Canvas і подію Action, яка означає виконання головоломки.

У Unity, елементом Canvas є графічний контейнер, який використовується для відображення інтерфейсу користувача у відеогрі. Canvas дозволяє вам розміщувати елементи UI, такі як кнопки, текстові поля, зображення та інші, на екрані та керувати їх розташуванням та зовнішнім виглядом.

Canvas є важливим компонентом для створення інтерактивного та респонсивного інтерфейсу в Unity. Він дозволяє створювати компоненти UI,

розміщувати їх у зручному порядку та керувати їх поведінкою та взаємодією з користувачем.

Canvas в Unity працює разом з іншими компонентами, такими як Button, Text, Image і Event System, для створення повноцінного інтерфейсу користувача. Також керувати взаємодією, виглядом та поведінкою елементів UI можлива за допомогою скриптів та налаштування Canvas. На рисунку 3.13 зображено другу частину класу Puzzle.

```
26      private void Start()
27      {
28          _currentScore = _startScore;
29          _actualScoreTMP.SetText(_currentScore.ToString());
30          _finalScoreTMP.SetText(_finalScore.ToString());
31          _canvas = GetComponent<Canvas>();
32      }
33
34      public void SetNodeNumber(int nodeNumber)
35      {
36          int temp = nodeNumber - _selectedNodes[_nodeStep-1];
37          Debug.Log(_nodeStep);
38          Debug.Log(temp);
39          Debug.Log(_buttonStep);
40          if ((temp == 0 || temp == 1) && _nodeStep==_buttonStep)
41          {
42              Debug.Log("true");
43              _selectedNodes[_nodeStep] = nodeNumber;
44              _nodeStep++;
45              _permissionToMultiply = true;
46          }
47          else
48          {
49              _permissionToMultiply = false;
50          }
51      }
```

Рисунок 3.13 – Друга частина класу Puzzle

Першим методом є стандартний метод Start в якому здійснюється ініціалізації полів, а також для встановлення значень текстових полів. Наступним методом є метод, який визначає на який вузол натиснув користувач. Для цього метод прикріплюється до кнопок і в параметрах натискання кнопки визначається, яке число буде використане в методі. Далі йде порівняння чи знаходиться даний вузол під попереднім шляхом порівняння його значення з попереднім значенням. Далі поле, яке відповідає за дозвіл на множення отримує значення “true” і

наступний метод може здійснювати маніпуляції із рахунком. На рисунку 3.14 зображено третю частину класу Puzzle.

```
53      public void MultiplyScore(float multiplyNumber)
54      {
55          if (_permissionToMultiply)
56          {
57              _currentScore *= multiplyNumber;
58              _actualScoreTMP.SetText(_currentScore.ToString());
59              _cursorImage.transform.localPosition -= new Vector3(0, 120, 0);
60              if (_nodeStep == _maxNodeStep && _currentScore == _finalScore)
61              {
62                  Debug.Log("Done");
63                  _canvas.enabled = false;
64                  puzzleCompleted?.Invoke();
65              }
66              else if (_nodeStep == 4 && _currentScore != _finalScore)
67              {
68                  Debug.Log("No");
69                  Reset();
70              }
71          }
72      }
73
74      public void SetButtonStep(int step)
75      {
76          _buttonStep = step;
77      }
```

Рисунок 3.14 – Третя частина класу Puzzle

Наступні два методи, які зображені на рисунку, також прикріплюються до кнопки і при натисканні отримують число для подальших маніпуляцій. Перший метод, який зображений на рисунку здійснює множення числа, яке було вказане на кнопці на поточний рахунок. Також даний метод оновлює текстове поле, яке демонструє користувачу поточний рахунок головоломки.

Якщо гравець натиснув на кнопки на всіх рівнях і поточний рахунок не співпадає з фінальним, то весь результат повертається до стартових значень. Якщо гравець виконав головоломку правильно, то даний клас закриває елемент Canvas і за допомогою метода Invoke викликає подію, яка спрацьовує в класі Player. На рисунку 3.15 зображено четверту частину класу Puzzle.

```
79      public void Reset()
80      {
81          _cursorImage.transform.localPosition = new Vector3(-490, 100, 0);
82          _selectedNodes[1] = 0;
83          _selectedNodes[2] = 0;
84          _selectedNodes[3] = 0;
85          _currentScore = _startScore;
86          _actualScoreTMP.SetText(_currentScore.ToString());
87          _nodeStep = 1;
88      }
89  }
```

Рисунок 3.15 – Четверта частина класу Puzzle

Даний метод, який зображений на рисунку 3.15, необхідний для повернення всіх значень до стартових і прикріплюється до кнопки з відповідною назвою або викликається з коду іншими методами. Також він повертає графічні елементи на їх стартові позиції.

На рисунку 3.16 зображено зовнішній вигляд елемента Canvas у інспекторі Unity.

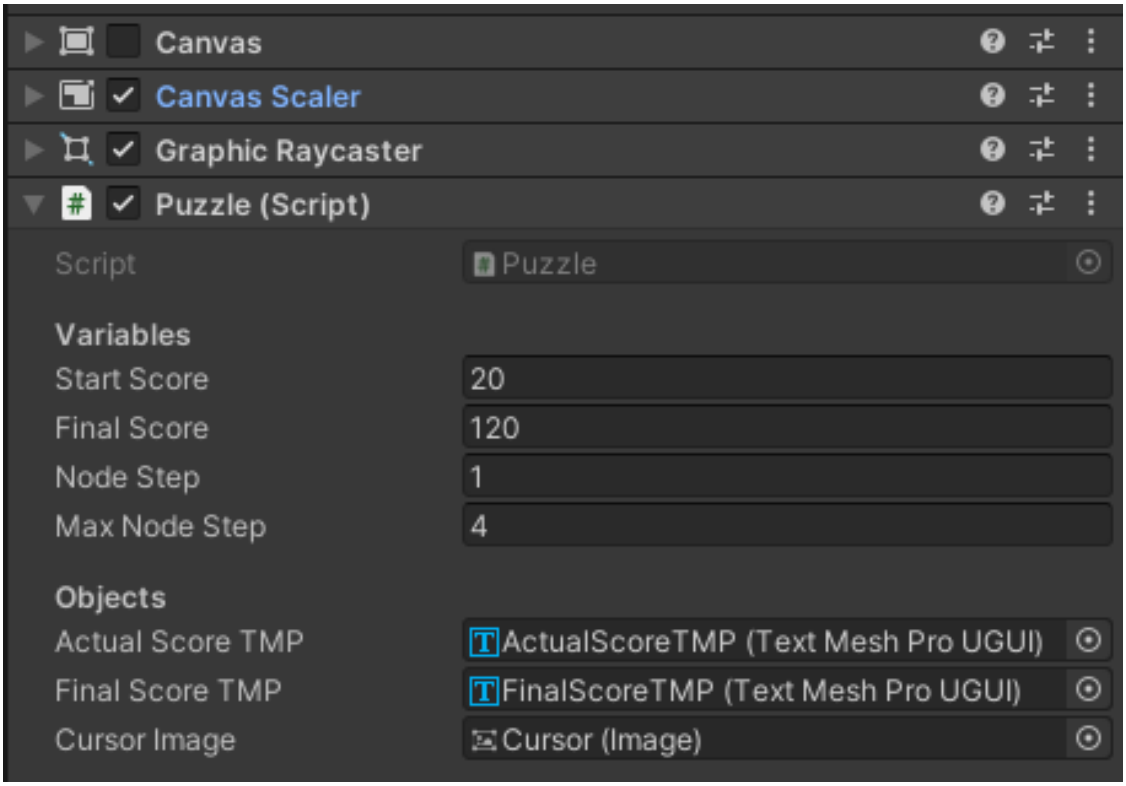


Рисунок 3.16 – Зовнішній вигляд інспектора елемента Canvas

На ньому видно серіалізовані поля, а також, що головний елемент, а саме Canvas по замовчуванню вимкнений. Далі, під час ігрового процесу при взаємодії зі щитком цей елемент за допомогою методу зі скрипта Puzzle вмикається і гравець матиме змогу проходити головоломку.

На рисунку 3.7 зображено налаштування однією з кнопок, які знаходяться в графічному контейнері Canvas. Як видно з рисунка при натисканні на кнопку здійснюється виклик трьох методів з передачею в них значень.

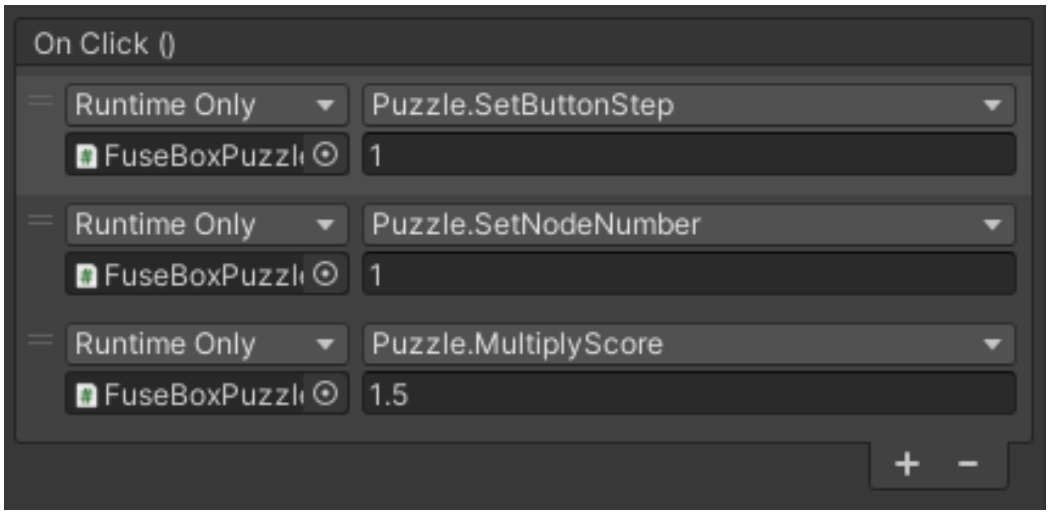


Рисунок 3.17 – Приклад зовнішнього вигляду кнопки в інспекторі

Код додаткових модулів, які використовуються основним класами подано у додатку Б.

3.2 Реалізація графічної частини

Для ігрового застосунку було створено вісім анімацій. Дані анімації використовує компонент Animator у якому було встановлено лінії переходів від однієї до іншої анімації та умови зміни анімації. У Unity даний компонент використовується для керування анімацією об'єктів у відеоіграх та застосунках. Він дозволяє створювати та керувати різними станами анімацій, переходами між ними та параметрами, що впливають на їхню поведінку.

Анімація Idle є базовою для всіх аніматорів, хоча може мати інші назви. Саме дана анімація запускається першої і переходить в інші анімації. Також як видно з рисунку для анімації Jump є умова, що він може виконуватися після любої анімації.

У аніматорі також були створені параметри, які використовуються для зміни анімацій. Два параметри є цілочисельного типу даних, один логічного типу даних і інших два типи даних як тригери для спрацювання анімацій.

Також у чотирьох анімація були створені Animation Event, а саме для анімацій Jump, Landing, Run, RunStop. Animation Event називають механізм, який дозволяє вставляти події в різні точки відтворення анімації.

Animation Events налаштовуються у редакторі анімацій Unity та у цьому ж редакторі пов'язуються з певними функціями в коді гри. Коли анімація досягає точки Animation Event під час відтворення, виконується відповідна функція у коді гри. Зовнішній вигляд елемента Animator зображено на рисунку 3.18.

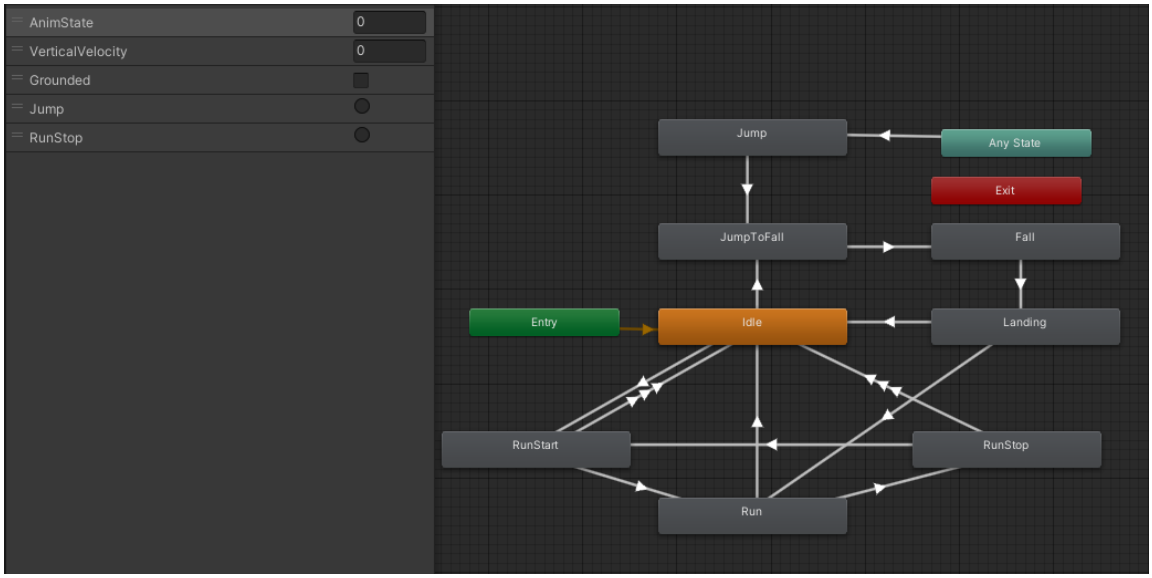


Рисунок 3.18 – Зовнішній вигляд елемента Animator

Animator в Unity називають компонент, який використовується для керування анімацією об'єктів у грі. Він дозволяє створювати та управляти різними станами, переходами та параметрами анімації для об'єкта. Animator працює спільно з анімаційними кліпами, які містять набір кадрів анімації. Анімаційні кліпи можуть бути створеними у спеціальних програмах для анімації, таких як

Unity's Animation Window, або бути імпортованими за допомогою зовнішніх інструментів.

Анімаційні кліпи елемента Animator містять інформацію про кадри анімації для різних станів об'єкта. Кліпи можна налаштовувати для відтворення на різних швидкостях, зворотно або з певною затримкою.Параметри аніматора визначають умови, за яких будуть виконуватися переходи між станами анімації. Вони можуть бути логічними значеннями, числовими параметрами або перерахуваннями.

Стани в аніматорі представляють окремі стани анімації, які об'єкт може мати. Вони можуть включати інформацію про анімаційний кліп, параметри та переходи до інших станів.Переходи визначають умови, за яких буде здійснюватися перехід від одного стану до іншого. Вони використовуються для визначення логіки переходу між різними анімаційними станами.

Механізм керування елементом Animator надає методи та властивості для керування станами, параметрами та виконанням анімації. Він дозволяє змінювати значення параметрів, викликати переходи між станами та контролювати відтворення анімації з коду скрипта. На рисунку 3.19 зображено зовнішній вигляд фінальної головоломки.

Рисунок 3.19 – Зовнішній вигляд фінальної головоломки

Користувач на кожному рівні натискає на кнопку і тим самим збільшує рахунок на коефіцієнт, який написаний на кнопці. Гравець може обрати лише дві нижні сусідні кнопки від попередньої кнопки. Стрілка з лівого боку зображує актуальний ряд кнопок для вибору. На рисунку 3.20 зображено зовнішній вигляд головного меню розроблюваної відеогри.

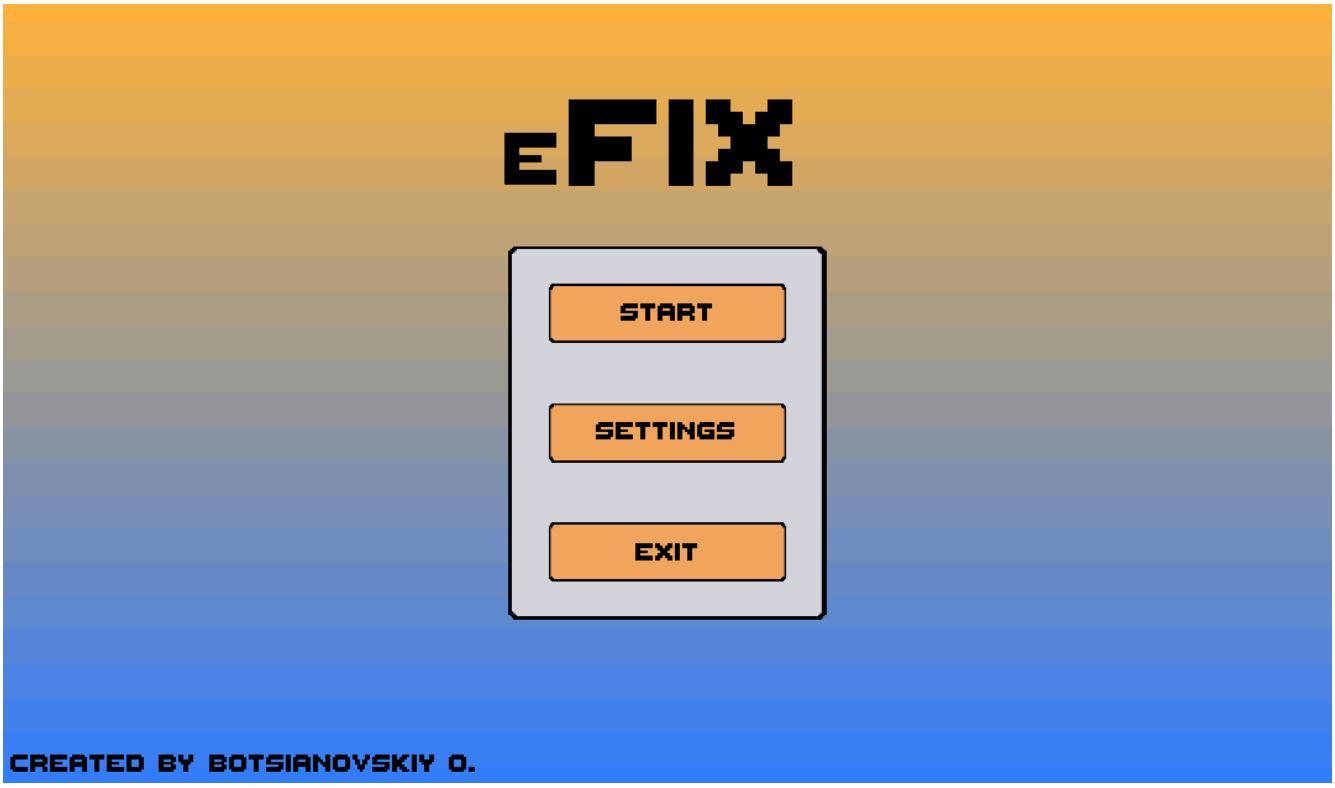


Рисунок 3.20 – Зовнішній вигляд головного меню

Головне меню було реалізовано на основі макета, який був попередньо спроектований. В головному меню присутні три кнопки, які відповідають за початок гри, налаштування і вихід з відеогри.

3.3 Тестування

Процес тестування є важливим етапом розробки ігрового застосунку. Тестування допомагає виявити помилки, деякі неполадки та неочікувані поведінки в ігровому проекті. Це дозволяє виправити проблеми перед тим, як гра дійде до кінцевого користувача. Виявлення і виправлення помилок рано у процесі

розробки може значно скоротити час та зусилля, необхідні для подальшого розробки.

Якісне тестування допомагає переконатися, що гра відповідає очікуванням користувача та заданим вимогам. Воно дозволяє перевірити, чи працюють функції гри так, як повинні, чи немає непередбачених проблем з продуктивністю, стабільністю та користувальницьким інтерфейсом.

Тестування ігрового додатку здійснювалося за допомогою ігрового рушія Unity в якому є інтегровані засоби тестування. Для тестування був створений окремий проект в якому були ті самі класи, але зі зміненими сигнатурами методів для тестування. На рисунку 3.21 зображено успішно пройдений тест в ігровому рушії Unity.

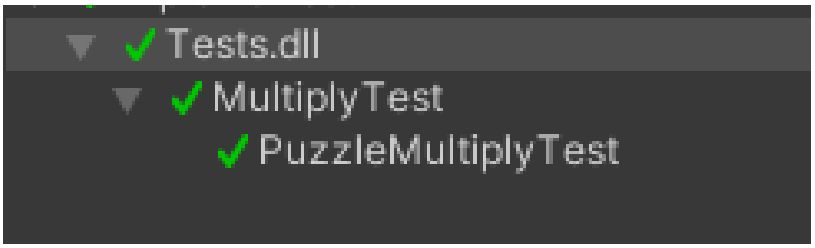


Рисунок 3.21 – Зовнішній вигляд успішно пройденого тесту в Unity

На рисунку 3.22 зображено код класу написаного тесту.

```
7   public class MultiplyTest
8   {
9       Puzzle puzzle = new Puzzle();
10
11      [Test]
12      public void PuzzleMultiplyTest()
13      {
14          puzzle.currentScore = 20;
15          float temp = puzzle.MultiplyScore(1.5f);
16          Assert.AreEqual(20*1.5f, temp);
17      }
18  }
19
```

Рисунок 3.22 – Зовнішній код класу написаного тесту.

Також ігровий рушій Unity має змогу запускати симуляцію відеогри і тим самим дозволяє розробникам тестувати відеогру під час її розробки. Таким чином під час розробки відеогри постійно здійснювалося її тестування шляхом запуску окремих фрагментів відеогри.

3.4 Висновки

В даному розділі здійснювалася розробка ігрового застосунку на ігровому рушії Unity з використанням мови програмування C#. Розробка програмної та візуальної частини здійснювалася паралельно одна одній.

Також було описано основні класи і їх методи ігрового застосунку. Було описано їх взаємодію з іншими компонентами розроблюваного програмного рішення. Також було описано налаштування об'єктів в інспекторі Unity.

Було описано розробку графічної частини відеогри. Було описано взаємодії анімацій в ігровому компоненті аніматора. Було продемонстровано скріншоти окремих сцен і ігрових компонентів в ігровому рушії.

Заключним етапом розробки стало тестування ігрового застосунку. Тестування здійснювалося за допомогою ручного тестування шляхом запуску симуляції відеогри або її окремих фрагментів. Також були написані Unit тести в окремому проекті, де були видозмінені використані класи. За допомогою цих тестів було протестовано елементи головоломок відеогри.

ВИСНОВКИ

Поставлена мета і задача кваліфікаційної роботи були виконані у повній мірі. На основі усіх попередніх етапів, які були виконані у даній кваліфікаційній роботі, було проведено аналіз підсумків.

У першому розділі був проведений детальний аналіз предметної області відповідно до теми кваліфікаційної роботи. Також був здійснений аналіз популярних існуючих рішень із встановленням їхніх переваг та недоліків. На основі отриманої інформації були розроблені основні вимоги до розробки майбутнього програмного рішення. Також була обрана модель життєвого циклу, яка найбільше адаптована для розробки відеоігор.

У другому розділі було здійснено аналіз наявних архітектур і патернів та було здійснено порівняння двох архітектур, які підходять для розробки ігрового застосунку. Також було розроблено макети інтерфейсу для ігрового застосунку за допомогою веб-сервісу Figma. Далі було здійснена розробка діаграм, які описують поведінку і алгоритм роботи ігрового застосунку. В кінці другого розділу були проаналізовані два ігрових рушія, їх переваги та недоліки і була сформована таблиця для порівняння.

У третьому розділі після етапів дослідження і проектування було здійснено реалізацію програмного рішення відповідно до вимог та раніше вибраних технологій з проведенням ретельного аналізу окремих компонентів. Під час розробки здійснювалося ручне тестування окремих модулів відеогри та було здійснено тестування за допомогою Unit-тестів при завершенні розробки ігрового застосунку.

Розроблений ігровий застосунок у повній мірі відповідає поставленим функціональним та нефункціональним вимогам. Для кожного розділу були сформовані проміжні висновки. Також відповідно до вибраної моделі життєвого циклу передбачається поетапна розробка майбутніх модулів відеогри.

ПЕРЕЛІК ДЖЕРЕЛ

1. Уніфікована мова програмування UML. Портал знань : веб-сайт. – URL: http://www.znannya.org/?view=uml (дата звернення: 26.02.2023).
2. Методичні вказівки щодо його виконання для студентів спеціальності 121 «Інженерія програмного забезпечення». URL: https://msn.khmnu.edu.ua/pluginfile.php/626097/mod_resource/content/1/Методичка%20по%20ДП.pdf (дата звернення: 13.02.2023).
3. Gregory J. Game Engine Architecture, Third Edition. A K Peters/CRC Press, 2018. – 1240 с.
4. Watkins R. Procedural Content Generation for Unity Game Development. Packt Publishing, Limited, 2016. – 260 с.
5. Unity Documentation. Unity Documentation: веб-сайт. URL: https://docs.unity3d.com/ScriptReference/. (дата звернення: 16.04.2023).
6. Rigidbody2D. Unity Documentation: веб-сайт. URL: https://docs.unity3d.com/ScriptReference/Collider2D.html (дата звернення: 10.03.2023).
7. Nystrom R. Game Programming Patterns. Genever Benning, 2014. 354 с.
8. Unity Learn. Unity Learn: веб-сайт. URL: https://learn.unity.com. (дата звернення: 10.03.2023).
9. Nystrom R. Game Programming Patterns. Apress, 2011. 300 с.
10. Canvas. Unity Documentation: веб-сайт. URL: https://docs.unity3d.com/ScriptReference/Canvas.html. (дата звернення: 26.04.2023).
11. Robert C. Martin. Clean Architecture: A Craftsman's Guide to Software Structure and Design / Robert C. Martin – Publisher(s): Pearson, 2017. – 432 с.
12. Collider2D. Unity Documentation: веб-сайт. URL: https://docs.unity3d.com/ScriptReference/Collider2D.html. (дата звернення: 18.04.2023).
13. Unity Asset Store. Unity Asset Store: веб-сайт. URL: https://assetstore.unity.com. (дата звернення: 18.04.2023).

14. Bond J. G. Introduction to Game Design, Prototyping, and Development. Pearson Education, Limited, 2022. – 106 с.

15. Visual Studio. Visual Studio: веб-сайт. URL: https://visualstudio.microsoft.com. (дата звернення: 16.02.2023).

16. Design Basics. Figma: веб-сайт. URL: https://www.figma.com/resource-library/design-basics. (дата звернення: 06.03.2023).

17. Троелсен Ендрю. Мова програмування C# та платформа .Net і .Net Core, 8-е видання.: Пер. з англ / Ендрю Троелсен., 2018 – 1328 с.

18. Огляд видів тестування. QATestlab: веб-сайт. URL: https://training.qatestlab.com/blog/technical-articles/review-the-types-of-testing/. (дата звернення: 02.05.2023).

19. Unit testing automation with Unity. Codemagic: веб-сайт. URL: https://blog.codemagic.io/unit-testing-automation-unity/. (дата звернення: 02.05.2023).

20. Леонов О. Тестування ПЗ (види тестування) / О. Леонов [Електронний ресурс] – Режим доступу: https://drukarnia.com.ua/articles/testuvannya-pz-vidi-testuvannya-JInS1#heading-3-4659 (дата звернення – 12.02.2023). – Назва з екрану.

21. Hocking J. Unity in Action: Multiplatform game development in C#. Manning Publications, 2018. – 400с.

22. Unity Forum. Unity Forum: веб-сайт. URL: https://forum.unity.com. (дата звернення: 22.04.2023).

23. Buttfield-Addison P., Manning J., Nugent T. Unity Game Development Cookbook: Essentials for Every Game. O'Reilly Media, 2019. 406 с.

24. Unity repositories. Github: веб-сайт. URL: https://github.com/search?q=unity+c%23. (дата звернення: 22.04.2023).

25. Jackson S. Mastering Unity 2D Game Development. Packt Publishing, Limited, 2014. – 474с.

26. Pico Park. PicoPark: веб-сайт. URL: https://picoparkgame.com/en/. (дата звернення: 14.02.2023).

27. Fez. Steam Store: веб-сайт. URL: https://store.steampowered.com/app/224760/FEZ/. (дата звернення: 14.02.2023).

28. Unravel. Steam Store: веб-сайт. URL: https://store.steampowered.com/app/1225560/Unravel/. (дата звернення: 14.02.2023).

29. Introduction to Unity UI. Kodeco: веб-сайт. URL: https://www.kodeco.com/34347684-introduction-to-unity-ui-part-1. (дата звернення: 20.04.2023).

30. Dickinson C. Unity 2017 Game Optimization - Second Edition. Packt Publishing, Limited, 2017. – 376 с.

31. Geig M. Unity Game Development in 24 Hours, Sams Teach Yourself. Pearson Education, Limited. – 383 с.

32. Animator. Unity Documentation: веб-сайт. URL: https://docs.unity3d.com/ScriptReference/Animator.html. (дата звернення: 24.04.2023).

33. Collider's and Triggers in Unity. Chris Hilton personal blog: веб-сайт. URL: https://christopherhilton88.medium.com/colliders-and-triggers-in-unity-understanding-the-basics-7192714f3440. (дата звернення: 26.04.2023).

34. What is Unreal Engine. BairesDevBlog: веб-сайт. URL: https://www.bairesdev.com/blog/what-is-unreal-engine/. (дата звернення: 12.04.2023).

35. New Input System. Сайт Unity Documentation: веб-сайт. URL: https://docs.unity3d.com/Packages/com.unity.inputsystem@1.6/manual/index.html. (дата звернення: 14.04.2023).

36. Creating 2D Player Movement. YarsaLabs Dev Blog: веб-сайт. URL: https://blog.yarsalabs.com/player-movement-with-new-input-system-in-unity/. (дата звернення: 14.04.2023).

37. ECS For Unity. Unity: веб-сайт. URL: https://unity.com/ecs. (дата звернення: 14.04.2023).

38. Unity With MVC. Toptal: веб-сайт. URL: https://www.toptal.com/unity-unity3d/unity-with-mvc-how-to-level-up-your-game-development. (дата звернення: 15.04.2023).

39. MonoBehaviour. Unity Documentation: веб-сайт. URL: https://docs.unity3d.com/ScriptReference/MonoBehaviour.html. (дата звернення: 14.04.2023).

40. Events&Delegates in Unity. gamedevbegginer: веб-сайт. URL: https://docs.unity3d.com/Manual/EventFunctions.html. (дата звернення: 14.04.2023).

41. Scolastici C. Unity 2D Game Development Cookbook. Packt Publishing - ebooks Account, 2015. 256 с.

42. Event functions. Unity Documentation: веб-сайт. URL: https://docs.unity3d.com/Manual/EventFunctions.html. (дата звернення: 11.04.2023).

43. Code Monkey. Code Monkey: веб-сайт. URL: https://unitycodemonkey.com. (дата звернення: 20.03.2023).

44. Популярні життєві цикли розробки ПЗ. QATestlab: веб-сайт. URL: https://training.qatestlab.com/blog/technical-articles/popular-software-development-life-cycles/. (дата звернення: 08.02.2023).

ДОДАТОК А

(обов'язковий)

ПРЕЗЕНТАЦІЙНИЙ МАТЕРІАЛ

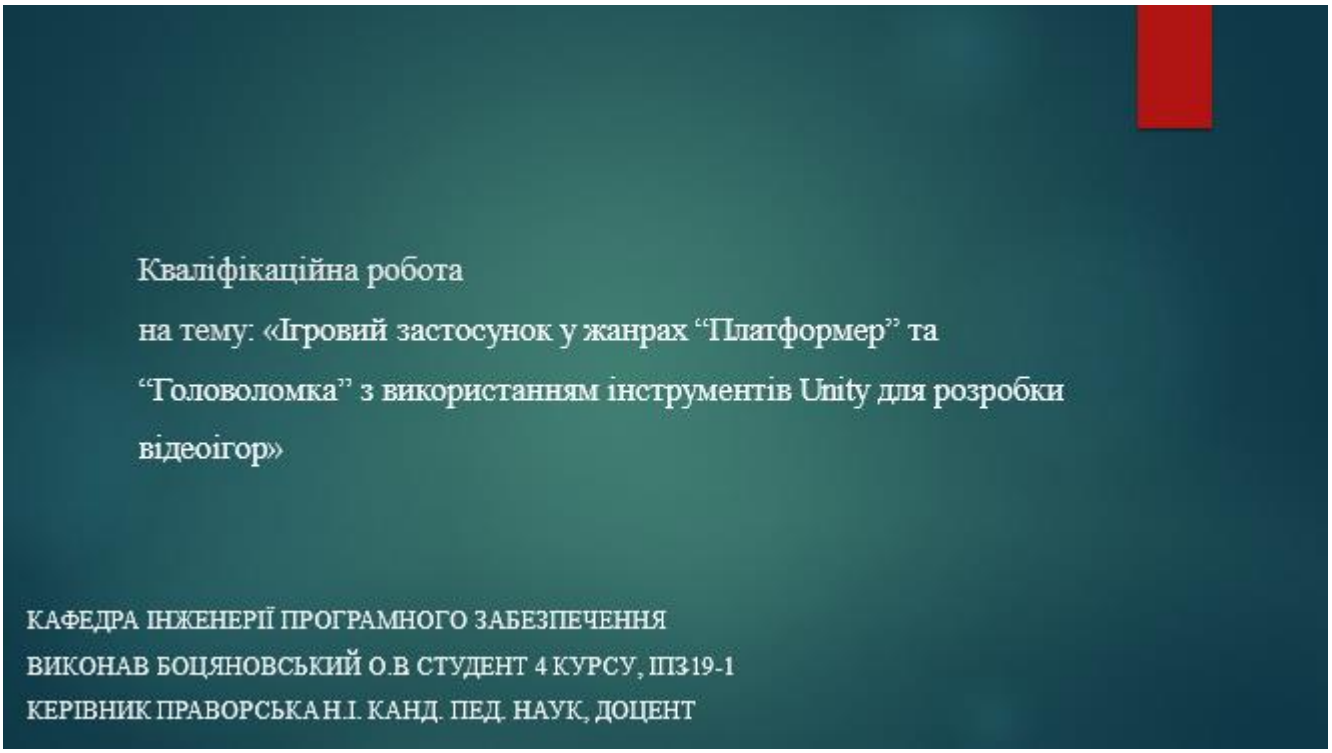


Рисунок А.1 – Слайд №1

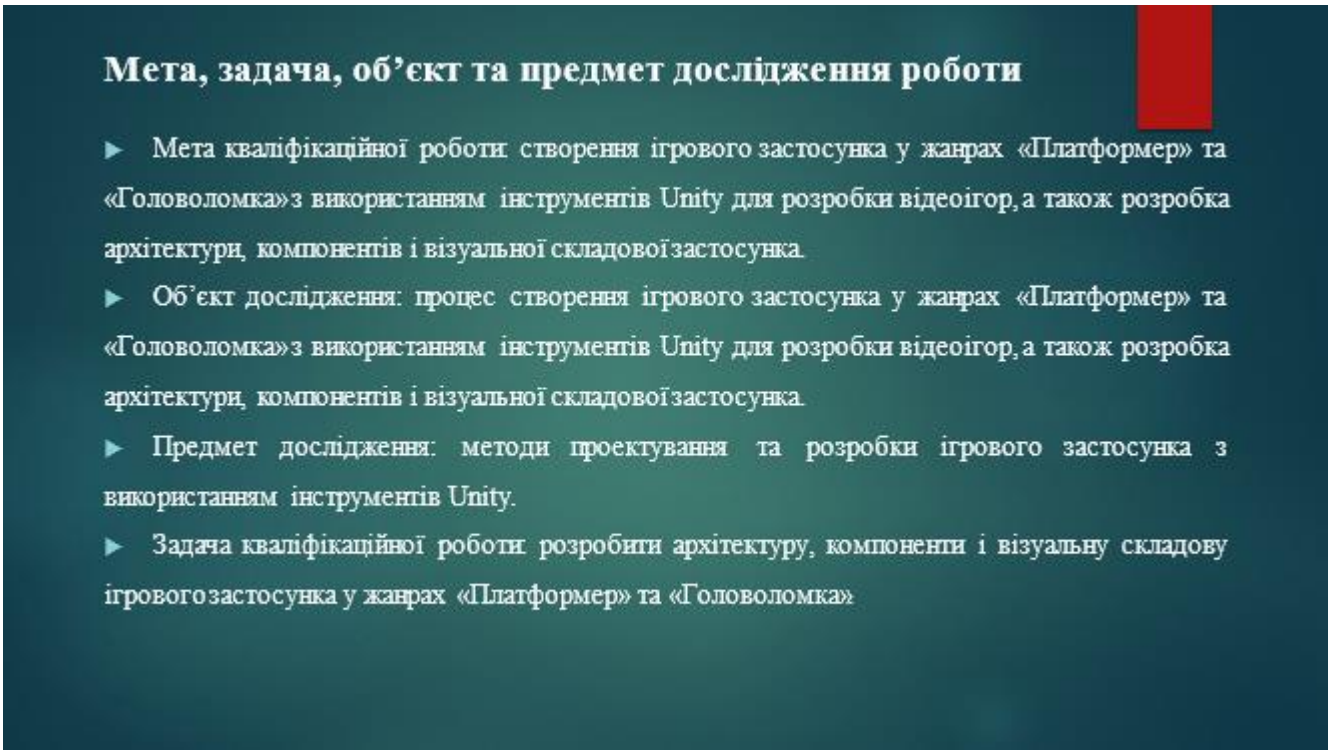


Рисунок А.2 – Слайд №2

Актуальність кваліфікаційної роботи

Розробка відеоігор є актуальною темою, оскільки при розробці ігор використовуються переважно найновіші технології і їх розробка стимулюю створення нових. Крім того, відеоігри стали однією з найбільш популярних форм розваг у світі, що робить їх важливими з точки зору культурної та соціальної важливості.

Також є актуальним розробляти відеоігри з точки зору бізнесу, оскільки галузь відеоігор стала значним сегментом ринку розваг, що генерує значну кількість прибутку для розробників. Розробка відеоігор стала важливим аспектом економіки країн, що мають розвинену галузь інформаційних технологій.

Обидва цих фактори роблять розробку відеоігор важливою темою для дослідження і розвитку в різних сферах життя, що забезпечує необхідність в компетентних фахівцях у цій галузі.

Рисунок А.3 – Слайд №3

Аналіз існуючих рішень

При аналізі наявних рішень було проаналізовано три подібні відеогри. Підсумовуючи аналіз трьох подібних відеоігор, які відповідають предметній області, можна зазначити, що вони використовують систему рівнів і пошук певного предмета для проходження рівня є вдалим, і дану систему слід використати для майбутнього ігрового застосунка. Всім відеоіграм притаманне нарощування складності по мірі проходження відеогри та пряма і непряма взаємодія з навколишнім світом.

Загалом всі три ігрових застосунка є схожими і при цьому в кожній відеогрі реалізовані свої цікаві і унікальні механіки.



Рисунок А.4 – Слайд №4

Проектування ігрового застосунка

На основі аналізу наявних життєвих моделей для реалізації програмного продукту було вибрано найбільш підходящу модель життєвого циклу, а саме інкрементну модель життєвого циклу. Інкрементна модель передбачає, що вимого не завжди є добре відомими і їх визначення не є легким, але вони можуть бути визначеними у кожному циклі, але при цьому вони не змінюються дуже часто. Також дана модель передбачає, що сформовані вимоги демонструються складність програмної системи і, що вимоги володіють функціональними властивостями на ранньому етапі циклу.

Рисунок А.5 – Слайд №5

Вибір архітектури і патернів

На основі аналізу наявних архітектур, які можуть бути використані для розробки ігрового застосунка було вибрано ЕС архітектуру, яка є спрощеною архітектурою ECS. Також була проаналізована архітектура MVC.

При розробці застосунка буде використовуватися такі патерни, як Singleton, State Machine, CompositionOver Inheritance.

Патерн Singleton дозволить створювати лише один екземпляр класу в одній запущеній сцені, а також звертатися до нього з любого місця програми. Патерн State Machine дозволить наділити певні об’єкти певної кількістю станів і переключати їх з наявного переліку станів. Патерн Composition Over Inheritance дозволяє створювати більш гнучкі та масштабовані системи, оскільки він дозволяє динамічно додавати та видаляти компоненти з об'єктів в процесі редагування гри.

Рисунок А.6 – Слайд №6

Рисунок А.7 – Слайд №7

Рисунок А.8 – Слайд №8

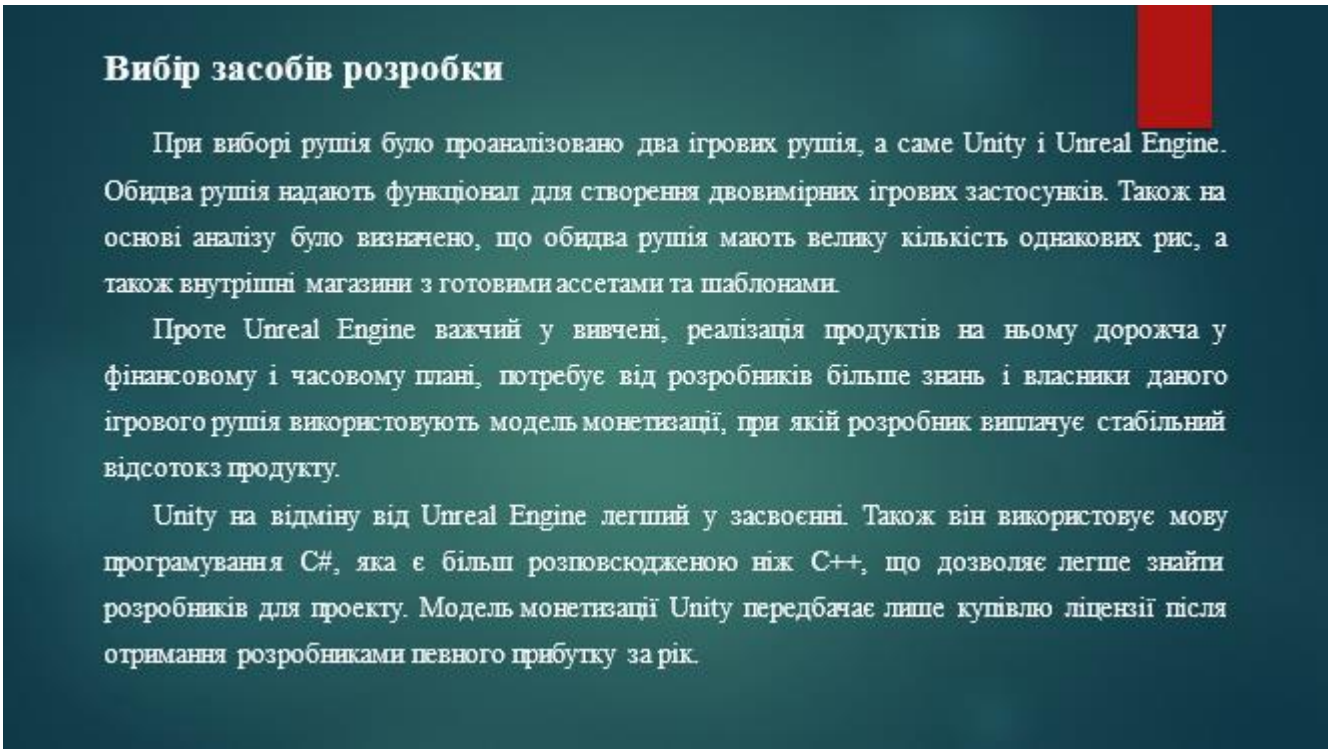


Рисунок А.9 – Слайд №9

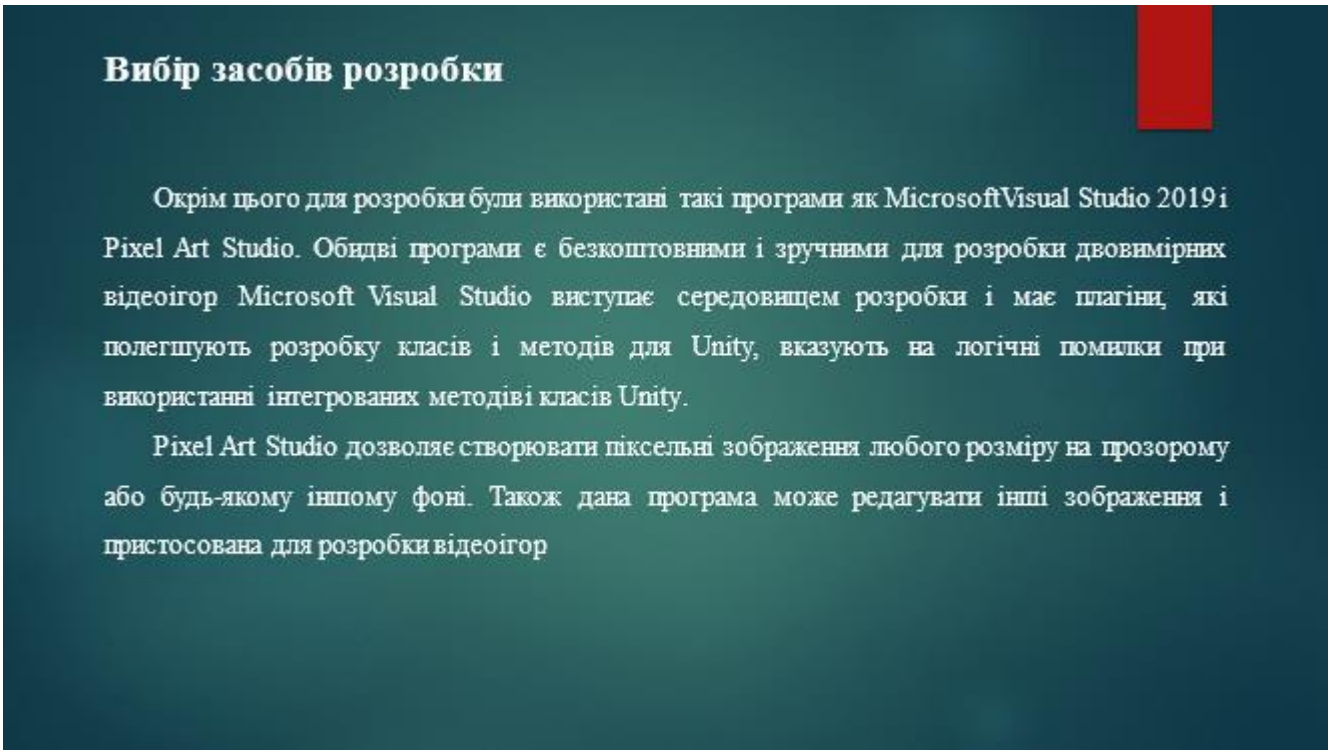


Рисунок А.10 – Слайд №10

Рисунок А.11 – Слайд №11

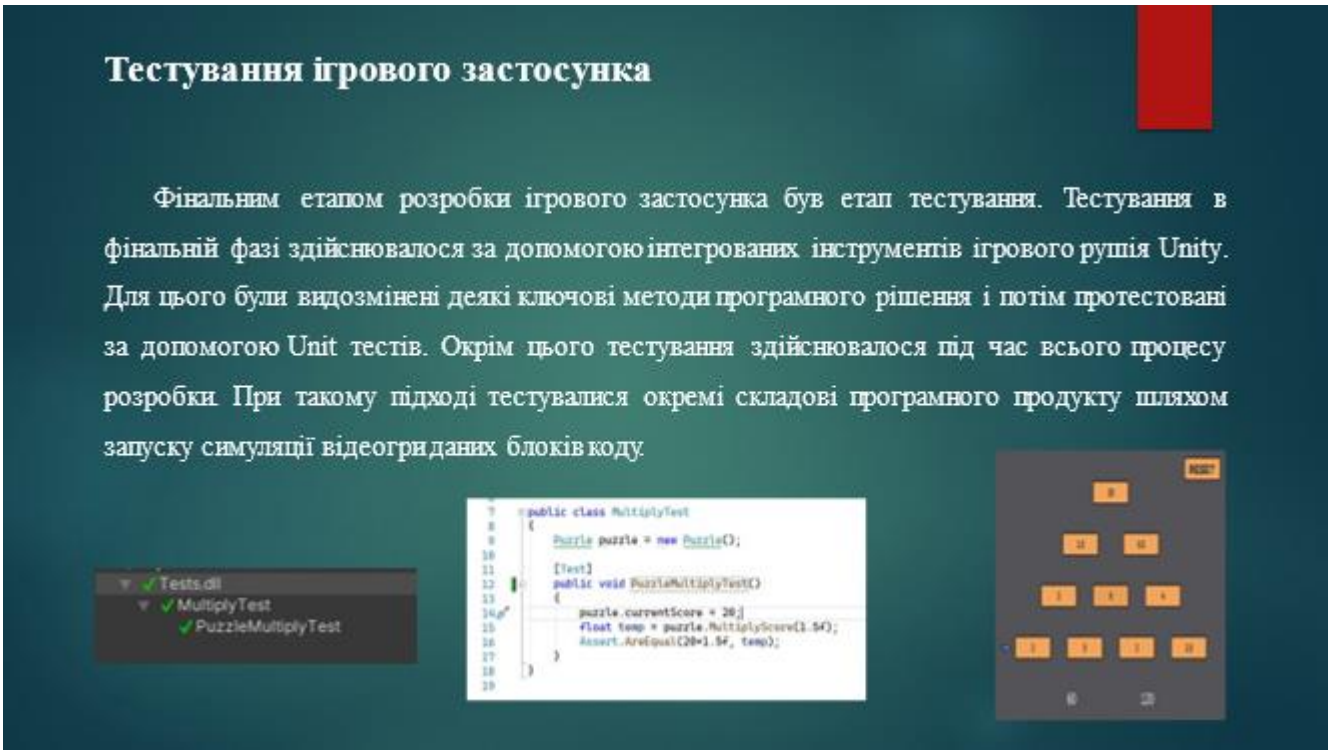


Рисунок А.12 – Слайд №12

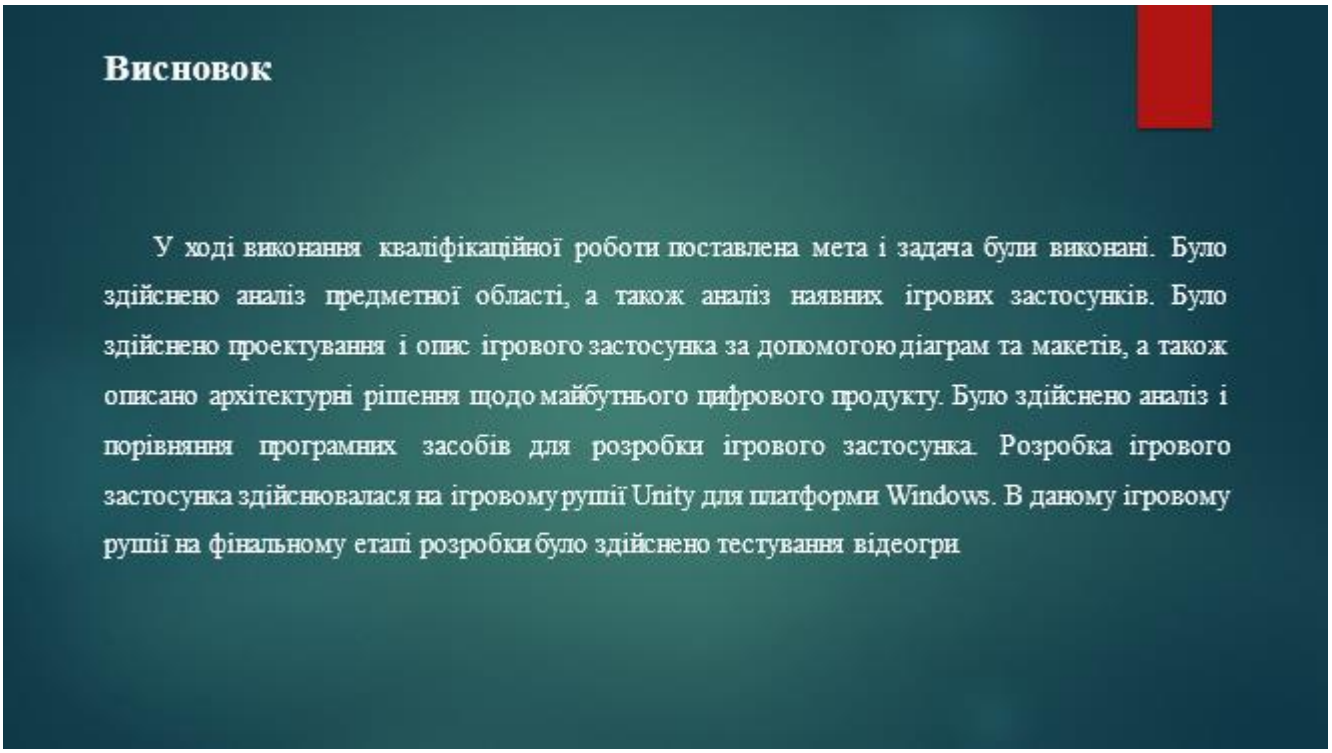


Рисунок А.13– Слайд №13

Рисунок А.14 – Слайд №14

ДОДАТОК Б

(обов’язковий)

ПРОГРАМНИЙ КОД ДОДАТКОВИХ МОДУЛІВ

Б.1 Код класу FuseBox

```
using UnityEngine;

public class FuseBox : MonoBehaviour
{
    [Header("Canvas")]
    [SerializeField] private Canvas _hintCanvas;
    [SerializeField] private Canvas _puzzleCanvas;

    public void EnableHintCanvas()
    {
        _hintCanvas.enabled = true;
    }

    public void DisableHintCanvas()
    {
        _hintCanvas.enabled = false;
    }

    public void TogglePuzzleCanvas()
    {
        bool temp = _puzzleCanvas.enabled;
        _puzzleCanvas.enabled = !temp;
    }
}
```

Б.2 Код класу BrokenCable

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
public class BrokenCable : MonoBehaviour
{
    [SerializeField] private Sprite _fixedCabelSprite;

    private SpriteRenderer _spriteRenderer;

    void Start()
    {
        _spriteRenderer = GetComponent<SpriteRenderer>();
    }
    private void ChangeSpriteToFixed()
    {
        _spriteRenderer.sprite = _fixedCabelSprite;
    }

    private void DisableBrokenCableScript()
    {
        Destroy(GetComponent<BrokenCable>());
    }
    public void FixCable()
    {
        ChangeSpriteToFixed();
        DisableBrokenCableScript();
    } }
```

Б.3 Код класів Sound та AudioManager

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

namespace Assets.Scripts
{
    [System.Serializable]
    public class Sound
    {
        public string _name;
        public AudioClip[] _clips;

        [Range(0f, 1f)]
        public float volume = 1.0f;
        [Range(0f, 1.5f)]
        public float pitch = 1.0f;
        public Vector2 _randomVolumeRange = new Vector2(1.0f, 1.0f);
        public Vector2 _randomPitchRange = new Vector2(1.0f, 1.0f);

        private AudioSource _source;

        public void SetSource(AudioSource source)
        {
            _source = source;
            int randomClip = Random.Range(0, _clips.Length - 1);
            _source.clip = _clips[randomClip];
        }
```

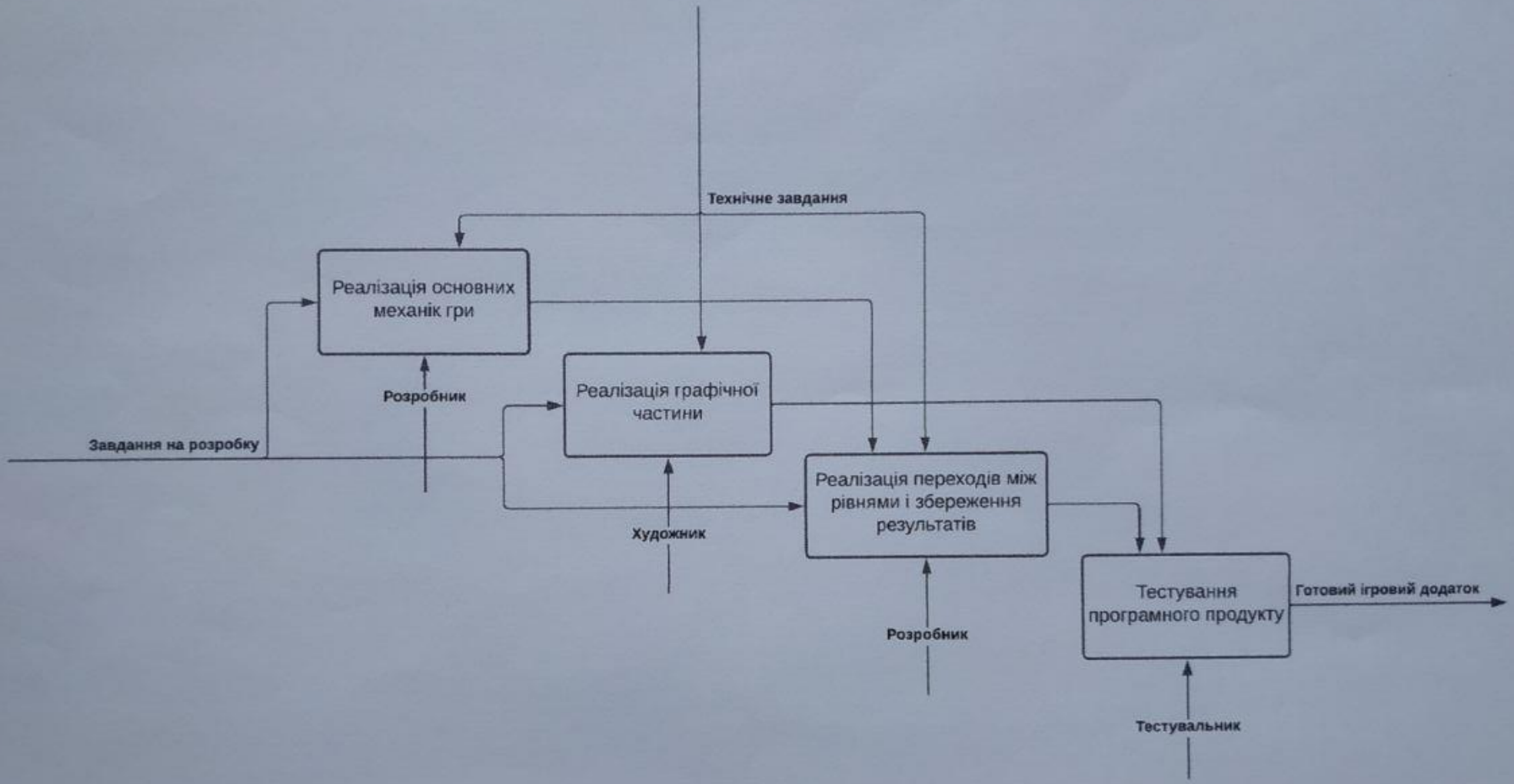
```
        public void Play()
        {
            if (_clips.Length > 1)
            {
                int randomClip = Random.Range(0, _clips.Length - 1);
                _source.clip = _clips[randomClip];
            }
            _source.volume = volume * Random.Range(_randomVolumeRange.x,
_randomVolumeRange.y);
            _source.pitch = pitch * Random.Range(_randomPitchRange.x,
_randomPitchRange.y);
            _source.Play();
        }
    }

    public class AudioManager : MonoBehaviour
    {
        public static AudioManager instance;

        [SerializeField]
        Sound[] _sounds;

        private void Awake()
        {
            if (instance != null)
            {
                Debug.LogError("More than one AudioManger in scene");
            }
            else {
```

```
            instance = this;
        }
    }

    private void Start()
    {
        for (int i = 0; i < _sounds.Length; i++)
        {
            GameObject go = new GameObject("Sound_" + i + "_" + _sounds[i]._name);
            go.transform.SetParent(transform);
            _sounds[i].SetSource(go.AddComponent<AudioSource>());
        }
    }

    public void PlaySound(string name)
    {
        for (int i = 0; i < _sounds.Length; i++)
        {
            if (_sounds[i]._name == name)
            {
                _sounds[i].Play();
                return;
            }
        }

        Debug.LogWarning("AudioManager: Sound name not found in list: " + name);
    }
  }
}
```

ГРАФІЧНА ЧАСТИНА

(обов’язкова)

КвРІПЗ.190125.01.01.Е8



Зм.	Арк.	№ докум.	Підпис	Дата	КвРІПЗ.190125.01.01.Е8			
Розробив		Боцяновський О. В.		6.06	Діаграма декомпозиції IDEF0 процесу реалізації застосунку	Літера	Маса	Масштаб
Керівник		Праворська Н. І.		6.06		Аркуш 1	Аркушів 3	
Консульт.						ХНУ, ІПЗ-19-1		
Н. Контр.		Яшина О. М.		6.06				
Зав. каф.		Бедратюк Л. П.						

КвРІПЗ.190125.01.01.Е8

Зм.	Арк.	№ докум.	Підпис	Дата	КвРІПЗ.190125.01.01.Е8			
Розробив		Бондаренко О. В.		6.06	Діаграма декомпозиції IDEF0 алгоритму проходження рівня	Літера	Маса	Масштаб
Керівник		Гравєрська Н.		6.06				
Консульт.						Аркуш 2	Аркушів 3	
Н. Контр.		Яшина О. М.		6.06		ХНУ, ІПЗ-19-1		
Зав. каф.		Бездрапок Л. П.		6.06				

КвРІПЗ.190125.01.01.Е8

Зм.	Арк.	№ докум.	Підпис	Дата	КвРІПЗ.190125.01.01.Е8				
Розробив		Бодиновський О. В.		6.06	Діаграма варіантів використання			Літера	Маса	Масштаб
Керівник		Правдорова Н. І.		6.06						
Консульт.										
								Аркуш 3	Аркушів 3	
Н. Контр.		Яшина О. М.		6.06						
Зав. каф.		Федяпок П. П.		6.06				ХНУ, ІПЗ-19-1		

Супровідні документи

Anti-Plagiarism v-15.257

Максимальне співпадіння з одним документом 15.0%

Словники перевірки: en_US, ru_RU, ua_UA. **Помилок в документах: 9%**

	Документ		Сумарний збіг по Базі Даних	
	Символи	Лексеми	Символи	Лексеми
ID: 114527 Назва: БКР Ігровий застосунок у жанрах “Платформер” та “Головоломка” з використанням інструментів Unity для розробки відеоігор Додано в БД: 2023-06-01 Автора: Боцяновський О.В. Керівники: Праворська Н.І. Консультанти: Опоненти:	80650	722	12399 (15%)	109 (15%)

Джерело плагіату

ID	Опис	Наявність плагіату в документі	
		Символи	Лексеми
111869	Назва: Звіт з переддипломної практики бакалавр Добавлено в БД: 2023-03-02 Автора: Боцяновський О.В. Керівники: Праворська Н.І, Консультанти: Опоненти:	11909 (15.0%)	100 (14.0%)

UNICHECK

Ім'я користувача:
Кафедра ІПЗ

ID перевірки:
1015380212

Дата перевірки:
01.06.2023 22:50:37 EEST

Тип перевірки:
Doc vs Internet + Library

Дата звіту:
01.06.2023 22:51:38 EEST

ID користувача:
100005589

Назва документа: **Боцяновський_Олексій,_перевірка_на_антиплагіат**

Кількість сторінок: **68** Кількість слів: **12531** Кількість символів: **94653** Розмір файлу: **3.19 MB** ID файлу: **1015045630**

Виявлено модифікації тексту (можуть впливати на відсоток схожості)

3.93%
Схожість

Найбільша схожість: **1.15%** з джерелом з Бібліотеки (ID файлу: **1015017362**)

2.59% Джерела з Інтернету	350	Сторінка 70
2.62% Джерела з Бібліотеки	77	Сторінка 72

0% Цитат

Вилучення цитат вимкнене

Вилучення списку бібліографічних посилань вимкнене

0%
Вилучень

Немає вилучених джерел

Модифікації

Виявлено модифікації тексту. Детальна інформація доступна в онлайн-звіті.

Підозріле форматування	13 сторінок

Завідувачу кафедри
інженерії програмного забезпечення
проф. Бедратюку Л. П.
студента групи ПЗ-19-1
Баздиковського О.В.
Прізвище, ініціали

ЗАЯВА

Прошу закріпити за мною тему кваліфікаційної роботи освітнього ступеня «бакалавр» за спеціальністю 121 «Інженерія програмного забезпечення»: Ігровий застосунок у жанрах „Платформер" та „Головоломка" з використанням інструментів Unity для розробки відеоігор.

(керівник роботи – Правдюка Наталія Іванівна)
Прізвище, ім'я, по батькові

05.02.2023
Дата

Підпис студента

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

ДЕКЛАРАЦІЯ УЧАСНИКА ОСВІТНЬОГО ПРОЦЕСУ

щодо дотримання академічної доброчесності

Цією декларацією я, Боцяновський Олексій Віталійович,

студент IV курсу спеціальності 121 – Інженерія програмного забезпечення,
група ІПЗ-19-1

здобувач вищої освіти (шифр та назва спец-ті, курс, академічна група)

підтверджую, що ознайомився (-лась) з Положенням про систему забезпечення академічної доброчесності у Хмельницькому національному університеті та Кодексом академічної доброчесності і **зобов'язуюсь** дотримуватися їх вимог під час освітнього процесу, проведення наукової діяльності, виконання організаційно-адміністративних функцій тощо.

Усвідомлюю, що у разі порушення мною принципів академічної доброчесності нестиму відповідальність перед академічною спільнотою ХНУ згідно з нормами, визначеними Положенням про систему забезпечення академічної доброчесності у Хмельницькому національному університеті, законодавства України.

05 лютого 2023 р.

Підпис

Завідувачу кафедри інженерії програмного забезпечення проф. Бедратюку Л. П.

здобувача вищої освіти

Боцяновський О. В.

Прізвище, ініціали

факультет ІТ, 4 курс, група ІПЗ-19-1

ЗАЯВА

З правилами чинного Положення «Про систему забезпечення академічної доброчесності в Хмельницькому національному університеті» від 01.07.2022, згідно з яким виявлення плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту і застосування заходів дисциплінарної та академічної відповідальності, ознайомлений. Про використання програмно-технічних засобів для перевірки кваліфікаційних робіт здобувачів вищої освіти на плагіат оповіщений та надаю свою згоду на обробку й збереження університетом моєї роботи в інституційному репозитарії університету.

Також надаю університету право на передачу моєї роботи для обробки та збереження в базах даних програмно-технічних засобів (Unicheck та/або Anti-Plagiarism) і використання роботи для виявлення академічного плагіату в інших роботах, які перевіряються програмно-технічними засобами та користувачами, що мають доступ до цих програмно-технічних засобів, виключно в обмежених цілях для виявлення плагіату в текстах робіт.

Робота для перевірки університетом надається в друкованому та електронному варіанті. Електронна версія моєї роботи збігається (ідентична) з друкованою.

31.05.2023

дата

підпис

РІШЕННЯ ЕКСПЕРНОЇ КОМІСІЇ
КАФЕДРИ ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ
ПРО ДОПУСК КВАЛІФІКАЦІЙНОЇ РОБОТИ ДО ЗАХИСТУ

Підтверджуємо ознайомлення з результатом звіту подібності щодо роботи, продукованими програмно-технічним засобом (ами) перевірки текстів на плагіат:

Назва: «Ігровий застосунок у жанрах "Платформер" та "Головоломка" з використанням інструментів Unity для розробки відеоігор»
Автор: Боцяновський Олексій Віталійович
Спеціальність: 121 – Інженерія програмного забезпечення
Освітня програма: Освітньо-професійна програма «Інженерія програмного забезпечення»
Науковий керівник: Праворська Наталія Іванівна, кандидат педагогічних наук, доцент
Після аналізу звіту подібності зроблено такий висновок:

№	Висновок	Позначка про відповідність
1	Запозичення, виявлені в роботі, є законними і не є плагіатом. Робота приймається до захисту.	**відповідає**
2	Виявлені запозичення не є плагіатом, розміщені в розділах, які не описують безпосередньо авторське дослідження, але кількість цитат перевищує обсяг, виправданий поставленою метою роботи. Робота приймається до захисту, але має бути відкоригована. Відкоригований варіант має бути поданий на кафедру за 2 дні до захисту, разом із заявою щодо самостійності виконання письмової роботи та ідентичності друкованої й електронної версії роботи	
3	Виявлені запозичення не є плагіатом, але частково розміщені в розділах, які описують безпосередньо авторське дослідження, а кількість цитат перевищує обсяг, виправданий поставленою метою роботи. В зв'язку з цим мета роботи та поставлені завдання не були досягнені. Робота може бути допущена до захисту (наступного року) після того, як буде відкоригована та допрацьована і успішно пройде повторну перевірку на академічний плагіат.	
4	Робота містить навмисні текстові спотворення, передбачувані спроби укриття запозичень або інші прояви академічного плагіату. Робота містить фабрикацію або фальсифікацію даних. Робота не допускається до захисту.	
5	Інше:	

Підтвердження:

Запозичення, виявлені в роботі, є законними і не є плагіатом, оскільки:

1) у тексті кваліфікаційної роботи системами перевірки на плагіат виявлено схожість з деякими документами в частині загальновживаних обов'язкових словосполучень у стандартних бланках (титулка, відомість документів), у структурі змісту, назвах розділів/підрозділів тощо, у назвах публікацій у переліку джерел посилання;

2) в якості запозичень системою було зафіксовано деякі послідовності вихідного коду і посилання на бібліотеки, які є стандартними мовними конструкціями програмування та не можуть розглядатися як об'єкт авторських прав і, відповідно, їх порушення;

3) усі запозичення є фрагментарними або мають належним чином оформленні посилання;

4) виявлені модифікації тексту не впливають на відсоток схожості.

Сумарний обсяг всіх запозичень, визначений системою виявлення збігів ідентичності/ схожості, складає 3,93% і адресується до 427 джерел, що, з урахуванням наведених обґрунтувань, відповідає характеру теми і свідчить на користь кваліфікаційної роботи.

Дата 6.06.23

Завідувач кафедри _____ Леонід БЕДРАТЮК

Гарант освітньої програми _____ Леонід БЕДРАТЮК

Керівник кваліфікаційної роботи _____ Наталія ПРАВОРСЬКА

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

РЕЦЕНЗІЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ
освітнього ступеня «Бакалавр»

Дипломник Боцяновський Олексій Віталійович

Тема Ігровий застосунок у жанрах "Платформер" та "Головоломка" з використанням інструментів Unity для розробки відеоігор

Спеціальність 121 – Інженерія програмного забезпечення

Обсяг кваліфікаційної роботи:

Кількість листів креслень 3 : кількість сторінок записки 66

1.Короткий зміст пояснювальної записки та прийнятих рішень У кваліфікаційній роботі було досліджено і проаналізовано предметну область, визначено усі функціональні та нефункціональні вимоги. Був проведений аналіз існуючих програм на ринку існуючих застосунків, розглянуто їх переваги і недоліки, та доведено актуальність розробки. На основі сформованих вимог спроектовано, програмно реалізовано та протестовано застосунок. Результати тестування показали, що розроблений ігровий застосунок працює коректно та готовий до використання.

2. Висновок про відповідність роботи поставленому завданню Кваліфікаційна робота виконана відповідно до поставленого завдання та з дотриманням всіх вимог.

3. Характеристика виконання кожного розділу роботи, ступінь використання останніх досягнень науки і техніки та передових методів роботи У вступі доведено актуальність теми, визначено мету та завдання дипломного проектування. У першому розділі проведено аналіз предметної області, розглянуто існуючі рішення та визначені функціональні і нефункціональні вимоги до розроблюваного програмного забезпечення. У другому розділі проведено аналіз архітектур, які можуть бути використаними, розглянуто їх переваги і недоліки та визначено, що система буде відповідати архітектурі "Entity-Component". У третьому розділі підготовлено всі залежності для написання коду та виконано практичну розробку програмних модулів і описано їх особливості, в результаті чого створено програмний продукт. Також у цьому розділі виконано тестування застосунку з метою визначення його відповідності функціональним вимогам.

4. Позитивні сторони роботи Тематика кваліфікаційної роботи є актуальною, оскільки розробка ігрових застосунків є перспективною, прибутковою та конкурентоспроможною галуззю індустрії розробки програмних продуктів. Для розробки використані новітні технології та засоби проектування, програмування і тестування.

5. Негативні сторони роботи У роботі через обраний графічний стиль виникає складність з ідентифікацією об'єктів. Також в програмному продукті є недостатня кількість підказок для користувача, а тому бажано було б додати більше підказок для вирішення головоломок.

6. Оцінка графічного оформлення та пояснювальної записки Графічне оформлення виконано відповідно до теми кваліфікаційної роботи та подано у вигляді діаграм і рисунків. Пояснювальна записка оформлена згідно вимог чинних стандартів.

7. Відгук про кваліфікаційну роботу в цілому Кваліфікаційна робота заслуговує позитивної оцінки. Матеріал пояснювальної записки структурований, послідовний, чіткий та простий, що дозволяє чітко зрозуміти викладений матеріал у рамках тематики проектування. Графічний матеріал дає можливість наочно побачити деталі проектування системи.

8. Інші зауваження

9. Оцінка кваліфікаційної роботи Кваліфікаційна робота виконана у повному обсязі, відповідає поставленій задачі та заслуговує на оцінку «відмінно».

РЕЦЕНЗЕНТ Мартинюк Валерій Володомирович, професор, доктор технічних наук, зав. кафедри автоматизації, комп'ютерно-інтегрованих технологій та робототехніки ХНУ.

«6» 06 202__ р. _____
(підпис)