Хмельницький національний університет
Факультет інформаційних технологій
Кафедра інженерії програмного забезпечення

КВАЛІФІКАЦІЙНА РОБОТА

Кучменко Костянтин Юрійович
Прізвище, ім'я, по батькові студента(ки)

на здобуття ступеня вищої освіти Бакалавра

Ігровий застосунок у жанрі "Платформер" з інтерфейсом управління на основі голосової взаємодії з використанням технологій Unity
Назва теми

Галузь знань 12 «Інформаційні технології»

Спеціальність 121 «Інженерія програмного забезпечення»

Освітня програма Освітньо-професійна програма «Інженерія програмного забезпечення»

Шифр КвРІПЗ. 200166.01.13.ПЗ

Виконав студент IV курсу, група ІПЗ-20-1 _____ Костянтин КУЧМЕНКО
Підпис Ім'я, ПРІЗВИЩЕ

Керівник канд. пед. наук, доцент _____ Наталія ПРАВОРСЬКА
Науковий ступінь, вчене звання Підпис Ім'я, ПРІЗВИЩЕ

Нормоконтролер ст. викладач _____ Ганна БЕДРАТЮК
Посада Підпис Ім'я, ПРІЗВИЩЕ

До захисту допускаю:
Завідувач кафедри інженерії програмного забезпечення _____ Леонід БЕДРАТЮК
Підпис Ім'я, ПРІЗВИЩЕ

6 червня 2024 р.

Хмельницький 2024

6. Консультанти розділів кваліфікаційної роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	Безратюк Г. І.	06.06.24	06.06.24
Антиплагіат	Форкун Ю. В., доцент	30.05.24	06.06.24

7. Дата видачі завдання « 02 » січня 2024 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 задач та вимог, розробка технічного завдання	01.01 – 20.02.2024	
3 Проектування програмного забезпечення	21.02 – 20.03.2024	
4 Програмна реалізація з використанням відповідних засобів розробки. Тестування ПЗ	21.03 – 30.04.2024	
5 Написання вступу, загальних висновків, оформлення джерел посилання та додатків. Оформлення пояснювальної записки КвР згідно вимог стандартів	01.05 – 25.05.2024	
6 Попередній захист КвР	Травень 2024	Згідно графіка
7 Перевірка КвР на плагіат, нормоконтроль, отримання відгуків, рецензій та інших супровідних документів. Брошурування (зшиття) пояснювальної записки.	26.05 – 30.05.2024	
8 Здача КвР на кафедру; підготовка КвР для розміщення у репозитарії ХНУ; підготовка до захисту та захист КвР	з 01.06.2024	

Студент _____ Костянтин КУЧМЕНКО
Підпис Ім'я, ПРІЗВИЩЕ

Керівник роботи _____ Наталія ПРАВОРСЬКА
Підпис Ім'я, ПРІЗВИЩЕ

АНОТАЦІЯ

Тема кваліфікаційної роботи: Ігровий застосунок у жанрі "Платформер" з інтерфейсом управління на основі голосової взаємодії з використанням технологій Unity.

Автор проекту: Кучменко Костянтин Юрійович.

Керівник проекту: Праворська Наталія Іванівна.

Пояснювальна записка: 55 с., 31 рис., 0 табл., 3 дод., 40 джерел.

Графічна частина: 17 слайдів.

UNITY, C#.

Метою проекту є створення інноваційного ігрового застосунку, який поєднує жанр "Платформер" та голосову взаємодію, що забезпечить новий рівень захопленості та доступності для гравців.

У кваліфікаційній роботі було проведено аналіз цільової аудиторії, дослідження можливостей технологій Unity, визначено оптимальний підхід до розробки програмного забезпечення, спроектовано і протестовано застосунок, та підведено підсумки.

Для розробки було використано технології Unity та голосову взаємодію.

В результаті було створено ігровий застосунок у жанрі "Платформер" з голосовим управлінням, який має дружній інтерфейс, підтримує різноманітні функціональні можливості, такі як: подолання викликів та перешкод, ворожі атаки, битви з босами, головоломки, що змушують гравця обмірковувати свої дії та приймати рішення.

06.06.24
Дата

Підпис

ВІДОМІСТЬ ДОКУМЕНТІВ

№ рядка	Формат	Позначення документа	Найменування документа	К-сть аркушів	№ екз.	Примітка
			Текстові документи			
1	А4	КвРІПЗ.200166.01.13.ПЗ	Пояснювальна записка	94		
2	А4		Завдання на кваліфікаційну роботу	1		
3	А4		Анотація	1		
			Графічні документи			
4	А4		Презентаційні матеріали	17		
5	А3	КвРІПЗ.200166.01.13.ПЗ	**Діаграма вікон інтерфейсів**	1		
6	А3	КвРІПЗ.200166.01.13.ПЗ	**ER Діаграма класів**	1		
7	А3	КвРІПЗ.200166.01.13.ПЗ	**UML Діаграма варіантів використання**	1		

Змн.	Арк.	№ докум.	Підпис	Дата	КвРІПЗ. 200166.01.13.ПЗ			
Виконав		Кучменко К.Ю.		06.06	Ігровий застосунок у жанрі "Платформер" з інтерфейсом управління на основі голосової взаємодії з використанням технологій Unity	Літ.	Арк.	Аркушів
Керівник		ПраворськаН.І.		06.06			1	1
Н. Контр.		Бедратюк Л.П.		06.06		ХНУ, ІПЗ-20-1		
Зав. Каф.		Бедратюк Л.П.		06.06				

ЗМІСТ

					КвРІПЗ. 200166.01.13.ПЗ			
Змн.	Арк.	№ докум.	Підпис	Дата				
Виконав		Кучменко К.Ю.		06.06.24	Ігровий застосунок у жанрі "Платформер" з інтерфейсом управління на основі голосової взаємодії з використанням технологій Unity	Літ.	Арк.	Аркушів
Керівник		ПраворськаН.І		06.06.24				
				06.06.24		ХНУ, ІПЗ-20-1		
Н. Контр.		Бедратюк І.І		06.06.24				
Зав. Каф.		Бедратюк Л.П.		06.06.24				

ВСТУП

В ігровій сфері відзначається важливий вплив на суспільство, що сприяє розвитку різних навичок, таких як стратегічне мислення, співпраця та креативність. Дослідження підтверджують, що відеоігри покращують когнітивні функції, соціальні навички і навіть допомагають зменшити рівень стресу. Розробка новаторських ігор, які сприяють розвитку користувачів, є ключовим аспектом ігрової індустрії, і до таких ігор відносяться ті, що використовують можливість голосової взаємодії з ігровим світом.

Актуальність теми "розробка ігрового застосунку "Платформер" з голосовим управлінням на базі Unity" визначається насамперед наступними чинниками:

– Інноваційність – поєднання жанру "Платформер" з голосовим управлінням відкриває нові можливості для ігрової індустрії, створюючи інноваційний та захоплюючий геймплей;
– популярність – жанр "Платформер" вже має широку аудиторію фанатів, тому розробка нової гри з використанням цього жанру приверне увагу багатьох гравців.

У сучасному світі використання голосової взаємодії та ігрових технологій стає все більш поширеним і важливим аспектом розвитку інтерактивних систем. Дана дипломна робота присвячена створенню ігрового застосунку у жанрі "Платформер", заснованому на голосовій взаємодії з використанням технологій Unity.

Жанр "Платформер" [1] – є одним із найпопулярніших у світі відеоігор. У цій грі гравець має керувати персонажем, який мусить подолати різноманітні виклики та перешкоди. Серед них вважаються:

– ворожі атаки, що варіюються за типом та інтенсивністю;
– битва з великим босом, який зазвичай включає в себе кілька фаз;
– різноманітні види атак;

що робить гру надзвичайно складною і змушує гравця проходити рівень заново, а також спонукає вивчати послідовність і типи цих атак, щоб пройти далі. Крім того, в грі також присутні різноманітні головоломки, які змушують гравця ретельно обмірковувати свої дії та приймати рішення.

Додавання голосового інтерфейсу управління може значно підвищити рівень захопленість гри та зробити її більш доступною для гравців. Це надасть новий рівень взаємодії, забезпечуючи користувачам неповторний досвід від ігрового процесу. Впровадження такої технології не лише робить гру цікавішою, а й відкриває нові можливості для взаємодії з віртуальним світом.

Вибір платформера для цього дослідження зумовлений його популярністю та широким спектром можливостей для цікавих геймплейних концепцій. Гра у платформера часто вимагає від гравця точності та швидкості у рухах, що може забезпечити захоплюючий досвід, особливо коли поєднується з інноваційним керуванням через голос.

У цьому дослідженні все було спрямовано на створення ігрового застосунку, який поєднує в собі жанр "Платформер" та інтерфейс управління на основі голосової взаємодії. Мета полягає в ретельному вивченні та аналізі цільової аудиторії, дослідженні можливостей технологій Unity для реалізації цього проекту та визначенні оптимального підходу до розробки програмного забезпечення який надасть гравцям новий ігровий досвід.

Об’єктом дослідження даної тематики є розробка ігрового застосунку у жанрі "Платформер" з інтерфейсом управління на основі голосової взаємодії з використанням технологій Unity.

Предметом дослідження є процес розробки та реалізації ігрового застосунку з урахуванням технічних та дизайнерських аспектів, а також аналіз його ефективності та прийняття вирішальних рішень щодо використання голосової взаємодії для керування гравцем.

Головною метою даного дослідження є розробка та реалізація ігрового застосунку у жанрі "Платформер" з інтерфейсом управління на основі голосової

взаємодії, який буде забезпечувати користувачам новий та захоплюючий досвід гри.

Для досягнення поставленої мети необхідно вирішити наступні основні завдання:

– вивчити предметну область;

– провести аналіз існуючих рішень у галузі голосової взаємодії та ігрових технологій;

– визначити вимоги до програмного забезпечення;

– вибрати та використати відповідні інструменти та технології для розробки програмного забезпечення;

– розробити прототипи та діаграми, що відображатимуть принципи роботи програмного продукту;

– реалізувати програмний продукт та провести його тестування для виявлення та виправлення можливих помилок.

1 ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Змістовний аналіз предметної області, її структурних та функціональних особливостей

Ігрова індустрія і надалі має значний вплив на суспільство, відкриваючи нові можливості для розвитку різних навичок та вмінь, від стратегічного мислення до співпраці та креативності. Недавні дослідження підтверджують, що відеоігри допомагають у покращенні когнітивних функцій, розвивають соціальні навички та навіть можуть допомогти знизити рівень стресу. Отже, створення ігор, спрямованих на розвиток користувачів, вважається важливим аспектом в ігровій індустрії.

Програмне забезпечення можна вважати грою, якщо воно відповідає наступним критеріям:

– взаємодія з користувачем – програма повинна мати відкритий чи закритий інтерфейс, який дозволяє користувачу взаємодіяти з нею. Це може бути через графічний інтерфейс, текстовий інтерфейс або, в разі ігор, через візуальні або звукові ефекти, до прикладу натискання на клавіатурі кнопки space дасть змогу гравцеві підскочити що зображено на рисунку 1.1;

Рисунок 1.1 – UML діаграма станів з демонстрацією одного із видів взаємодії з користувачем

– цілеспрямована діяльність – програмне забезпечення повинне мати визначену мету або завдання, яке користувач повинен виконати або яке він може вплинути на подальший розвиток ситуації. Нижче наведений приклад кількох цілей, а саме: виконати всі квести в грі, пройти від точки А до точки Б, після переходу з точки А до точки Б на точці Б буде чекати бос, якого потрібно перемогти, після чого перемоги над босом перейти на наступний рівень. Це можна продемонструвати на діаграмі станів, яка показана на рисунку 1.2;

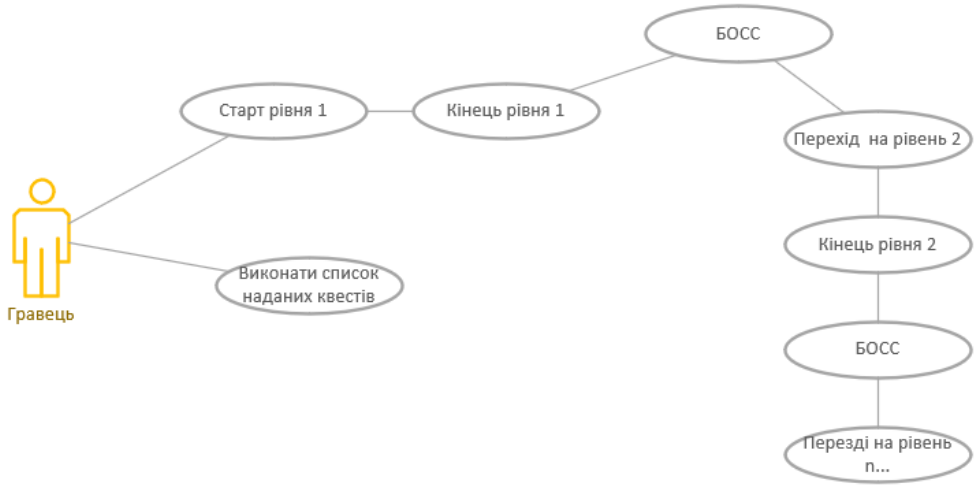


Рисунок 1.2 – UML діаграма станів з демонстрацією кількох можливих цілей в грі

– елементи виклику – гра повинна містити елементи виклику або складності, які стимулюють користувача до активної участі та розв'язання проблем, до прикладу виконання тих ж самих квестів або до прикладу перемога над босом і отримання речей за це, або бонусів, більше детально на рисунку 1.3;

Рисунок 1.3 – UML діаграма станів з демонстрацією мотивації гравця грати далі

– елементи розваги або задоволення – гра повинна надавати користувачеві задоволення або розвагу. Це може бути через веселі сценарії, відчуття досягнень, конкуренцію з іншими користувачами або просто приємні візуальні або аудіо ефекти приклад на рисунку 1.4;

Рисунок 1.4 – UML діаграма станів з демонстрацією можливих розваг в такому роді ігор

– можливість впливати на результат – користувач повинен мати можливість впливати на хід подій або результати гри своїми діями або рішеннями, до прикладу діалог з представником поселення рисунок 1.5.

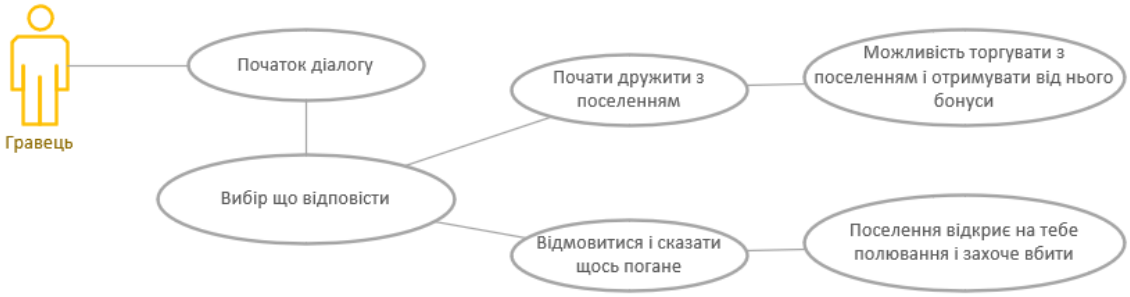


Рисунок 1.5 – UML діаграма станів з демонстрацією того, як може працювати вибір гравця на хід подій у грі

– прогресія – гра повинна мати систему прогресії, що дозволяє користувачеві розвиватися, отримувати нові можливості або підвищувати рівень влади над середовищем гри приклад продемонстрований на рисунок 1.6.

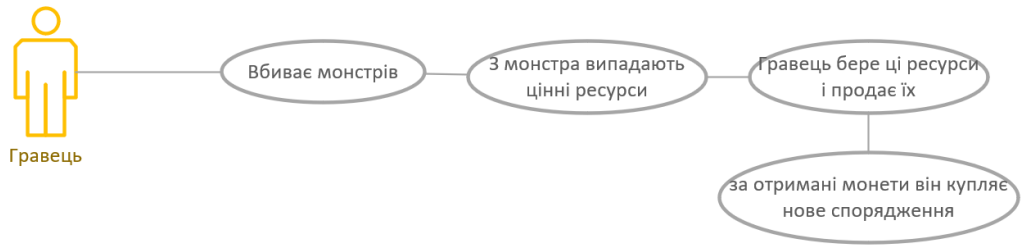


Рисунок 1.6 – UML діаграма станів з демонстрацією того, як може працювати прогресія в грі

Якщо говорити про той жанр, на якому буде базуватися розроблюваний застосунок з голосовим керуванням, варто звернутися до основних особливостей цього жанру:

– головний персонаж – гравець керує головним персонажем гри, який зазвичай є фіксованим персонажем з власними характеристиками та навичками;

– рівні або етапи – гра поділена на різні рівні або етапи, які гравець має пройти послідовно. Кожен рівень може мати унікальний дизайн, виклики та обставини;

– платформи та перешкоди – гравець має подолати різноманітні платформи та перешкоди, такі як, рухомі платформи, шипи, ворожі персонажі тощо;

– колекційні предмети – у різних місцях рівня можуть розташовуватися різні колекційні предмети, такі, як монети, зірки, кристали тощо, які гравець може збирати для отримання бонусів або досягнень;

– ворожі персонажі – гравець може зіткнутися з ворожими персонажами або ворожими об'єктами, які йому слід уникати або подолати для продовження гри;

– боси – на кінці деяких рівнів може зустрітися великий і сильний бос, з яким гравець повинен поборотися для завершення рівня;

– головоломки – деякі рівні можуть містити головоломки або складні завдання, які гравець має розв'язати, щоб продовжити гру;

– підйомники та інші механічні пристрої – різноманітні механічні пристрої, такі як підйомники, трампліни, телепорти тощо, можуть допомагати гравцю подолати перешкоди або дістатися до важкодоступних місць;

– персоналізація героя – персоналізація свого головного персонажа шляхом вибору з різних костюмів, аксесуарів або навіть характеристик;

– синхронізація з іншими пристроями, у випадку якщо це кросс платформена гра – синхронізування прогресу з іншими пристроями, щоб вони могли грати на своєму смартфоні, планшеті або навіть консолі, продовжуючи гру з того місця, де вони зупинилися.

Після ретельного аналізу усіх зазначених аспектів виявлено, що найбільш оптимальним варіантом для даного проекту є розробка гри у жанрі "Платформер" з використанням голосової взаємодії. Цей вибір обґрунтовується великим різноманіттям можливостей, що притаманні цьому жанру, які ідеально доповнюють потенціал голосового керування.

Основна механіка гри полягатиме у керуванні головним персонажем за допомогою голосових команд та інших аудіо-інструкцій. З цим у виду, планується включити різноманітні перешкоди та прості головоломки, які гравець зможе подолати або розв'язати, використовуючи голосове керування.

Також варто врахувати важливість введення системи досягнень, яка стимулюватиме гравців до подальшого проходження рівнів та виконання додаткових завдань. Ці досягнення можуть бути пов'язані з різноманітними викликами, такими як збір певної кількості колекційних предметів, завершення рівнів за мінімальний час або подолання всіх перешкод без помилок.

Крім того, можна впровадити систему персоналізації гравця, яка буде даватись за досягнення певних витоків у грі. Наприклад, гравець може отримати нові костюми або аксесуари для свого персонажа після успішного завершення певної кількості рівнів або досягнення визначених цілей. Це дозволить гравцям персоналізувати свого героя і створити унікальний образ, що відображатиме їхні досягнення та стиль гри.

Такий підхід гарантує захопливий геймплей, що поєднує інноваційне голосове керування з класичними елементами платформера, забезпечуючи гравцям неповторний досвід і задоволення від ігрового процесу.

Розробка також була висвітлена в тезах Всеукраїнської науково-практичної конференції «Актуальні проблеми комп’ютерних наук АПКН-2023». Кучменко К.Ю., Праворська Н.І. Ігровий застосунок у жанрі – “Платформер” з інтерфейсом управління на основі голосової взаємодії з використанням технологій unity. Збірник наукових праць за матеріалами XV Всеукраїнської науково-практичної конференції «Актуальні проблеми комп’ютерних наук АПКН-2023». Хмельницький. 2023. – 157-160 с.

1.2 Аналіз наявного програмно-технічного забезпечення предметної області

На даний момент на ринку не так багато ігор, які використовують таку технологію, як голосове керування, а особливо мало тих, хто це використовує в таких ігрових жанрах, як платформери. Але все ж таки є кілька популярних ігор, які використовують цю технологію і доволі успішно. Перша гра, яка використовує таку технологію являється Scream Go Hero [2] зображено на рисунку 1.7.

Гра максимально проста і правила в ній також максимально прості, чим гучніше кричить користувач, тим вище буде підстрибувати персонаж і все. Все що потрібно, це переміщатися по платформах, ухилятись від ворогів та пасток і набивати бали, чим більше тим краще, також є валюта – вишня, за яку можна купити кастомізацію в грі.

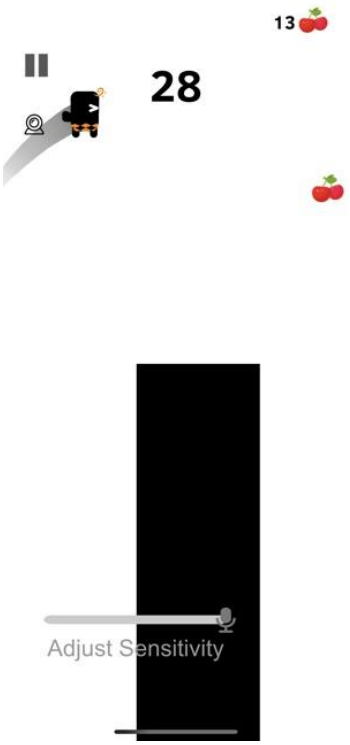


Рисунок 1.7 – Момент з геймлею в грі Scream Go Hero

Переваги гри:

– простота: гра має простий дизайн і максимально просту суть – набрати найбільше можливо балів і потрапити в топ рейтингу. Також можна вважати, що, якщо користувач відкрив всі елементи кастомізації, то гру він пройшов;

– першопроходець на мобільному ринку ігор з голосовим керуванням: оскільки він є першопрохідцем, це означає, що він зайняв вигідну для себе нішу.

Наступною грою в цьому сегменті є "One Hand Clapping" [3]. Гра доступна для персональних комп'ютерів і може бути придбана в магазині Steam. "One Hand Clapping" – це вокальний 2D платформер. Розв'язуйте головоломки, співаючи або використовуючи мікрофон, і відчуйте силу свого голосу, який змінює оточуючий світ. Гра є релаксуючою та надихаючою платформер-головоломкою, де основний акцент робиться на вокалі для подальшого просування у яскравому світі. Посилюйте впевненість у своєму голосі, використовуючи мелодію, ритм і гармонію як інструменти. Не поспішайте. Користувачу немає нічого втрачати і ніхто його не покарає за помилку.

"One Hand Clapping" має декілька переваг порівняно з іншими іграми на ринку:

– унікальний геймплей – гра використовує вокал, як ключовий механізм для розв'язання головоломок і просування в грі, що робить її особливою серед інших платформерів;

– новаторський підхід до взаємодії з гравцем – за допомогою мікрофону гравець може взаємодіяти з оточенням і вирішувати завдання, співаючи або виражаючи різні звукові варіанти, що дозволяє створити унікальний іммерсивний досвід;

– релаксуючий характер – гра пропонує спокійний геймплей, сприяючи відпочинку і релаксації. Вона відрізняється від більшої частини інших ігор, які часто базуються на швидкості та адреналіні;

– сприяння самовираженню – "One Hand Clapping" дає гравцям можливість використовувати свій голос, як інструмент для розв'язання завдань, що створює можливість для творчого самовираження та вираження особистості, до прикладу, гравець може завдяки тембру свого голосу побудувати шлях на

вверх у грі і подолати пустоту і добратися до потрібного місця, що можна побачити на рисунку 1.8;

Рисунок 1.8– Механіка будівництва шляху завдяки голосу у грі One Hand Clapping

– позитивний настрій – завдяки яскравій графіці, що можна побачити на рисунку 1.9, музичному супроводу та цікавим головоломкам, гра викликає позитивні емоції у гравців, що робить її привабливою для широкого аудиторії.

Рисунок 1.9 – Демонстрація графіки в грі One Hand Clapping

Ще одна гра "Phasmophobia" [4]. Гра у жанрі хорор-дослідження з виживанням, яку грають з першої особи. Гравець працює самостійно або у групі

з до трьох інших гравців, щоб виконати контракт, в якому вони повинні визначити тип привиду, який мешкає на вказаному місці. Гравці можуть спілкуватися за допомогою голосового чату, як локально на короткі відстані, так і глобально за допомогою радіотелефонів. У грі "Phasmophobia" є розпізнавання мови, що дозволяє певним пристроям, проклятим предметам і навіть привиду слухати гравців, розуміти ключові слова та фрази. Як гра виглядає можна побачити на рисунку 1.10.

Рисунок 1.10 – Демонстрація моменту з гри Phasmophobia

"Phasmophobia" має кілька переваг порівняно з подібними іграм, які використовують голосову взаємодію:

– реалістичність – використання голосового чату сприяє більш реалістичному досвіду, оскільки гравці можуть вільно обговорювати стратегії та реагувати на події в грі, як це зробили б у реальному житті;

– інтерактивність – голосова взаємодія дозволяє вбудувати елементи інтерактивності, такі як розпізнавання мови, що робить ігровий процес більш цікавим та цікавим для гравців;

– тактичність – здатність спілкуватися з товаришами в грі дає можливість використовувати тактичні прийоми та координацію дій, що дозволяє збільшити шанси на успішне виконання завдань.

Підсумовуючи проведений аналіз конкуренції на ринку ігор, які використовують подібну технологію у вигляді голосового керування, важливо зазначити, що вони відрізняються кардинально. Деякі можуть базуватися тільки на одній механіці, де керування персонажем здійснюється за допомогою звуків, тоді як інші, мають кілька механік, які спрямовані на керування голосом, такі як зміна позицій платформ або будівництво нових платформ за допомогою голосу, що робить гру набагато цікавішою.

Серед проаналізованих ігор все ж таки можна виділити одну цікаву механіку, а саме керування персонажем за допомогою звуків.

1.3 Постановка задач та вимог до програмного продукту

В результаті аналізу предметної області та вивчення переваг та недоліків існуючих рішень, було визначено, що дана гра у жанрі платформера, що використовує голосове керування, буде мати наступний функціонал і буде привабливою для різноманітної аудиторії від дітей до дорослих:

– голосове керування головним персонажем – гравець зможе керувати рухами головного персонажа, стрибати, за допомогою голосових команд, а точніше завдяки різноманітним звукам;

– різноманітність перешкод та головоломок – у грі буде присутній різноманітний набір перешкод та простих головоломок, які додадуть глибини та складності геймплею, збагачуючи ігровий досвід;

– висока доступність для аудиторії – голосове керування спростить взаємодію з грою, зробивши її доступною для різних вікових груп, включаючи дітей та дорослих;

– привабливий геймплей – поєднання класичного жанру платформера з інноваційним голосовим керуванням створить захоплюючий геймплей, який зацікавить широкий коло гравців;

– ігровий досвід – висока якість графіки та звукового супроводу дозволить створити атмосферу, яка захопить гравця в світ гри.

Для ще більшої зацікавленості гравців у гру планується впровадити систему кастомізації, яка буде розблоковуватися по мірі проходження рівнів. Наприклад, на кожному 2-3 рівні гравець матиме можливість вибрати новий аксесуар для свого головного персонажа. Це дозволить гравцям налаштовувати свого героя та додавати в гру елементи індивідуальності.

Також буде впроваджена система зіркового рейтингу, яка дозволить гравцям перевіряти свій рівень професіоналізму у грі. Після завершення кожного рівня гравець отримає від 1 до 3 зірок, в залежності від його успішності. Це стимулюватиме гравців до досягнення кращих результатів та підвищення свого рівня майстерності.

Крім того, для тих, хто любить виклики та загадки, будуть доступні секретні рівні. Щоб їх відкрити, гравцям потрібно буде виконати певні секретні завдання чи квести, про які вони не будуть знати наперед. Це додатково збільшить інтригу та зацікавленість гравців у грі. За демонстрацію досягнень буде створений список секретних завдань, спочатку вони будуть приховані і користувачу потрібно буде по різному взаємодіяти з грою, щоб зрозуміти що потрібно зробити, і коли користувач це зробить, то вони будуть відкриватись у цьому списку.

Такі нововведення роблять ігровий процес більш динамічним, цікавим і різноманітним, що сприятиме збільшенню захоплення гравців та подальшому успіху гри.

1.4 Висновки дослідження предметної області та постановки задачі

У першому розділі було проведено ґрунтовний аналіз предметної області, зосередившись на жанрі "Платформер" та його особливостях у контексті розробки гри з голосовим керуванням. Було визначено, що цей жанр оптимально підходить для реалізації проекту завдяки його динамічності, різноманітності та потенціалу для інтеграції голосової взаємодії.

Описано основні механіки гри, які ґрунтуються на голосових командах та аудіо-інструкціях для керування персонажем, подолання перешкод та вирішення головоломок.

У другому розділі було проаналізовано ринок існуючих ігор де було визначено що на ринку існують ігри, які використовують голосове керування, але їх кількість обмежена, особливо в жанрі "Платформер". Проаналізовано декілька популярних прикладів, таких як Scream Go Hero, One Hand Clapping та Phasmophobia.

Scream Go Hero пропонує простий геймплей, де керування персонажем здійснюється за допомогою гучності голосу. Гра має елементи кастомізації та орієнтована на набір максимальної кількості балів.

One Hand Clapping відрізняється унікальним геймплеєм, де розв'язання головоломок та просування в грі залежать від вокальних здібностей гравця. Гра має релаксуючий характер, сприяє самовираженню та використовує яскраву графіку.

Phasmophobia використовує голосовий чат для реалістичного досвіду та командної взаємодії. Гра має елементи розпізнавання мови, що робить її більш цікавою та інтерактивною.

Аналіз ринку свідчить про те, що існує потенціал для розробки нових і цікавих ігрових механік з використанням голосового керування.

У третьому розділі було визначено, який саме функціонал має бути у грі для того, щоб зацікавити користувача. До цього функціоналу входять такі елементи як: голосове керування головним персонажем, різноманітні перешкоди та головоломок, система кастомізації персонажа, секретні рівні.

2 ПРОЄКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Проєктування архітектури та структури системи

У цьому розділі будуть детально розглядатися всі аспекти проєктування, починаючи від аналізу вимог і закінчуючи розглядом можливостей масштабування та розширення системи. Мета роботи – розробити чіткий та ефективний план архітектури [6], який відповідатиме потребам та цілям проекту. У цьому контексті розглянемо такі аспекти, як вибір архітектурних шаблонів, визначення компонентів системи, моделювання взаємодії між ними, а також забезпечення безпеки та надійності програмного забезпечення. Цей розділ буде важливим кроком у створенні високоякісного та функціонального програмного продукту, який задовольнить вимоги представленого проекту.

Першим, що потрібно розглянути те, як буде працювати основна механіка гри, а саме механіка керування персонажем завдяки голосовому керуванню. Архітектура гравця полягає в організації різних компонентів програми, таких як механіка руху персонажа, обробка вхідних подій (у тому числі голосового керування), управління станом гравця, обробка зіткнень та візуалізація. Основні компоненти включають:

а) Player – головний клас, що буде представляти гравця у грі. Він відповідає за рух персонажа, обробку вхідних подій (натискання кнопок миші або голосове керування), анімацію [15] яка буде працювати завдяки Unity animation [16] та інші дії гравця, блок-схема [5] як працює гравець зображена на рисунку 2.1;

Рисунок 2.1 – Блок-схема класу Player

б) GameController – цей клас буде відповідати за управління грою в цілому. Він включає логіку перемоги, поразки, паузи тощо (рисунок 2.2);

Рисунок 2.2 – Блок-схема класу GameController

в) SettingsController – клас, що буде керувати налаштуваннями гри, такими як чутливість мікрофона для голосового керування (рисунок 2.3);

Рисунок 2.3 – Блок-схема класа SettingsController

г) ShopMenu – клас, який буде відповідати за функціонал магазину в грі. Ось його основні функції які передбачаються у ньому:

1) ініціалізація магазину – під час запуску гри магазин ініціалізується, відображаючи першу сторінку товарів та інші елементи інтерфейсу;

2) відкриття товарів – товари будуть відкриватись поступового, в залежності від кількості рівнів які пройде гравець;
3) відображення товарів – здійснює відображення доступних товарів у магазині, які можна придбати;
4) навігація по сторінках магазину – гравець може переглядати різні сторінки магазину з допомогою кнопок вправо і вліво;
5) закриття магазину – забезпечує можливість закрити вікно магазину після придбання товарів або в будь-який момент;
6) анімаційні ефекти – використовуються анімаційні ефекти для плавного переходу між сторінками магазину та зміни розміру елементів інтерфейсу;
7) обробка звуків – відтворення звуків під час взаємодії користувача з магазином;

д) ChoseHat – це клас, який буде відповідати за вибір головного убору гравцем в грі, ось його основні функції які передбачаються у ньому:

1) збереження інформації про вибрані капелюхи – функція, яка буде зберігати інформацію про вибрані капелюхи за допомогою PlayerPrefs [25];
2) відображення вибраних капелюхів – відображення піктограм та тексту для кожного капелюха, в меню магазину, це потрібно для того щоб гравець візуально розумів що він вибрав капелюх;
3) відображення вибраних капелюхів – відображення піктограм та тексту для кожного капелюха, в меню магазину, це потрібно для того, щоб гравець візуально розумів що він вибрав капелюх;

е) SettingMenu – клас, який буде відповідати за управління налаштуваннями звуку в головному меню гри, ось його основні функції які передбачаються у ньому:

1) ініціалізація налаштувань звуку – ініціалізація налаштувань збережених завдяки PlayerPrefs;

2) оновлення налаштувань – викликатиме функцію, яка буде кожен кадр оновлювати відображення кнопок звуку відповідно до змін у PlayerPrefs;
3) керування звуком – буде функціонал, який дасть можливість включати і відключати звук, при кожній зміні буде змінюватись статус збережених даних і вони будуть перезаписуватись для того, щоб зберегти нові налаштування для звука.

Блок-схема, яка описує функціонал класу SettingMenu зображена на рисунку 2.4.

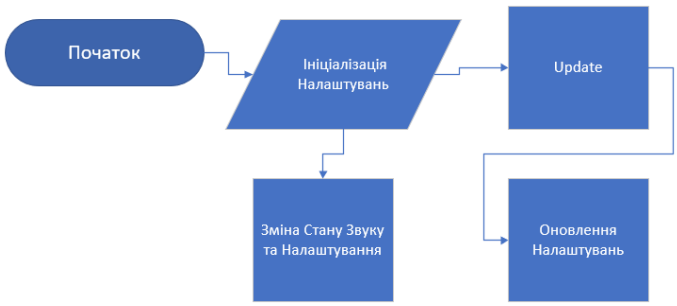


Рисунок 2.4 – Блок-схема класу SettingMenu

є) LevelUnLockSystem – клас, який буде керувати доступністю рівнів у грі, розблоковуючи їх для гравця при досягненні певних умов або прогресу у грі, ось його основні функції які передбачаються у ньому:

1) ініціалізація розблокованих рівнів – отримується значення розблокованих рівнів з PlayerPrefs, усі кнопки (або інші інтерактивні елементи) для переходу на наступні рівні, які зберігаються в масиві buttons, та графічні об'єкти з зображенням замку, які зберігаються в масиві LockObject, спочатку встановлюються у стані заблокованого рівня (або замку);
2) розблокування рівнів – за допомогою циклу, перебираються всі рівні до значення levelUnLock. Для кожного рівня встановлюється можливість взаємодії з кнопкою (чи іншим інтерактивним

елементом) та вимикається відображення замка. Таким чином, гравець може вибирати і грати рівні, які вже розблоковані.

ж) LevelDone – буде відповідати за виконання дій, які пов'язані з завершенням рівня гравцем. Основні функції та їх опис:

1) реакція на зіткнення гравця з об'єктом рівня – коли гравець зіткнувся з об'єктом, що відповідає за завершення рівня, виконуються наступні дії: розблоковується рівень (якщо це не секретний рівень), показується панель завершення рівня, відтворюється звук порталу, запускається анімація розблокування зірок;
2) анімація розблокування зірок – якщо гравець заробив одну, дві або три зірки, відбувається відповідна анімація. Анімація включає: збільшення розміру зірки і переходу до наступного рівня;
3) розблокування наступного рівня – після завершення поточного рівня має перевірятись, чи було досягнуте нове найвище значення рівня у грі, якщо так, то цей рівень розблоковується для гри.

з) StarOnLevelMenu – має відповідати за відображення кількості зірок на меню рівнів гри. Ось більш архітектурний опис функціоналу класу:

1) шніціалізація – при запуску гри викликається метод Start [21], який отримує кількість зірок з локального сховища за ключем і містить номер рівня;
2) управління зірками – буде містити метод, який перевіряє кількість зірок, збережених в локальному сховищі за номером рівня, в залежності від кількості зірок, відповідні об'єкти зірок у графічному інтерфейсі вмикаються або вимикаються;
3) збереження кількості зірок – при завершенні рівня кількість зірок зберігається в локальному сховищі під ключем, що містить номер рівня.

і) Trap_n – пастки будуть розділені на кілька класів, кожен з яких матиме назву Trap_n, де n – це унікальний ідентифікатор пастки, оскільки їх буде багато. Ці класи включатимуть пастки, що активуються внаслідок колізій, а також ті, які постійно оновлюються. Наприклад, пастка може переміщатися від однієї точки до іншої.

Блок-схема, яка описує функціонал класу Trap_n зображена на рисунку 2.5.

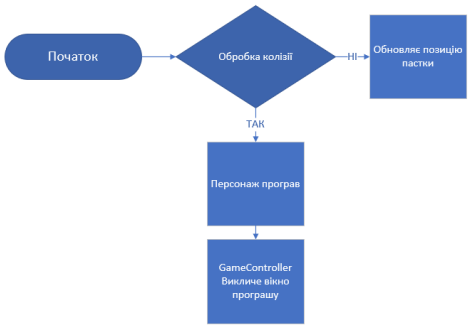


Рисунок 2.5 – Блок-схема класу Trap_n

ї) SecretManager – клас, який дозволятиме гравцеві відкривати та переглядати секретні рівні гри, а також керує їх доступністю в головному меню. Ось більш архітектурний опис функціоналу класу:

1) ініціалізація секретних рівнів – встановлення початкових значень для секретних рівнів. Завантаження збережених даних про стан секретних рівнів з пам'яті;
2) оновлення стану кнопок і блокування – перевірка стану кожного секретного рівня. Визначення, чи рівень є доступним для гравця. Активування або деактивування відповідних кнопок та об'єктів блокування;
3) відкриття секретного рівня – буде створений метод, що дозволяє відкрити певний секретний рівень. Збереження інформації про стан відкритості рівня в пам'яті гри.

й) Secret_n – у всіх класах секретів назви будуть виглядати як Secret_n, де n – порядковий номер секрету, оскільки їх буде багато. Секрети будуть розділятися на ті, які активуються через зіткнення, та ті, що стають доступними при досягненні певного стану. Наприклад, якщо гравець програє 100 разів, він отримає доступ до секретного досягнення та секретного рівня.

Блок-схема, яка описує функціонал класу Secret_n зображена на рисунку 2.6.

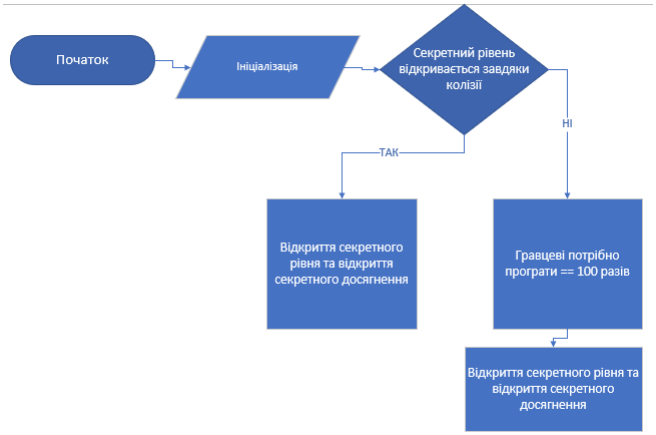


Рисунок 2.6 – Блок-схема класу Secret_n

к) MainMenu – буде відповідати за управління головним меню гри. Основні функції цього класу включатимуть відкриття і закриття різних меню, навігацію між рівнями, відкриття магазину, налаштувань та інших елементів:

1) меню головного вікна – кнопки для відкриття різних меню (налаштування, магазин, тощо). Анімація відкриття та закриття головного меню;
2) меню налаштувань – можливість зміни налаштувань гри (наприклад, звук);
3) навігація між рівнями – можливість переходу до рівнів гри з головного меню. Анімація відкриття та закриття меню вибору рівнів;

4) меню "Що це" – відображення інформаційних карток з описом різних елементів гри. Навігація між інформаційними картками;
5) меню магазину – відображення доступних товарів для придбання;
6) взаємодія з гравцем – обробка вибору рівня гравцем. Відображення та оновлення інформації про рівні, які гравець може відкрити;
7) інші функції – вихід з гри, відкриття посилання на Instagram, відображення та управління списком рівнів гравця.

2.2 Проєктування інтерфейсу користувача

В Unity за створення інтерфейсу та відповідає елемент canvas [9], це той елемент, який являється полотном для таких компонентів [17] як: image [11] який буде містити у собі sprite [12], який буде відображати картинки і графіку, rectTransform [14], button [10] та текст [8]. І з цих елементі вже буде будуватись меню та підменю в грі. Також для побудови інтерфейсу буде використовуватись компонент canvas group [13], який буде відповідати за зміну прозорості об’єкта.

Головне меню – це вихідна точка для користувача в грі, його головний навігаційний центр, де він може виконати різноманітні дії. Тут гравець матиме можливість відкривати список доступних рівнів та розпочинати гру, переглядати досягнення, які він вже здобув, відвідати магазин, де можна отримати різноманітні предмети для кастомізації за проходження рівнів або за виконання секретних завдань.

Ще одним важливим елементом головного меню гри буде її назва, а також анімація цього тексту. Назва гри буде відображатись із відповідною анімацією, яка буде відтворювати переливання кольорів, відповідних головному меню. Наприклад, текст буде постійно змінювати свої кольори від чорного до білого

тону, узгоджуючись з загальним стилем головного меню. Це додасть атмосферності та допоможе створити єдиний стиль і дизайн в грі.

Також у головному меню будуть доступні налаштування, де гравець зможе налаштувати звук, та інші параметри гри. Додатково, буде доступне меню з туторіалом, яке допоможе користувачу освоїти основи гри, оскільки гра має специфічний геймплей, який гравець не зможе зрозуміти з першого разу і можливо, це заплутає його або можливо і розлютить, тому в грі буде такого роду меню.

Окремим елементом головного меню будуть пасхалки. Гравець матиме можливість відкривати заховане меню зі списком секретних рівнів, які спочатку будуть приховані. Ці рівні можна буде відкривати по мірі виконання секретних завдань в грі.

До найменш важливих елементів у грі відноситься анімація її інтерфейсу. Усі елементи інтерфейсу будуть анімовані, щоб надати відчуття живості та інтерактивності користувачу. Ці анімації охоплюватимуть широкий спектр від дрібних ефектів, таких як: зміна розміру об'єктів, до більш складних, наприклад, їх переміщення по екрану. Такий підхід дозволить створити захоплююче та вражаюче візуальне враження, яке підтримуватиме зацікавленість гравця протягом всієї гри.

Розташування кнопок у головному меню буде наступним: кнопка "Грати" буде розташована в центрі екрана, щоб привернути увагу користувача та надати йому можливість швидко розпочати гру. Кнопки для відкриття інших меню (наприклад, "Налаштування", "Магазин") будуть розміщені вверху екрана, щоб забезпечити зручний доступ до них. Досягнення будуть розташовані поруч з кнопкою "Грати", щоб підвищити увагу гравця до можливостей отримання досягнень, які можуть додати цікавості грі. Кнопка для відкриття туторіалу може бути розташована поруч з кнопкою магазину в правому верхньому куті, оскільки, це звичайне місце для таких кнопок у багатьох іграх. Крім того, кнопка для доступу до секретних рівнів може бути реалізована, як прихований елемент, який з'являтиметься у вигляді іконки головного героя в лівому

нижньому куті екрана, надаючи гравцеві можливість відкрити цей розділ за бажанням, також буде кнопка “Вийти з гри”, в нижньому правому куті.

Ось, як буде виглядати візуально меню рисунок 2.7.

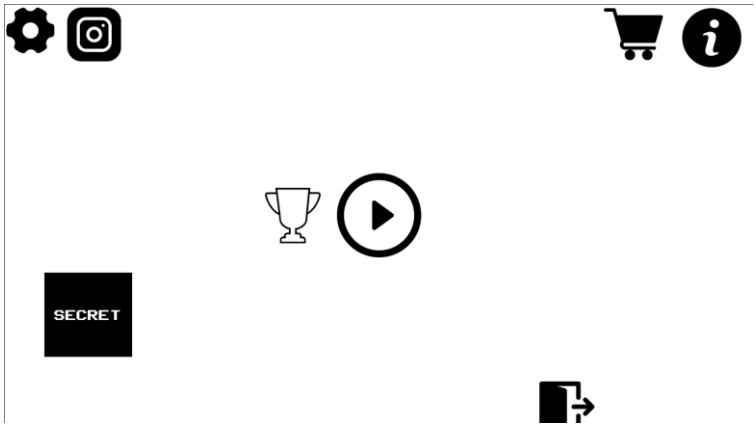


Рисунок 2.7 – Візуальний приклад розстановки кнопок в головному меню

Більшість підменю буде супроводжуватися анімацією, яка буде використовувати метод корутини [20] який дасть змогу виконувати дії асинхронно. Після того, як гравець запустить гру,його відправить на головне меню і поява головного меню буде супроводжуватись такою анімацією – кнопка “Грати” а також кнопка “Досягнення” до запуску гри будуть знаходитись за границями камери під камерою, після запуску гри ці дві кнопки плавно пересунуться по координаті Y до середини камери, в цей же момент текст з назвою гри буде знаходитись за границями камери але уже у верху, і також після переходу у головне меню переміститься по координаті Y до середини камери тільки трішечки вище ніж кнопка грати та кнопка досягнення. Другі кнопки такі як: “Налаштування”, “Instagram”, “Магазин”, “Як грати”, на початку будуть знаходитись на тих ж самих місцях на яких і є, але вони будуть мати нульовий розмір свого об’єкта, після того, як кнопка “Грати” і “Досягнення” з текстом назви гри перемістяться, на свої позиції, то ці кнопки почнуть збільшуватись один за одним до 1, спочатку збільшиться кнопка “Налаштування” потім кнопка “Instagram”, потім кнопка “Магазин” і в самому кінці кнопка “Як грати”.

Тепер перейдемо до підменю в головному меню. Розпочнемо з кнопки "Вийти з гри". Ця кнопка просто відповідає за вихід з гри та повне припинення роботи програми.

Наступним меню є "Як грати?". Це меню досить важливе, зважаючи на специфіку керування грою, зокрема наявність голосового керування. При натисканні на кнопку відкриватиметься вікно з візуальною демонстрацією та текстовим описом кожної функції. Меню буде поділене на сторінки, які можна буде гортати завдяки стрілкам вліво і вправо, на кожній сторінці буде написано про той або інший функціонал в грі і як з ним взаємодіяти, і в цьому меню будуть такі пункти як:

– як грати – тут буде описано процес гри. "Гра проста у своїй суті: все, що вам потрібно – це наявність мікрофона та можливість створювати звуки. Управління персонажем здійснюється за допомогою мікрофона. Ви можете кричати, відригувати, співати, хоч лупцювати ложкою по каструлі – все це звук, а звук у цій грі – це рух" (рисунок 2.8);

– налаштування чутливості мікрофона – тут демонструється процес налаштування чутливості мікрофона. "На цій сторінці ви зможете візуально побачити та текстово описати налаштування чутливості мікрофона. Завдяки повзунку користувач може налаштувати, наскільки сильно потрібно кричати чи дихати в мікрофон, щоб гравець рухався у грі" (рисунок 2.9).

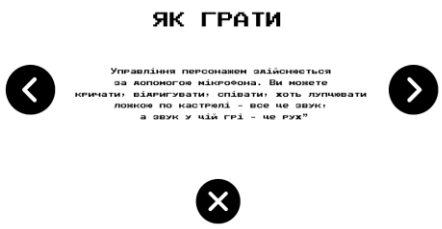


Рисунок 2.8 – Візуальний приклад сторінки “Як грати” в підменю “Як грати”

Рисунок 2.9 – Візуальний приклад сторінки “Налаштування чутливості” в підменю “Як грати”

Далі йде меню налаштувань. Меню, яке знаходиться в лівому верхньому куті, при натисканні на яке, буде відкриватись маленьке і компактне меню налаштувань, де будуть такі параметри налаштування як:

– включити і виключити звуки в грі – це кнопка яка відповідатиме за включення та виключенням звук які будуть в грі;

– налаштування чутливості мікрофона – це буде кнопка, яка буде включати повзунок, який потрібно буде перетягувати вліво або вправо в залежності від того, як користувачу потрібно налаштувати мікрофон. Якщо користувач хоче зробити його більш чутливим, то потрібно перетягнути повзунок у ліву сторону, а якщо захоче зробити його менш чутливим, то просто треба перетягнути повзунок в праву сторону;

– кнопка, яка дає змогу перемкнути керування грою – кнопка, яка дасть змогу вимкнути голосове керування і включити керування завдяки клікам миші. Ця кнопка створена для людей, які захочуть пройти гру без звуку, на всяк випадок, якщо гра гравцю сподобалась, але кричати він не може по різним причинам, до прикладу заважатиме сусідам, то він просто зможе перемкнути на керування мишею і пройти її;

– кнопка закриття меню – кнопка при натисканні, на яку буде закриватись меню налаштувань і буде відкриватись головне меню.

Ось, як буде виглядати візуально меню (рисунок 2.10).

Рисунок 2.10 – Візуальний приклад підменю “Налаштування”

Ще одним елементом меню є кнопка "Instagram". При натисканні на неї гравець буде перенесений на сторінку розробника в Instagram, де буде публікуватися різноманітну інформацію про гру. Це включатиме в себе новини про оновлення, оголошення конкурсів та інші цікаві матеріали, що стосуються гри.

Меню магазин, меню, яке знаходиться в правому верхньому куті, і при натисканні на кнопку магазина буде відкриватися меню з списком шапочок, які гравець може надіти на головного героя, але спочатку гравцеві дається 1 шапочка, другі шапочки він зможе відкрити проходячи рівні. Меню магазина складається з таких елементів:

– головний герой – іконка головного героя, на яку можна надівати шапочки і візуально приглядатись, як ця шапочка буде виглядати на головному героєві;

– індикатор, який демонструє, який рівень потрібно пройти перед тим, як гравець отримає цю шапочку – цей індикатор буде під іконкою гравця, і буде демонструвати номер рівня, який потрібно пройти для відкриття шапочки, Також, крім простих рівнів, там будуть секретні рівні, які будуть відкриватись за секретні досягнення;

– кнопка вибрати шапочку – ця кнопка буде розташована під індикатором цільового рівня, і буде доступна тоді, коли гравець відкриє цю шапочку, після

чого гравець зможе натиснути на неї. Тоді в гравця буде ця шапочка, а також на цій кнопці відмітиться що гравець одів цю шапочку;

– стрілка вліво і стрілка вправо – ці елементи навігації потрібні для того, щоб переключати шапочки з списку наявних шапочок;

– текст кількості шапочок в магазині – це текстовий індикатор, який демонструє сторінки, на якій сторінці знаходиться шапочка і скільки всього є сторінок;

– кнопка закриття меню – кнопка при натисканні на яку, буде закриватись магазин і буде відкриватись головне меню.

Ось, як буде виглядати візуально меню (рисунок 2.11).

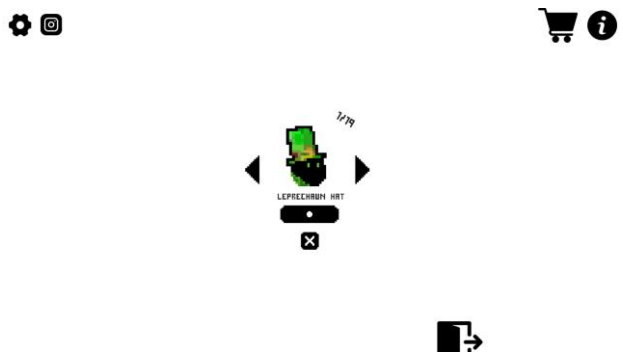


Рисунок 2.11 – Візуальний приклад підменю “Магазину”

Наступним підменю буде список рівнів – це підменю, яке буде відкриватись після того, як гравець натисне на кнопку “Грати”, і це меню буде складатись з кількох елементів.

Кнопка для переходу на рівень – на цій кнопці буде писатись номер рівня, на який буде переносити гравця якщо він натисне на кнопку.

Кнопки для перемикання рівнів – буде дві кнопки, одна буде зліва, а друга з права, і вони будуть давати можливість перемикати рівні між собою.

Під кнопкою переходу на рівень будуть зірочки – це зірковий рейтинг. Рейтинг, який буде демонструвати те, наскільки гравець пройшов той або інший рівень, всього зірок буде 3 штуки.

Текст, який буде показувати що це меню списку рівнів – просто текст над кнопкою для переходу на рівень де буде писатись “Рівні”.

Ось, як буде виглядати візуально меню (рисунок 2.12).

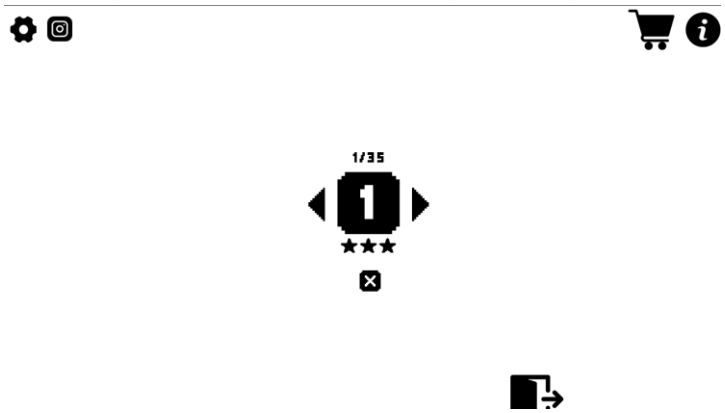


Рисунок 2.12 – Візуальний приклад того, як буде виглядати підменю “Список рівнів”

Наступним підменю, яке буде у головному меню – це меню “Досягнення”. У цьому меню буде знаходитись список всіх секретних досягнень гравця, які він може отримати граючи в гру. Це меню буде ділитись на кілька елементів.

Перший це заголовок де буде знаходитись інформація така як:

– текст з кількістю завдань – це текст, які відображає скільки всього є завдань і скільки з них виконав гравець;

– текст з процентом виконаних завдань – це текст, який буде відображати відсоток суми всіх виконаних завдань.

Друга частина меню – це скролл меню, яке можна прокручувати вниз, у ньому буде знаходитись вся інформація про досягнення, по суті – це список досягнень.

Остання частина – це кнопка виходу з цього меню, яка буде відповідати за закриття меню.

Ось так буде виглядати приблизно підменю, рисунок 2.13.

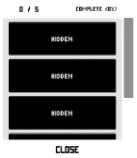


Рисунок 2.13 – Візуальний приклад того, як буде виглядати підменю “Досягнення”

У цьому меню не буде анімацій, воно буде просто відкриватись при натисканні на кнопку досягнення яка буде знаходитись коло кнопки грати.

Наступним меню буде меню виграшу – це меню, яке буде з’являтися після того, як гравець буде добиратись до фінішу, де буде просто плавно з'являться чорний екран і на ньому посередині будуть з'являтись по одному один за одним зірочки, які будуть демонструвати на скільки зірок пройшов рівень користувач. І після того, як зірки з'являться, з'явиться кнопка далі, яка буде переносити гравця на наступний рівень. Ось так буде виглядати це меню, рисунок 2.14.

Рисунок 2.14 – Візуальний приклад того, як буде виглядати меню “Виграшу”

Наступним меню, яке буде доступне у грі, є меню паузи. Це меню призначене для призупинення гри та матиме лише дві кнопки, а також регулятор чутливості мікрофона (рисунок 2.15).

Рисунок 2.15 – Візуальний приклад того, як буде виглядати меню “Паузи”

Ще одним меню, яке буде знаходитись на рівні, це меню програшу, це меню, яке буде з’являтись після того, як гравець попаде на пастку. Це меню буде мати такі кнопки як, перезавантажити рівень, повернутись у головне меню, а також буде мати текст який буде показувати що ти програв (рисунок 2.16).

Рисунок 2.16 – Візуальний приклад того, як буде виглядати меню “Паузи”

2.3 Аналіз та вибір технологій і методів реалізації

Для розробки даної гри був обраний ігровий двигун Unity [39]. Unity – це потужний і популярний багатоплатформовий рушій для розроблення ігор і додатків. Він надає програмістам і дизайнерам можливість створювати інтерактивні проекти для комп'ютерів, мобільних пристроїв, ігрових консолей та інших платформ. Unity відомий своєю гнучкістю, широким набором інструментів і підтримкою різних технологій, роблячи його ідеальним вибором для початківців і досвідчених розробників.

Unity відзначається численними перевагами, які зроблять його привабливим вибором для багатьох розробників. По-перше, його інтерфейс легкий у засвоєнні, що полегшує процес розробки, навіть для тих, хто тільки починає свій шлях у програмуванні. Далі, Unity володіє потужним графічним рушієм, який дозволяє створювати захоплюючі візуальні ефекти та анімацію. Також важливо відзначити, що Unity має широку підтримку плагінів і активну спільноту, готову допомогти своїм досвідом та ресурсами.

Однак у Unity є й деякі недоліки. По-перше, для деяких проектів може знадобитися більше програмування, особливо, коли створюється складна ігрова механіка. Крім того, Unity може бути вимогливим до ресурсів комп'ютера, особливо при роботі з об'ємними проектами, але гра, яка буде розроблятися згідно теми кваліфікаційної роботи проста в програмуванні, а також не вибаглива до ресурсів, тому Unity є ідеальним варіантом для створення гри.

Наступний інструмент, який буде використовуватись для створення гри – це IDE Visual Studio Code [40]. Visual Studio Code – це безкоштовний, легкий, але потужний редактор вихідного коду, який працює на робочому столі та в Інтернеті і доступний для Windows, macOS, Linux та Raspberry Pi OS. Він має вбудовану підтримку JavaScript, TypeScript та Node.js, а також багату екосистему розширень для інших мов програмування (таких як: C++, C#, Java, Python, PHP та Go), середовищ виконання (таких як: .NET та Unity), середовищ (таких як: Docker та Kubernetes) та хмар (таких як: Amazon Web Services, Microsoft Azure та Google Cloud Platform).

Visual Studio Code є незамінним інструментом для розробки ігор в Unity, особливо при використанні мови програмування C#. Завдяки вбудованій підтримці C# і багатій екосистемі розширень, розробники можуть легко створювати, редагувати та налагоджувати код гри прямо у Visual Studio Code.

Одна з ключових переваг полягає у синтаксичному виділенні та автодоповненні коду, що робить процес написання коду більш зручним і зменшує кількість помилок. Також важливо відзначити можливість відлагодження коду безпосередньо з редактора, що дозволяє швидко виявляти та виправляти помилки.

Крім того, Visual Studio Code дозволяє налаштовувати редактор за потребами конкретного проекту завдяки великій кількості розширень для різних мов програмування та інструментів. Інтеграція з Unity – ще одна вагома перевага, яка дозволяє розробникам працювати над кодом гри безпосередньо з редактора, не виходячи з інтерфейсу розробки Unity.

У цілому, Visual Studio Code у поєднанні з Unity створює потужний та зручний інструментарій для створення високоякісних ігор на мові програмування C#.

Aseprite [33] – це програмне забезпечення для створення та редагування піксельної графіки та анімації. Воно спеціалізується на створенні 2D-графіки, зокрема для ігор, анімаційних фільмів та веб-коміксів. Aseprite надає широкий спектр інструментів для малювання, редагування та анімації піксельних зображень, включаючи різноманітні пензлі, засоби заповнення, функції кадрування та анімаційні інструменти. Ця програма буде використовуватися для створення 2D pixel art [35] графіки в гри, до прикладу для того щоб намалювати головного героя або для того, щоб намалювати капелюшки.

Photoshop CC 2018 [34] – це досить всебічний графічний редактор, який дозволяє створювати та редагувати зображення, працювати з графічним дизайном та ретушувати фотографії. Він розроблений компанією Adobe і відомий своєю потужністю та багатофункціональністю. Photoshop надає широкий спектр інструментів для маніпулювання зображеннями, включаючи

роботу з растровою та векторною графікою. З його допомогою можна створювати складні композиції, обробляти фотографії, наносити різноманітні ефекти та фільтри, а також працювати з текстом та іншими графічними об'єктами. В грі він буде використовуватись для того, щоб редагувати деякі елементи UI, або іконок.

Для створення цієї гри також планується використання кількох плагінів або ассетів в Unity, які спростять виконання ряду завдань у процесі розробки гри. Використання цих ресурсів дозволить ефективніше впоратися з різними аспектами створення гри та зменшить час, необхідний для розробки.

Ці плагіни та ассети можуть включати в себе різноманітні ресурси, такі як: готові моделі персонажів або оточення, набори звуків, текстури, анімації, системи штучного інтелекту, готові розв'язки для реалізації певних ігрових механік та інше. Вони можуть також надавати зручний інтерфейс для налаштування і редагування цих ресурсів прямо в середовищі Unity.

Для пошуку ресурсів для гри я використовував такі сайти як Asset Store та Itch.io.

Asset Store [26] – це центральний ресурс, де збирається розмаїття безкоштовних та платних матеріалів, які створені як самою компанією Unity Technologies, так і членами глобальної спільноти розробників.

Тут можна знайти широкий спектр ресурсів: від текстур, моделей та анімації до повноцінних проектів, навчальних посібників та розширень. Цей різноманітний контент створений з метою полегшення процесу розробки, допомагаючи вам вдосконалити ваші проекти, гри або додатки.

Використання ресурсів з магазину дозволяє значно зекономити час та зусилля, які витрачаються на створення аналогічних елементів "з нуля". Замість цього, ви можете швидко і легко додати необхідні ресурси до вашого проекту, скорочуючи час розробки і поліпшуючи якість готового продукту.

Другий сервіс це Itch.io [27] – це відкрита платформа для незалежних творців цифрового контенту, особливий акцент якої зроблений на незалежні відеоігри. Тут будь-хто може реалізувати свої творчі ідеї, продавати створений

контент та керувати своїм профілем. На цій платформі є як ігри, так і ресурси для цих ігор, таких як: безкоштовна графіка та код, які мають на собі ліцензію MIT або CC0, що дає змогу використовувати контент у любому виді, як у комерційному так і у іншому виді.

Ліцензія MIT [28] – це проста та дозвільна ліцензія на програмне забезпечення, яка надає користувачам широкі права щодо використання і розповсюдження програми та її документації. Вона дозволяє користувачам використовувати, копіювати, модифікувати, об'єднувати, публікувати, поширювати, субліцензувати та продавати програмне забезпечення за умови, що ліцензія та повідомлення про авторські права включені в усі копії або значні частини програми.

Ліцензія CC0 [29] – це найбільш вільна форма ліцензії, яка надає можливість завантажувати, використовувати, змінювати, демонструвати або будь-яким іншим чином використовувати будь-який твір, що має цю ліцензію.

Ця ліцензія, також відома як Creative Commons Zero (CC0), фактично перетворює контент у суспільне надбання, аналогічно контенту без авторського права. Вона дозволяє авторам поширювати свої роботи без будь-яких обмежень, що створює сприятливі умови для вільного використання творчого контенту.

Тому було обрано ці магазини, оскільки при отриманні платного або безплатного контенту там для своєї гри розробник буде знати те, що в нього не буде проблем з авторським правом.

В грі будуть використовуватись кілька інструментів з Asset Store а саме, LeanTwean та Achievement System.

LeanTwean [31] – це легка та потужна бібліотека анімації для Unity, яка дозволяє розробникам створювати складні анімаційні ефекти зручним та ефективним способом. Вона надає широкий набір функцій для керування рухом, зміною розміру, прозорістю та іншими властивостями гри. В грі LeanTwean буде використовуватися для створення анімацій для інтерфейсу.

Achievement System [30] – це інструмент, який дозволяє розробникам легко створювати та керувати ігровими досягненнями. Цей ассет буде

використовуватися для створення логіки та елементів інтерфейсу які відповідають за досягнення.

Cinemachine [36] – інструмент для динамічних розумних безкодових камер, які дозволяють створювати найкращі кадри на основі композиції сцени та взаємодії, дозволяючи вам налаштовувати, ітерувати, експериментувати та створювати поведінку камери в реальному часі. Цей інструмент був використаний для створення камери для гравця.

TextMeshPro [37] – це найкраще текстове рішення для Unity. Цей інструмент був обраний для використання текст в інтерфейсі.

Pixel Art Forest [32] – це візуальний ассет (графіка), яка буде використовуватися для створення заднього фону у грі, він має кілька слоїв тому завдяки цьому можна створити паралакс ефект.

2.4 Висновки проєктування програмного забезпечення

На початку розділу було детально розглянуто архітектуру програмного забезпечення для розроблюваної гри. Визначено основні компоненти та класи, які відповідають за різні аспекти функціоналу гри, такі як керування персонажем, обробка голосових команд, головне меню, магазин, рівні, анімації тощо.

Для кожного класу було описано його основні функції та взаємозв'язки з іншими компонентами системи. Представлено блок-схеми [5], що ілюструють логіку роботи цих класів [7].

Архітектура була спроектована з урахуванням принципів модульності, розширюваності та гнучкості. Кожен клас відповідає за певну функціональність, що полегшує подальшу підтримку та розвиток програмного забезпечення.

Детально описано взаємодію між компонентами, такими як обробка голосових команд, керування персонажем, відображення графічного інтерфейсу та анімацій.

Загалом, ця архітектура забезпечує чітку структуру програмного забезпечення, розподіл відповідальності між компонентами та можливість для подальшого розширення функціоналу гри.

У другому пункті було виконане проєктування інтерфейсу користувача, яке охоплювало різноманітні меню, що забезпечують зручний доступ до всіх функцій програми чи гри. Серед цих меню відзначаються головне меню, яке є основним хабом для навігації, підменю налаштувань для керування параметрами гри, меню магазину для отримання додаткового контенту, підменю "як грати?" з інструкціями для новачків, а також меню списку рівнів та секретних рівнів для вибору рівня гри. До інших важливих елементів інтерфейсу входять меню досягнень, кнопка Instagram для спілкування з гравцями, а також меню програшу, виграшу та паузи, які дозволяють користувачам керувати грою та взаємодіяти з нею в різних ситуаціях.

Для кожного елементу інтерфейсу, такого як меню налаштувань, було детально описано взаємодію користувача з цим елементом. Це робить інтерфейс більш привабливим та зрозумілим для користувача. Поліпшення візуального враження від взаємодії з програмою чи грою є важливим елементом задоволення користувача від продукту.

У третьому пункті були описані інструменти, які будуть використовуватися для створення гри. Для розробки гри буде використано ігровий рушій Unity через його потужність, гнучкість, зручний інтерфейс та підтримку різних платформ. У якості інтегрованого середовища розробки (IDE) для написання коду на C# буде використовуватися Visual Studio Code через його зручність, підтримку C# та інтеграцію з Unity.

Для створення піксельної графіки та анімації буде використовуватися програма Aseprite, а для редагування зображень та створення деяких елементів інтерфейсу – Photoshop CC 2018. Для пришвидшення розробки планується

використання безкоштовних та платних ресурсів і плагінів з Unity Asset Store та Itch.io, які мають відкриті ліцензії MIT або CC0. Це дозволить уникнути проблем з авторськими правами.

З Asset Store будуть використані плагіни LeanTween для створення анімацій інтерфейсу та Achievement System для реалізації системи досягнень. Також буде використаний графічний ассет Pixel Art Forest для створення фону з паралакс-ефектом. Підібрані технології та інструменти забезпечать зручний процес розробки гри з необхідним функціоналом та естетичними якостями.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ

3.1 Програмна реалізація модулів

Першим модулем буде керування гравцем, і він буде містити в собі кілька методів [18] які будуть відповідати за функціонал.

Awake() [22] – цей метод викликається в момент створення об'єкта. Він перевіряє, чи існують пристрої для запису звуку та ініціює вхід мікрофона, якщо такі пристрої доступні.

Start() – цей метод викликається один раз при запуску програми. Він ініціалізує різні компоненти гравця, такі як Rigidbody2D, Animator та інші. Також він встановлює початкові значення для деяких змінних і починає анімацію появи гравця.

StartSpawn() – цей метод викликається для початку гри або після смерті гравця. Він встановлює певні параметри для початку гри, зупиняє анімацію смерті, створює ефект появи гравця та скидає лічильник смертей.

FinishSpawnAnimation() – цей метод викликається в кінці анімації появи гравця. Він переводить анімацію у стан бездіяльності.

FixedUpdate() [24] – цей метод викликається кожен кадр. Він перевіряє стан гравця (чи він мертвий) та викликає методи для керування гравцем (за допомогою миші або голосу), якщо гра не призупинена.

OnTriggerEnter2D(Collider2D collision) [19] – цей метод викликається, коли об'єкт гравця зіштовхується з іншим об'єктом. Він перевіряє теги об'єктів, з якими гравець зіштовхується, і виконує відповідні дії (наприклад, визначає, чи гравець потрапив у пастку або влучив у кулю).

FlapPlayer() – цей метод викликається для здійснення стрибка гравця. Він встановлює швидкість та напрямок руху гравця, додає силу вгору, відтворює звук стрибка та створює ефект стрибка.

FlapPlayerWithVoice() – цей метод викликається для стрибка гравця за допомогою голосу. Він аналізує аудіосигнал з мікрофону та викликає метод стрибка, якщо рівень звуку перевищує задану чутливість.

LoadVoiceSensitivity() – цей метод завантажує рівень чутливості голосу з налаштувань.

CloseCollider() – цей метод викликається після смерті гравця. Він вимикає колайдер гравця, щоб уникнути подальшої взаємодії з об'єктами.

FlipAnimation() – цей метод встановлює анімацію обертання гравця відповідно до напрямку руху.

Нижче наведено код який відповідає за взаємодію гравця з мікрофоном, де після того як звук був захоплено, його обробляють, і провіряють чи це звук з мікрофона і якщо так то запускають метод FlapPlayer(), який відповідає за підстрибування.

```
void FlapPlayerWithVoice()
   {
      //get mic volume
      int dec = 128;
      float[] waveData = new float[dec];
      int micPosition = Microphone.GetPosition(null) - (dec + 1); // null means the first microphone
      microphoneInput.GetData(waveData, micPosition);

      // Getting a peak on the last 128 samples
      float levelMax = 0;

      for (int i = 0; i < dec; i++)
      {
         float wavePeak = waveData[i] * waveData[i];
         if (levelMax < wavePeak) levelMax = wavePeak;
```

```
        }

        float level = Mathf.Sqrt(Mathf.Sqrt(levelMax));

        if (level > SettingsController.voiceSensitivity && !flapped)
        {
            if (GameController.isPaused == false)
            {
                FlapPlayer();
            }
            flapped = true;
        }

        if (level < SettingsController.voiceSensitivity && flapped)
        {
            flapped = false;
        }
    }
```

Наступним модулем буде менеджер, що відповідає за функціонування різноманітних меню на рівні гри, зокрема за меню паузи та екрану програшу. Цей модуль керує їх функціоналом та анімацією.

Start() – цей метод викликається при старті гри. Він ініціалізує початкові значення деяких змінних, налаштовує анімації та встановлює значення часу до включення екранного заставного зображення.

StartTransition() – цей метод запускає анімацію переходу між сценами. Він поступово знижує прозорість ефекту переходу до нуля та вимикає його.

Died() – цей метод викликається, коли гравець помирає. Він встановлює прапорець gameOver в true, викликає метод очікування поразки.

GoToMenu(string NameScene) – цей метод переводить гравця до головного меню гри.

ChangeToScene(string sceneToChangeTo – цей метод змінює сцену гри на задану.

RestartGame() – цей метод перезапускає гру.

PauseGame() – цей метод призупиняє гру та відображає меню паузи.

ResumeGame() – цей метод відновлює гру після паузи.

Наступний модуль це модуль який відповідає за основні налаштування гравця.

ChangeToScene(string sceneToChangeTo) – цей метод відповідає за зміну сцени гри на ту, яка передається в параметрі.

Update() [23] – цей метод викликається кожен кадр. Він викликає метод LoadPrefs() для завантаження збережених налаштувань.

AdjustVoiceSensitivity(float newSensitivity) – цей метод відповідає за зміну чутливості голосу. Він отримує нове значення чутливості та зберігає його в PlayerPrefs.

LoadPrefs() – цей метод завантажує збережені налаштування гри. Він завантажує значення чутливості голосу з PlayerPrefs і встановлює його на слайдері.

Нижче показано як працює функція збереження і виставлення налаштувань чутливості мікрфона.

```
private void Update()
{
    LoadPrefs();
}

public void AdjustVoiceSensitivity(float newSensitivity)
{
    voiceSensitivity = newSensitivity;
```

```
        PlayerPrefs.SetFloat("sensitivity", newSensitivity);
    }

    private void LoadPrefs()
    {
        //loading Voice Sensitivity
        voiceSensitivity = PlayerPrefs.GetFloat("sensitivity", 0.3f);
        slider.value = voiceSensitivity;
    }
```

Наступний модуль буде відповідати за систему магазину, в цей модуль буде входити комплекс класів та методів.

Першим класом буде клас ShopMenu, він відповідає за управління магазином у грі. Він містить логіку для відображення та вибору різних капелюхів для гравця, а також управління відкриттям та закриттям магазину. Крім того, він забезпечує анімаційні ефекти під час переходів між кепками та сторінками магазину.

Start() – цей метод ініціалізує початкові значення деяких змінних та перевіряє відкриті меню рівнів чи таємні меню перед відкриттям магазину.

CloseShopMenu() – цей метод закриває меню магазину. Він викликається при натисканні кнопки закриття магазину.

HatsMenuClosed() – цей метод приховує всі об'єкти, пов'язані з капелюхами, крім першого капелюха.

StartShopMenu(int Hat, int Skin) – цей метод відповідає за відкриття меню магазину. Він відтворює анімаційні ефекти відкриття меню, а також відображає вибрану кепку та гравця.

CloseShopMenu(int Hat, int Skin) – цей метод відповідає за закриття меню магазину. Він відтворює анімаційні ефекти закриття меню та приховує всі елементи, пов'язані з кепками.

RightButtonClick() та LeftButtonClick() – ці методи відповідають за переміщення між сторінками кепок у магазині. Вони відтворюють анімаційні ефекти та відображають відповідні кепки.

Нижче показано шматочок кода який відповідає за запуск меню магазину, який запускається завдяки корутині, і вмикає послідовність з анімацій.

```
IEnumerator StartShopMenu(int Hat, int Skin) // Open shop menu
    {
        StartSound("Transition");
        LeanTween.moveX(Left.gameObject.GetComponent<RectTransform>(), -311f, 0.2f);
        yield return new WaitForSeconds(0.4f);
        StartSound("Transition");
        LeanTween.moveX(Right.gameObject.GetComponent<RectTransform>(), 311f, 0.2f);
        yield return new WaitForSeconds(0.4f);
        StartSound("Transition");
        StartCoroutine(AnimButton(PlayerSkin[Skin], 0.2f, 2.5f, 1.8865f));
        yield return new WaitForSeconds(0.5f);
        StartSound("Transition");

LeanTween.moveY(HatSetting[Hat].HatSkin[0].gameObject.GetComponent<RectTransform>(), HatSetting[Hat].HatY[0], 0.2f);
        StartCoroutine(SpawnHatObjects(true, HatSetting[Hat].HatName[0], 0.9f, 0.8f));
        StartCoroutine(AnimButton(HatButton[Hat], 0.2f, 5f, 4.7522155f));
        yield return new WaitForSeconds(0.4f);
        StartSound("Transition");
```

```
LeanTween.moveY(CloseButton.gameObject.GetComponent<RectTransform>(), -283f, 0.3f);
        LeanTween.scale(HatCounterobject, new Vector3(0.57586f, 0.57586f, 0.57586f), 0.2f);
    }
```

Наступним класом буде NewHatOpened, він відповідає за відображення повідомлення про відкриття нового капелюха в грі.

Start() – цей метод викликається при старті об'єкта. Він завантажує збережене значення про те, чи вже було відкрито новий капелюх, а також перевіряє, чи вже пройдено необхідну кількість рівнів для відкриття цього капелюха. Якщо капелюх ще не відкрито і не відповідно пройдено необхідну кількість рівнів, він запускає анімаційне повідомлення.

StartMessage() – цей метод відповідає за анімаційне відображення повідомлення про відкриття нового капелюха. Він відтворює анімаційні ефекти переміщення повідомлення, показує його протягом деякого часу, а потім приховує його. Після цього він зберігає інформацію про те, що повідомлення про відкриття капелюха було відображено, та знищує сам об'єкт.

Наступний клас який входить сюди це ц ChoseHat, він відповідає за вибір та управління капелюхами для персонажа в грі. Він дозволяє користувачеві вибрати конкретний капелюх з доступних варіантів у магазині.

Start() – цей метод викликається при старті об'єкта. Він ініціалізує змінну SkinLength на основі кількості доступних шкірок. Також він завантажує збережені дані про вибрані користувачем шкірки зі збереження та встановлює відповідні значення для змінних IsSelect та Skin.

SelectThatSkin(int SkinNumber) – цей метод викликається при виборі користувачем певної шкіри. Він змінює значення IsSelect на номер вибраної шкіри та зберігає цю інформацію в збереженні.

Останній клас, який буде входити в цей модуль – це клас HatOpen_LEVEL. Він відповідає за відкриття кнопки, яка буде дозволяти одягати капелюх, при досягнені потрібного рівня.

Метод DefoultLevel() перевіряє, чи користувач вже пройшов поточний рівень, заданий параметром HowLevel. Якщо користувач пройшов рівень, то кнопка для відкриття рівня (LevelOpenButton) приховується (деактивується), інакше кнопка залишається видимою.

Метод SecrettLevel() виконує аналогічну функцію, але для секретного рівня. Він перевіряє, чи користувач вже завершив секретний рівень, заданий параметром HowSecretLevel. Якщо користувач вже завершив секретний рівень, то кнопка для відкриття рівня приховується, в іншому разі кнопка залишається видимою.

Наступний модуль – це модуль, який відповідатиме за логіку секретних рівнів а також за логіку досягнень.

Перший клас, який буде у цьому модулі – це клас SecretManager, який відповідає за відкриття рівнів.

Метод Start встановлює початкові значення для змінних SecretStart і SecretNumber. Він також завантажує дані збереження для кнопок заблокованих секретних рівнів зі збереження гравця.

Метод Update перевіряє стан кожного заблокованого рівня і встановлює відповідний стан кнопки. Якщо рівень є заблокованим, кнопка вимикається і відображається об'єкт блокування. Якщо рівень розблокований, кнопка стає активною, а об'єкт блокування приховується.

Метод OpenSecterLevel викликається для розблокування секретного рівня з вказаним номером. Він зберігає інформацію про розблокування рівня в збереженні гравця.

Наступним класом є один із класів, який відповідає за секрет, а саме Secret_3.

Метод Start() викликається при початку роботи об'єкта. Він завантажує збережені дані про кількість смертей (DeathHowe і HoweDeathPlayer) і

перевіряє, чи було вже досягнуто визначеної кількості смертей. Якщо гравець досягнув 100 смертей, викликається метод Unlock для відкриття досягнення "secret3", розблоковується секретний рівень 3, оновлюється збереження про зміну статусу смертей і встановлюються значення для SecretNumber і SecretStart в класі SecretManager.

Метод AddDeath() дозволяє додавати одну смерть до лічильника HoweDeathPlayer. Якщо кількість смертей не перевищує 100, значення зберігається в збереженні гравця.

Наступний модуль – це модуль, який відповідає за управління головним меню, модуль MainMenu.

Метод Start() викликається при початку роботи об'єкта. Він ініціалізує різні поля, встановлює початкові значення, реєструє звуки, налаштовує різні елементи і розпочинає деякі анімації.

Метод OpenSettingMenu() відкриває вікно налаштувань.

Метод CloseSettingMenu() закриває вікно налаштувань.

Метод OpenMicrophoneMenu() відкриває меню мікрофону.

Метод CloseMicrophoneMenu() закриває меню мікрофону.

Метод GoToLevelMenu() перехід до меню рівнів.

Метод RightButtonClick() і LeftButtonClick() обробляють кліки на кнопки "праворуч" і "ліворуч" для прокрутки списку рівнів.

Метод CloseLevelMenuClick() закриває меню рівнів.

Метод OpenLevel() відкриває вказаний рівень.

Метод OpenInstagram() відкриває Instagram за вказаним URL.

Метод WhatIsMenu() відкриває або закриває меню "Що це".

Метод WhatIsRightButtonClick() і WhatIsLeftButtonClick() обробляють кліки на кнопки "праворуч" і "ліворуч" для прокрутки інформаційних карток "Як це працює".

Метод MainPlayer() відтворює анімацію головного героя.

Метод OpenShop() і CloseShop() відкриває або закриває меню магазину.

Метод ExitGame() завершує гру.

Метод OpenPlayerLevelData() і ClosePlayerLevelData() відкриває або закриває меню списку рівнів гравця.

Наступний модуль відповідає за систему відкриття рівнів.

Першим класом, який буде відповідати за це, буде клас LevelDone, який відповідає за обробку завершення рівня в грі. Він містить різні методи та поля для управління ефектами завершення рівня, відкриття наступного рівня та іншими функціями.

OnTriggerEnter2D(Collider2D collision) – цей метод викликається, коли гравець зіткнеться з об'єктом, що має тег "Player". Він відкриває елементи інтерфейсу завершення рівня, блокує рух гравця, встановлює прапорець завершення рівня та викликає методи для визначення кількості зірок та розблокування наступного рівня.

LockStart() – заблоковує рух гравця та відображає панель завершення рівня.

OneStar(), TwoStar(), ThreeStar() – анімує появу відповідно однієї, двох або трьох зірок в залежності від кількості набраних гравцем очок.

UnLockLevel() – розблоковує наступний рівень, якщо поточний рівень більше або дорівнює останньому відкритому рівню.

NextLevel(string SceneName) – відкриває наступний рівень.

NextLevelStart(string SceneName) – починає перехід на наступний рівень.

Наступний клас, який входить до цього модуля – це LevelUnLockSystem, він відповідає за розблокування рівнів у грі. Він має на меті забезпечити відповідність між рівнями, які гравець може грати, та відповідними кнопками у головному меню гри.

levelUnLock – поле, яке зберігає номер останнього розблокованого рівня.

Buttons – це масив кнопок, які ведуть до наступних рівнів.

LockObject – це масив об'єктів, які представляють заблоковані рівні (зазвичай це ілюстрації замків або інших перешкод).

buttons[i].interactable = false – ця команда робить кнопку buttons[i] недоступною для взаємодії, тобто гравець не може натискати її.

LockObject[i].SetActive(true) – ця команда активує об'єкт LockObject[i], що представляє заблокований рівень.

buttons[i].interactable = true – ця команда робить кнопку buttons[i] доступною для взаємодії, тобто гравець може натискати її.

Наступний модуль відповідає за систему зірок – це система, яка буде демонструвати гравцеві на скільки зірок він пройшов рівень.

Цей модуль також ділиться на кілька класів і першим класом буду клас StarOnLevelUI. Він відповідає за відображення кількості зірок, які отримав гравець на конкретному рівні у грі.

StarSwitch() – метод відповідає за відображення зірок на екрані в залежності від кількості StarCount. Він активує або деактивує відповідну кількість об'єктів Star, щоб показати, скільки зірок отримав гравець на рівні.

StarCountSave(int Count) – цей приватний метод викликається для збереження кількості зірок, отриманих на рівні, у локальний файл даних користувача за допомогою PlayerPrefs. Він перевіряє, чи кількість отриманих зірок більше, ніж та, яка вже зберігається, і, якщо так, зберігає нову кількість. Крім того, він перевіряє, чи рівень завершено (LevelDone.LevelFinish == 1), перш ніж зберігати кількість зірок.

StarCount – статичне поле, яке зберігає кількість зірок, отриманих на рівні. Використовується для зберігання і відображення кількості зірок.

LevelNumber – рядок, що зберігає ім'я рівня. Використовується для ідентифікації конкретного рівня в системі збереження PlayerPrefs.

Star – масив об'єктів, які представляють зірки на екрані. Використовується для відображення кількості отриманих зірок.

Наступним класом, який входить до цього модуля – є клас, який відповідає за відображення кількості зірок, отриманих на певному рівні, у головному меню гри, а саме клас StarOnLevelMenu.

StarManager() – приватний метод відповідає за управління відображенням зірок на екрані, в залежності від кількості отриманих зірок. Він перевіряє значення, збережене у PlayerPrefs для даного рівня, і відповідно до цього

значення активує або деактивує відповідну кількість об'єктів Star, щоб показати, скільки зірок було отримано на рівні.

StarCount – статичне поле, яке зберігає кількість зірок, отриманих на даному рівні. Використовується для зберігання та відображення кількості зірок на головному меню гри.

LevelNumber – рядок, який зберігає ім'я рівня. Використовується для ідентифікації конкретного рівня в системі збереження PlayerPrefs.

Star – масив об'єктів, які представляють зірки на екрані головного меню. Використовується для відображення кількості отриманих зірок.

Нижче продемонстровано те як працюють методи StarSwitch та StarCountSave.

```
private void StarSwitch()
    {
        if(StarCount == 0)
        {
            Star[0].SetActive(false);
            Star[1].SetActive(false);
            Star[2].SetActive(false);
            StarCountSave(0);
        }
        else if (StarCount == 1)
        {
            Star[0].SetActive(true);
            Star[1].SetActive(false);
            Star[2].SetActive(false);
            StarCountSave(1);
        }
        else if (StarCount == 2)
        {
```

```
            Star[0].SetActive(true);
            Star[1].SetActive(true);
            Star[2].SetActive(false);
            StarCountSave(2);
        }
        else if(StarCount == 3)
        {
            Star[0].SetActive(true);
            Star[1].SetActive(true);
            Star[2].SetActive(true);
            StarCountSave(3);
        }
    }
    // Save how many stars depending on how much you ate
    private void StarCountSave(int Count)
    {
        if (LevelDone.LevelFinish == 1)
        {
            if(Count > PlayerPrefs.GetInt("StarCount_Level" + LevelNumber))
            {
                StarCount = Count;
                PlayerPrefs.SetInt("StarCount_Level" + LevelNumber, StarCount);
            }
        }
    }
```

Останній клас, а саме клас, який відповідає за зірку, яка буде знаходитись на карті Star.

OnTriggerEnter2D(Collider2D collision) – цей метод викликається, коли об'єкт, до якого прикріплений скрипт, зіштовхується з іншим об'єктом, який має

коллайдер та тригерну зону. У цьому випадку, якщо об'єкт, з яким зіштовхнулася зірка, має тег "Player", зірка видаляється з сцени, збільшується лічильник зірок у класі StarOnLevelUI.

Наступний і останній модуль – це модуль, який відповідає за пастки, і всі пастки за логікою різні і їх приблизно 14 штук, тому буде продемонстрований приклад 1 з пасток, а саме Trap_2, яка відповідає за рух об'єкта між двома позиціями вгору та вниз.

Update() – цей метод викликається кожен кадр. У ньому відбувається обробка логіки руху об'єкта. Спочатку розраховується величина кроку руху залежно від швидкості об'єкта та часу, який пройшов з попереднього кадру. Потім перевіряється, чи досягнута об'єктом цільова позиція (target.position). Якщо так, змінна moveUp встановлюється в false, щоб змінити напрямок руху. Якщо об'єкт повернувся до своєї початкової позиції (startPos), moveUp встановлюється в true, щоб рухатися вгору. Далі залежно від значення moveUp, об'єкт рухається або до своєї початкової позиції, або до цільової позиції, використовуючи метод Vector3.MoveTowards().

3.2 Керівництво користувача

Для того щоб розпочати гру, все що потрібно гравцеві – це спочатку ознайомитись з грою у головному меню, натиснувши на кнопку у правому верхньому куті, яка називається “Як грати”. Там буде демонструватись і описуватись те, як працює керування гравця, а саме те, що для того, щоб грати потрібен мікрофон і вміння створювати різноманітні типи звуків. Також продемонстровано те, як регулювати чутливість мікрофона у грі (рисунок 3.1 та 3.2)

Рисунок 3.1 – Демонстрація того, як виглядає пояснення того, як грати у грі на сторінці №1

Рисунок 3.2 – Демонстрація того, як виглядає пояснення того, як грати у грі на сторінці №2

Наступним кроком для гравця є перевірка того, що є у підменю магазину, яке він може знайти у головному меню у правому верхньому куті, де є іконка шапочки. Натиснувши на неї, він зможе відкрити магазин з капелюхами, капелюшки можна отримати за проходження певної кількості рівнів. Рисунок 3.3

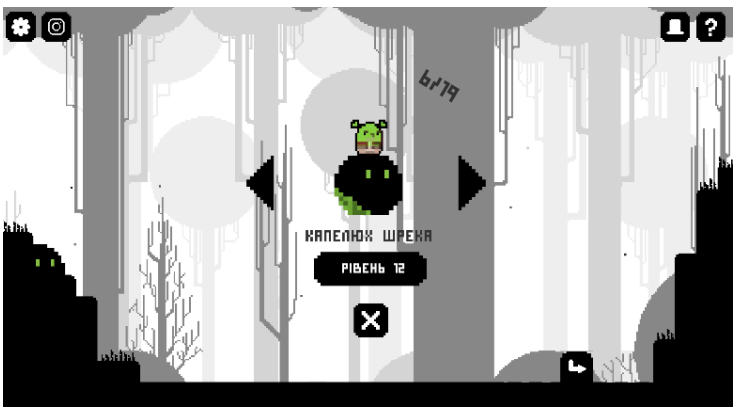


Рисунок 3.3 – Демонстрація того, як виглядає підменю “Магазин”

Також в головному меню є підменю з списком секретних досягнень, але те, як можна заробити ці досягнення секрет, думаю це логічно, на те вони й секретні (рисунок 3.4).

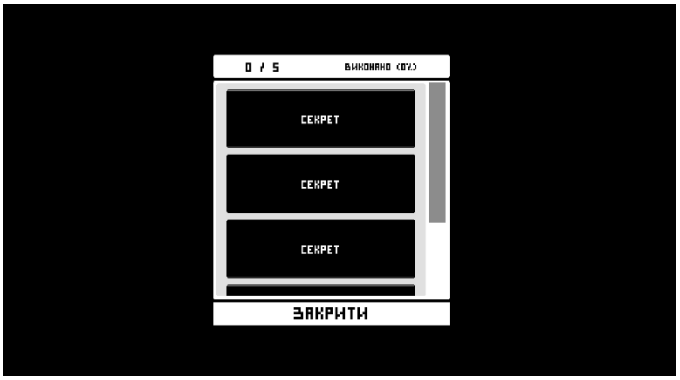


Рисунок 3.4 – Демонстрація того, як виглядає підменю “Досягнення”

Після того, як користувач ознайомився зі всіма підменю, які є у головному меню, можна перейти до того, як почати гру і як грати. Для того, щоб розпочати грати, треба просто натиснути кнопку “Грати” і вибрати 1 рівень, оскільки він по суті єдиний який доступний.

Після чого, можна починати кричати кричати, і добиратись до порталу. На шляху до порталу треба збирати зірочки, але це необов'язково, користувач просто може дістатися до порталу і перейти на наступний рівень (рисунок 3.5).

Рисунок 3.5 – Демонстрація того, як виглядає гемлей

На цьому все. Далі все, що потрібно – це просто витримка, багато сил кричати і багато часу щоб грати, оскільки у грі 35 різних рівнів.

3.3 Вимоги до технічних та програмних засобів

Для коректної роботи гри необхідно мати ряд технічних характеристик, які описані нижче.

Операційна система – гра підтримує операційні системи Windows 7, 8, або 10 (64-бітні).

Процесор – мінімум двоядерний процесор з тактовою частотою 2.0 ГГц або аналогічний.

Оперативна пам'ять – рекомендовано мати не менше 4 ГБ оперативної пам'яті.

Графічний адаптер – мінімум Intel HD Graphics 4000 або аналогічний.

Місце на диску – мінімум 100 мб вільного місця на жорсткому диску.

3.4 Тестування додатка

Під час розробки додатку було обрано низку методів, спрямованих на забезпечення якості та надійності програмного продукту, а саме на тестування ПЗ [38].

Функціональне тестування – даний метод використовувався для перевірки відповідності функціональних вимог програмного продукту. Кожна функція була протестована окремо, переконуючись, що вона працює належним чином та відповідає специфікаціям.

Ігрове тестування – організація публічних бета-тестувань, де гравці, що відповідають цільовій аудиторії, мають можливість випробувати гру перед її повним випуском. Це дозволяє отримати поверхневий огляд гри в реальних умовах від різних гравців і виявити потенційні проблеми, які можуть бути пропущені внутрішнім тестуванням.

Збір зворотного зв'язку – систематичне збирання відгуків та пропозицій від гравців під час тестування. Це може бути здійснено через спеціальні форми зворотного зв'язку або форуми спільноти гравців. У випадку розробленого програмного застосунку, це було зроблено завдяки Instagram, де проводилося опитування про гру, і про те, що можна були би в неї додати.

Після проходження усіх вищезгаданих етапів тестування гра вже демонструє значне покращення у якості та функціональності.

3.5 Висновки програмної реалізації та тестування

У першому розділі була проведена робота по опису наявних модулів у грі, які відповідають за різний функціонал у грі, в основному були виділенні такі модулі як: модуль керування гравцем, модуль головного меню, модуль меню на рівнях, модуль зіркового рейтингу, модуль відкриття рівнів, модуль налаштувань, модуль пасток.

Наступним кроком було описати технічні характеристики для гри, тут потрібно було описати при яких технічних характеристиках буде комфортно грати у гру. Були виділені такі характеристики як: процесор, місце на диску, операційна система, графічний адаптер, кількість оперативної пам’яті.

Останнім кроком відбулося проведення тестування гри, для перевірки і пошуку можливих багів. Були застосовані такі типи тестувань як: функціональне тестування, ігрове тестування, збір зворотного зв'язку.

ВИСНОВКИ

В даній кваліфікаційній роботі було виконано вичерпний аналіз та розробку різноманітних аспектів проекту гри з використанням голосового керування в жанрі "Платформер". Починаючи з першого розділу, виконано глибокий огляд жанру та його особливостей, спрямований на виявлення переваг для реалізації гри з голосовим керуванням. Встановлено, що динаміка та різноманітність "Платформера" створюють ідеальні умови для інтеграції голосової взаємодії.

У другому розділі проведено дослідження ринку існуючих ігор з голосовим керуванням, зокрема в жанрі "Платформер". Аналіз таких ігор, як Scream Go Hero, One Hand Clapping та Phasmophobia, дозволив виявити їхні особливості та слабкі сторони, що стало важливим для подальшого проектування.

У третьому розділі визначено необхідний функціонал гри та розроблено архітектуру програмного забезпечення. Описано основні компоненти та класи, що відповідають за різні аспекти функціоналу гри, такі як керування персонажем, обробка голосових команд, головне меню, рівні, анімації тощо. Архітектура спроектована з урахуванням принципів модульності та розширюваності, що забезпечує зручну підтримку та розвиток програмного забезпечення.

Далі, у другому пункті було проведено проєктування інтерфейсу користувача, що включало створення різноманітних меню для зручного доступу до функцій гри. Це сприяло створенню привабливого та зрозумілого інтерфейсу для гравців, що підвищує задоволення від взаємодії з продуктом.

У третьому пункті були описані інструменти, що використовувалися для розробки гри, від ігрового рушія Unity до графічних програм для створення піксельної графіки та анімацій. Вибір цих інструментів був обґрунтований

їхньою потужністю, зручністю використання та можливістю інтеграції з іншими компонентами проекту.

Завершальним етапом було тестування гри, що включало функціональне тестування, ігрове тестування та збір зворотного зв'язку. Це дозволило виявити та виправити можливі недоліки перед випуском продукту.

Праця включає в себе не лише технічні аспекти розробки гри, але й глибокий аналіз вимог користувачів та конкурентного середовища. Виконання досліджень та аналізу ринку дозволило зрозуміти потреби та вимоги гравців, а також ідентифікувати прогалини в наявних ігрових продуктах, що стало основою для формулювання концепції та функціональності розроблюваної гри.

Особлива увага приділялася інтерфейсу користувача, оскільки від нього в значній мірі залежить зручність та задоволення від гри. Професійне проєктування інтерфейсу спрощує навігацію гравця, забезпечує зрозумілість функціоналу та покращує загальний імідж продукту.

Також важливим аспектом є використання оптимальних технологій та інструментів розробки, що дозволяє ефективно втілювати ідеї та забезпечує високу якість готового продукту. Вибір Unity як ігрового рушія та Visual Studio Code як середовища розробки забезпечує зручність у використанні та можливість швидкої інтеграції нового функціоналу.

Загалом, кваліфікаційна робота є комплексним підходом до розробки ігрового продукту, що поєднує в собі технічну експертизу, аналітичні навички та уважність до деталей. Результатом цієї роботи є не лише створення гри, але й задоволення потреб користувачів та створення якісного ігрового досвіду.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Платформер. URL: https://www.kelleherbros.com/blog/2024/3/4/defining-the-platformer-genre (дата звернення: 25.04.2024).
2. Scream Go Hero. URL: https://play.google.com/store/apps/details?id=com.ketchapp.screamhero&hl=en&gl=US (дата звернення: 25.04.2024).
3. One Hand Clapping. URL: https://store.steampowered.com/app/893720/One_Hand_Clapping/ (дата звернення: 25.04.2024).
4. Phasmophobia. URL: https://store.steampowered.com/app/739630/Phasmophobia/ (дата звернення: 25.04.2024).
5. Блок-схема. URL: https://miro.com/diagramming/what-is-a-block-diagram/ (дата звернення: 25.04.2024).
6. Архітектура ПЗ. URL: https://www.kursak.com/arkhitektura-prohramnoho-zabezpechennia/ (дата звернення: 25.04.2024).
7. Класи в c#. URL: https://www.w3schools.com/cs/cs_classes.php (дата звернення: 29.04.2024).
8. Unity елемент UI - текст. URL: https://www.tutorialspoint.com/unity/unity_text_element.html (дата звернення: 29.04.2024).
9. Unity що таке Canvas. URL: https://docs.unity3d.com/Packages/com.unity.ugui@1.0/manual/UICanvas.html (дата звернення: 29.04.2024).
10. Unity що таке кнопка. URL: https://docs.unity3d.com/2018.2/Documentation/ScriptReference/UI.Button.html (дата звернення: 29.04.2024).

11. Unity що таке Image. URL: https://docs.unity3d.com/ru/2018.4/ScriptReference/UI.Image.html (дата звернення: 29.04.2024).

12. Що таке Sprite в Unity. URL: https://docs.unity3d.com/ScriptReference/Sprite.html (дата звернення: 29.04.2024).

13. Компонент Canvas Group. URL: https://docs.unity3d.com/ScriptReference/CanvasGroup.html (дата звернення: 29.04.2024).

14. Що таке компонент RectTransform. URL: https://docs.unity3d.com/ScriptReference/RectTransform.html (дата звернення: 29.04.2024).

15. Що таке 2D анімація. URL: https://www.cgspectrum.com/blog/what-is-2d-animation (дата звернення: 29.04.2024).

16. Unity Animation. URL: https://docs.unity3d.com/Packages/com.unity.2d.animation@10.1/manual/index.html (дата звернення: 29.04.2024).

17. Unity Components. URL: https://docs.unity3d.com/Manual/Components.html (дата звернення: 29.04.2024).

18. Що таке методи в c#. URL: https://programm.top/uk/c-sharp/tutorial/method/ (дата звернення: 01.05.2024).

19. Метод OnTriggerEnter2D. URL: https://docs.unity3d.com/ScriptReference/MonoBehaviour.OnTriggerEnter2D.html (дата звернення: 01.05.2024).

20. Corutine: https://docs.unity3d.com/Manual/Coroutines.html (дата звернення: 01.05.2024).

21. Unity метод Start для чого використовується. URL: https://docs.unity3d.com/ScriptReference/MonoBehaviour.Start.html (дата звернення: 01.05.2024).

22. Unity метод Awake для чого використовується. URL: https://docs.unity3d.com/ScriptReference/MonoBehaviour.Awake.html (дата звернення: 01.05.2024).

23. Unity метод Update для чого використовується. URL: https://docs.unity3d.com/ScriptReference/MonoBehaviour.Update.html (дата звернення: 01.05.2024).

24. Unity метод FixedUpdate для чого використовується. URL: https://docs.unity3d.com/ScriptReference/MonoBehaviour.FixedUpdate.html (дата звернення: 01.05.2024).

25. Unity збереження завдяки PlayerPrefs. URL: https://docs.unity3d.com/ScriptReference/PlayerPrefs.html (дата звернення: 01.05.2024).

26. Що таке Asset Store. URL: https://support.unity.com/hc/en-us/articles/210142503-What-is-the-Unity-Asset-Store-and-how-do-I-purchase-Assets (дата звернення: 01.05.2024).

27. Що таке Itch.io. URL: https://itch.io/docs/general/about (дата звернення: 01.05.2024).

28. Що таке MIT ліцензія. URL: https://memgraph.com/blog/what-is-mit-license (дата звернення: 03.05.2024).

29. Що таке CC0 ліцензія. URL: https://imagesuggest.com/blog/what-is-a-creative-commons-zero-cc0-license/#:~:text=Creative%20Commons%20Zero%20(CC0)%20licenses,free%20of%20any%20use%20restrictions. (дата звернення: 01.05.2024).

30. Achivment System. URL: https://assetstore.unity.com/packages/tools/game-toolkits/achievement-system-151624 (дата звернення: 03.05.2024).

31. LeanTwean. URL: https://assetstore.unity.com/packages/tools/animation/leantween-3595 (дата звернення: 03.05.2024).

32. Pixel Art Forest. URL: https://edermunizz.itch.io/free-pixel-art-forest (дата звернення: 03.05.2024).

33. Aseprite. URL: https://store.steampowered.com/app/431730/Aseprite/ (дата звернення: 03.05.2024).

34. Photoshop. URL: https://www.techtarget.com/whatis/definition/Photoshop (дата звернення: 03.05.2024).

35. Pixel Art. URL: https://www.tokioschool.com/en/news/what-is-pixel-art/ (дата звернення: 03.05.2024).

36. Cinemachine. URL: https://unity.com/unity/features/editor/art-and-design/cinemachine (дата звернення: 03.05.2024).

37. TextMeshPro. URL: https://docs.unity3d.com/Manual/com.unity.textmeshpro.html (дата звернення: 03.05.2024).

38. Що таке тестування програмного забезпечення. URL: https://qalight.ua/baza-znaniy/shho-take-testuvannya-programnogo-zabezpechennya/ (дата звернення: 03.05.2024).

39. Що таке Unity. URL: https://lemon.school/blog/shho-take-unity (дата звернення: 04.05.2024).

40. Що таке Visual Studio Code. URL: https://www.infoworld.com/article/3666488/what-is-visual-studio-code-microsofts-extensible-code-editor.html (дата звернення: 04.05.2024).

ДОДАТОК А

Клас Player:

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

public class Player : MonoBehaviour
{
    [Header("Move Setting")]
    [SerializeField] float speed;
    [SerializeField] GameObject JumpParticleEffect;
    [SerializeField] SpriteRenderer sr;
    public static bool isDead;

    private Rigidbody2D rb2d;
    private const float voiceUpForce = 180f;
    private Animator anim;
    private AudioSource AudioJump;
    private AudioClip microphoneInput;
    private bool flapped;
    private int HeDeath = 0;
    private GameController gamecontrolle;


    [Header("Death Setting")]
    [SerializeField] CircleCollider2D collider2d;
    private CinemachineVirtualCamera VCamera;
    [SerializeField] GameObject _Player;
    [SerializeField] GameObject DeathEffectPrefab;
    [SerializeField] CanvasGroup DeathEffect;


    [Header("SpawnSetting")]
    [SerializeField] GameObject SpawnEffectPrefab;


    public static bool PlayerMove;

    private void Awake()
    {
        if (Microphone.devices.Length > 0)
        {
            //initializing scripting
            microphoneInput = Microphone.Start(Microphone.devices[0], true, 999, 44100);
        }
    }

    // Use this for initialization
    void Start()
    {
        AudioJump = GetComponent<AudioSource>();
        rb2d = GetComponent<Rigidbody2D>();
        anim = GetComponent<Animator>();
```

```
    StartSpawn();
    VCamera = GameObject.FindObjectOfType<CinemachineVirtualCamera>();
    VCamera.Follow = _Player.transform;
    gamecontrolle = GameObject.FindObjectOfType<GameController>();
    Trap_12.IsFlip = false;
    FlapPlayer();
}

public void StartSpawn()
{
    anim.SetTrigger("Spawn");
    Instantiate(SpawnEffectPrefab, transform.position, transform.rotation);

    collider2d.isTrigger = false;
    isDead = false;
    GameController.isPaused = false;
    flapped = true;

    HeDeath = 0;
}

public void FinishSpawnAnimation()
{
    anim.SetTrigger("Idel");
}

// Update is called once per frame
void Update()
{
    //if the player is dead, stop the game
    LoadVoiceSensitivity();
    if (isDead == false)
    {
        if (SwitchControll.SwitchControlls == 0)
        {
            FlapPlayerWithVoice();
        }
        else
        {
            if (Input.GetMouseButtonDown(0))
            {
                if (GameController.isPaused == false)
                {
                    FlapPlayer();
                }
            }
        }
    }
}

private void OnTriggerEnter2D(Collider2D collision)
{
    if (collision.gameObject.tag == ("Trap"))
    {
        if (HeDeath == 0)
        {
            gamecontrolle.Died();
            StartCoroutine(CameraShake.Shake(3, 3f, 0.3f));
            float EffectSize = 88;
```

```
                LeanTween.scale(DeathEffect.gameObject, new Vector3(EffectSize, EffectSize, EffectSize), 0.5f);
                DeathEffect.LeanAlpha(0, 0.5f);


                StartSound("Death");
                Secret_3.AddDeath();
                isDead = true;
                rb2d.velocity = Vector2.zero;
                anim.SetTrigger("Dead");
                HeDeath += 1;
            }
        }

        if (collision.gameObject.tag == ("Bullet"))
        {
            if (HeDeath == 0)
            {
                gamecontrolle.Died();
                StartCoroutine(CameraShake.Shake(3, 3f, 0.3f));
                float EffectSize = 88;
                LeanTween.scale(DeathEffect.gameObject, new Vector3(EffectSize, EffectSize, EffectSize), 0.5f);
                DeathEffect.LeanAlpha(0, 0.5f);


                StartSound("Death");
                isDead = true;
                rb2d.velocity = Vector2.zero;
                anim.SetTrigger("Dead");
                HeDeath += 1;
            }
        }
    }


    void FlapPlayer()
    {
        if (Trap_12.IsFlip == false)
        {
            rb2d.velocity = transform.right * speed;
        }
        else
        {
            rb2d.velocity = transform.right * -speed;
        }
        //velocity is either rising or falling. Every time we push the jump button we get always the time response.
        //rb2d.velocity = Vector2.zero;
        //adding some force to rb2d (we do not want to change horizontally the player because the world is moving
around him)
        rb2d.AddForce(new Vector2(0, voiceUpForce));
        anim.SetTrigger("Flap");
        if (PlayerPrefs.GetInt("IsSound") == 0)
        {
            AudioJump.Play();
            //SoundSetting.PlaySound("Flap");
        }
        Instantiate(JumpParticleEffect, transform.position, transform.rotation);
    }
```

```
void FlapPlayerWithVoice()
{
    //get mic volume
    int dec = 128;
    float[] waveData = new float[dec];
    int micPosition = Microphone.GetPosition(null) - (dec + 1); // null means the first microphone
    microphoneInput.GetData(waveData, micPosition);

    // Getting a peak on the last 128 samples
    float levelMax = 0;

    for (int i = 0; i < dec; i++)
    {
        float wavePeak = waveData[i] * waveData[i];
        if (levelMax < wavePeak) levelMax = wavePeak;
    }

    float level = Mathf.Sqrt(Mathf.Sqrt(levelMax));

    if (level > SettingsController.voiceSensitivity && !flapped)
    {
        if (GameController.isPaused == false)
        {
            FlapPlayer();
        }
        flapped = true;
    }

    if (level < SettingsController.voiceSensitivity && flapped)
    {
        flapped = false;
    }
}

private void LoadVoiceSensitivity()
{
    SettingsController.voiceSensitivity = PlayerPrefs.GetFloat("sensitivity", 0.71f);
}

public void CloseCollider()
{
    Instantiate(DeathEffectPrefab, transform.position, transform.rotation);
    VCamera.Follow = null;
    collider2d.isTrigger = true;
}

private void StartSound(string SoundName)
{
    if (PlayerPrefs.GetInt("IsSound") == 0)
    {
        SoundSetting.PlaySound(SoundName);
    }
}

public void FlipAnimation()
{
    if (Trap_12.IsFlip == true)
```

```
    {
        sr.flipX = true;
    }
    else
    {
        sr.flipX = false;
    }
  }
}
```

Клас GameController:

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

public class GameController : MonoBehaviour {

  [Header("Death Menu")]
public GameObject gameOverText;
public bool gameOver = false;
  [Header("Paused Menu")]
  public static bool isPaused = false;
public GameObject pauseMenu;
  [SerializeField] CanvasGroup LosePanelBackground;
  [SerializeField] GameObject RestartButton;
  [SerializeField] GameObject GoToMenuButton;
  [SerializeField] GameObject LoseText;
  [SerializeField] CanvasGroup Transition;


  private int StartAnimation;
  private Loadingmanager loadingBar;
  public float scrollSpeed = -1.5f;
  void Start()
  {
    StartAnimation = 0;
    GoToMenuButton.transform.localScale = new Vector3(0, 0, 0);
    RestartButton.transform.localScale = new Vector3(0,0,0);
    LoseText.transform.localScale = new Vector3(0, 0, 0);
    LosePanelBackground.gameObject.SetActive(false);
    LosePanelBackground.alpha = 0;
    isPaused = false;
    StartCoroutine(StartTransition());
    Screen.sleepTimeout = SleepTimeout.NeverSleep;
    loadingBar = GameObject.FindObjectOfType<Loadingmanager>();
  }


  void Update()
  {
    if(Input.GetKeyDown(KeyCode.Escape))
    {
      if(LevelDone.IsDoneLevel == false)
      {
        if (Player.isDead == false)
        {
          PauseGame();
```

```
        }
       }
    }
}


IEnumerator StartTransition()
{
    Transition.LeanAlpha(0, 0.2f);
    yield return new WaitForSeconds(0.2f);
    Transition.gameObject.SetActive(false);
}

public void Died()
{
    gameOver = true;
    Invoke("WaitLose", 2f);
}

private void WaitLose()
{
    LosePanelBackground.gameObject.SetActive(true);
    LosePanelBackground.LeanAlpha(1, 0.3f);
    StartCoroutine(StartAnim());
}

IEnumerator AnimButton(GameObject obj, float Time, float StartSize, float EndSize)
{
    LeanTween.scale(obj, new Vector3(StartSize, StartSize, StartSize), 0.2f);
    yield return new WaitForSeconds(Time);
    LeanTween.scale(obj, new Vector3(EndSize, EndSize, EndSize), 0.2f);
}

IEnumerator StartAnim()
{
    yield return new WaitForSeconds(0.3f);
    StartCoroutine(AnimButton(RestartButton, 0.2f, 1.308801f, 1.138088f));
    yield return new WaitForSeconds(0.3f);
    StartCoroutine(AnimButton(GoToMenuButton, 0.2f, 0.7714439f, 0.7013126f));
    yield return new WaitForSeconds(0.3f);
    StartCoroutine(AnimButton(LoseText, 0.2f, 1.1f, 1f));
}


private void StartAnim1()
{
    StartCoroutine(AnimButton(RestartButton, 0.2f, 1.308801f, 1.138088f));
}

private void StartAnim3()
{
    StartCoroutine(AnimButton(LoseText, 0.2f, 1.1f, 1f));
}

public void DeletePrefs()
{
    PlayerPrefs.DeleteAll();
}
```

```
public void GoToMenu(string NameScene)
{
   StartSound("Click");
   ResumeGame();
   StartCoroutine(loadingBar.LoadLevelAsync(NameScene));
}

public void ChangeToScene(string sceneToChangeTo)
{
   StartSound("Click");
   StartCoroutine(ChangeToSceneWhait(sceneToChangeTo));
}

IEnumerator ChangeToSceneWhait(string sceneToChangeTo)
{
   yield return new WaitForSeconds(0.2f);
   if (isPaused)
      ResumeGame();
   StartCoroutine(loadingBar.LoadLevelAsync(sceneToChangeTo));
}

public void RestartGame()
{
   if(StartAnimation == 0)
   {
      StartAnimation++;
      StartSound("Click");
      Invoke("RestartWait", 0.2f);
   }
}

private void RestartWait()
{
   SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex);
}



public void PauseGame()
{
   StartSound("Click");
   if (!isPaused)
   {
      isPaused = true;
      pauseMenu.SetActive(true);
      Time.timeScale = 0;
   }
   else
   {
      isPaused = false;
      pauseMenu.SetActive(false);
      Time.timeScale = 1;
   }
}

public void ResumeGame()
{
```

```
 StartSound("Click");
        if (isPaused)
        {
            isPaused = false;
            pauseMenu.SetActive(false);
            Time.timeScale = 1;
        }
    }

    private void StartSound(string SoundName)
    {
        if (PlayerPrefs.GetInt("IsSound") == 0)
        {
            SoundSetting.PlaySound(SoundName);
        }
    }
}
```

Клас SettingController:

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

public class SettingsController : MonoBehaviour {

    [HideInInspector]
    public static float voiceSensitivity = 0.3f;
    public Slider slider;

    public void ChangeToScene (string sceneToChangeTo) {
            SceneManager.LoadScene (sceneToChangeTo);
}

    private void Update()
    {
        LoadPrefs();
    }

    public void AdjustVoiceSensitivity(float newSensitivity)
    {
        voiceSensitivity = newSensitivity;
        PlayerPrefs.SetFloat("sensitivity", newSensitivity);
    }

    private void LoadPrefs()
    {
        //loading Voice Sensitivity
        voiceSensitivity = PlayerPrefs.GetFloat("sensitivity", 0.3f);
        slider.value = voiceSensitivity;
    }
}
```

Клас ShopMenu:

```
using System.Collections;
```

```
using UnityEngine;
using UnityEngine.UI;

[System.Serializable]
class HatSetting // Interactive customization of each hat separately
{
   [Tooltip("It's HAT, HAT!")] public GameObject[] HatSkin;
   public GameObject[] HatName;
   [Tooltip("Coordinates where the hat will appear [Y]")] public float[] HatY;
}

public class ShopMenu : MonoBehaviour
{
   [Tooltip("class with cap values")] [SerializeField] HatSetting[] HatSetting;
   [Tooltip("Buttons for switching caps")] [SerializeField] GameObject Right, Left;
   [Tooltip("Player Skin")] [SerializeField] GameObject[] PlayerSkin;
   [Tooltip("Object with buy and open header buttons")] [SerializeField] GameObject[] HatButton;
   [Tooltip("Button that closes the shopping menu")] [SerializeField] GameObject CloseButton;
   [Tooltip("An object that stores all the caps settings - [Name, Sprite, Buttons]")] [SerializeField] GameObject[] Hat;
   [SerializeField] GameObject HatCounterobject;
   [Tooltip("text that displays the page in the store")] [SerializeField] Text CountText;
   [Tooltip("How many hats in the store")] [SerializeField] int Counter;
   [Tooltip("Page number")] int page;
   [Tooltip("Button Effects")] [SerializeField] LeanTweenType MenuBtnType;
   private MainMenu mainmenu;
   private SecretLevelMenuSetting secretlevel;
   public static bool IsOpenShop = false;


   private void Start()
   {
      CountText.text = page + 1 + "/" + Counter;
      HatsMenuClosed();
      IsOpenShop = true;
      page = 0;
      secretlevel = GameObject.FindObjectOfType<SecretLevelMenuSetting>();
      mainmenu = GameObject.FindObjectOfType<MainMenu>();
      for (int i = 0; i < HatSetting.Length; i++) // Checks open menus
      {
         if (i == page)
         {
            if (SecretLevelMenuSetting.SecretMenuOpen == false && MainMenu.LevelMenuIsOpen == true)
            {
               StartCoroutine(CloseLevelMenuAndOpenShop(i));
            }
            else if(SecretLevelMenuSetting.SecretMenuOpen == true && MainMenu.LevelMenuIsOpen == false)
            {
               StartCoroutine(CloseSecretLevelMenuAndOpenShop(i));
            }
            else if (SecretLevelMenuSetting.SecretMenuOpen == false && MainMenu.LevelMenuIsOpen == false)
            {
               StartCoroutine(StartShopMenu(i, 0));
            }
         }
      }
   }
```

```
    IEnumerator CloseLevelMenuAndOpenShop(int i) // closed Main Level Menu
    {
        mainmenu.CloseLevelMenuAndOpenShop();
        yield return new WaitForSeconds(1);
        StartCoroutine(StartShopMenu(i, 0));
    }

    IEnumerator CloseSecretLevelMenuAndOpenShop(int i) // closed Secret Level Menu
    {
        secretlevel.CloseSecretLevelMenuAndOpenShop();
        yield return new WaitForSeconds(1);
        StartCoroutine(StartShopMenu(i, 0));
    }


    private void HatsMenuClosed() // Closed all hats Besides Hat - [1]
    {
        for (int i = 1; i < Hat.Length; i++)
        {
            Hat[i].SetActive(false);
        }
    }


    IEnumerator StartShopMenu(int Hat, int Skin) // Open shop menu
    {
        StartSound("Transition");
        LeanTween.moveX(Left.gameObject.GetComponent<RectTransform>(), -311f, 0.2f);
        yield return new WaitForSeconds(0.4f);
        StartSound("Transition");
        LeanTween.moveX(Right.gameObject.GetComponent<RectTransform>(), 311f, 0.2f);
        yield return new WaitForSeconds(0.4f);
        StartSound("Transition");
        StartCoroutine(AnimButton(PlayerSkin[Skin], 0.2f, 2.5f, 1.8865f));
        yield return new WaitForSeconds(0.5f);
        StartSound("Transition");
        LeanTween.moveY(HatSetting[Hat].HatSkin[0].gameObject.GetComponent<RectTransform>(),
HatSetting[Hat].HatY[0], 0.2f);
        StartCoroutine(SpawnHatObjects(true, HatSetting[Hat].HatName[0], 0.9f, 0.8f));
        StartCoroutine(AnimButton(HatButton[Hat], 0.2f, 5f, 4.7522155f));
        yield return new WaitForSeconds(0.4f);
        StartSound("Transition");
        LeanTween.moveY(CloseButton.gameObject.GetComponent<RectTransform>(), -283f, 0.3f);
        LeanTween.scale(HatCounterobject, new Vector3(0.57586f, 0.57586f, 0.57586f), 0.2f);
    }

    public void CloseShopMenu()
    {
        for (int i = 0; i < HatSetting.Length; i++)
        {
            if (i == page)
            {
                StartCoroutine(CloseShopMenu(i, 0));
            }
        }
    }
```

```
IEnumerator CloseShopMenu(int Hat, int Skin) // Open shop menu
{
   StartSound("Transition");
   LeanTween.moveY(CloseButton.gameObject.GetComponent<RectTransform>(), -1500f, 0.3f);
   LeanTween.scale(HatCounterobject, new Vector3(0f, 0f, 0f), 0.2f);
   yield return new WaitForSeconds(0.4f);
   StartSound("Transition");
   LeanTween.moveY(HatSetting[Hat].HatSkin[0].gameObject.GetComponent<RectTransform>(),  963f,
0.2f);
   StartCoroutine(SpawnHatObjects(false, HatSetting[Hat].HatName[0], 0.9f, 0.8f));
   StartCoroutine(AnimButton(HatButton[Hat], 0.2f, 5f, 0f));
   yield return new WaitForSeconds(0.5f);
   StartSound("Transition");
   StartCoroutine(AnimButton(PlayerSkin[Skin], 0.2f, 2.5f, 0f));
   yield return new WaitForSeconds(0.4f);
   StartSound("Transition");
   LeanTween.moveX(Right.gameObject.GetComponent<RectTransform>(), 1517f, 0.2f);
   yield return new WaitForSeconds(0.4f);
   StartSound("Transition");
   LeanTween.moveX(Left.gameObject.GetComponent<RectTransform>(), -1517f, 0.2f);
   mainmenu.CloseShop();
   yield return new WaitForSeconds(0.4f);
   IsOpenShop = false;
   Destroy(gameObject);
}

IEnumerator SpawnHatObjects(bool Where, GameObject obj, float StartSize, float EndSize) // Zoom Effect
{
   yield return new WaitForSeconds(0.2f);
   if(Where == true)
   {
      StartCoroutine(AnimButton(obj, 0.2f, StartSize, EndSize));
   }
   else
   {
      StartCoroutine(AnimButton(obj, 0.2f, 0f, 0f));
   }
}

IEnumerator AnimButton(GameObject obj, float Time, float StartSize, float EndSize) // Zoom Effect 2
{
   LeanTween.scale(obj, new Vector3(StartSize, StartSize, StartSize), 0.2f);
   yield return new WaitForSeconds(Time);
   LeanTween.scale(obj, new Vector3(EndSize, EndSize, EndSize), 0.2f);
}


public void RightButtonClick()
{
   LeanTween.scale(Right, new Vector3(1.2f, 1.2f, 0), 0.4f).setEase(MenuBtnType);
   StartSound("Click");
   if (page < HatSetting.Length - 1)
   {
      page += 1;
      CountText.text = page + 1 + "/" + Counter;
      for (int i = 0; i < HatSetting.Length; i++)
      {
         if (i == page)
```

```
                {
                    Hat[i].SetActive(true);
                    HatSetting[i].HatSkin[0].SetActive(true);
                    LeanTween.moveY(HatSetting[i].HatSkin[0].gameObject.GetComponent<RectTransform>(),
HatSetting[i].HatY[0], 0.2f);
                    StartCoroutine(SpawnHatObjects(true, HatSetting[i].HatName[0], 0.9f, 0.8f));
                    StartCoroutine(SpawnHatObjects(true, HatButton[i], 5, 4.7522155f));
                }
                else
                {
                    Hat[i].SetActive(false);
                    HatSetting[i].HatSkin[0].SetActive(false);
                    LeanTween.moveY(HatSetting[i].HatSkin[0].gameObject.GetComponent<RectTransform>(),
963f, 0.2f);
                    StartCoroutine(SpawnHatObjects(false, HatSetting[i].HatName[0], 0.9f, 0.8f));
                    StartCoroutine(SpawnHatObjects(false, HatButton[i], 5, 4.7522155f));
                }
            }
        }
    }

    public void LeftButtonClick()
    {
        LeanTween.scale(Left, new Vector3(1.2f, 1.2f, 0), 0.4f).setEase(MenuBtnType);
        StartSound("Click");
        if (page > 0)
        {
            page -= 1;
            CountText.text = page + 1 + "/" + Counter;
            for (int i = 0; i < HatSetting.Length; i++)
            {
                if (i == page)
                {
                    Hat[i].SetActive(true);
                    HatSetting[i].HatSkin[0].SetActive(true);
                    LeanTween.moveY(HatSetting[i].HatSkin[0].gameObject.GetComponent<RectTransform>(),
HatSetting[i].HatY[0], 0.2f);
                    StartCoroutine(SpawnHatObjects(true, HatSetting[i].HatName[0], 0.9f, 0.8f));
                    StartCoroutine(SpawnHatObjects(true, HatButton[i], 5, 4.7522155f));
                }
                else
                {
                    Hat[i].SetActive(false);
                    HatSetting[i].HatSkin[0].SetActive(false);
                    LeanTween.moveY(HatSetting[i].HatSkin[0].gameObject.GetComponent<RectTransform>(),
963f, 0.2f);
                    StartCoroutine(SpawnHatObjects(false, HatSetting[i].HatName[0], 0.9f, 0.8f));
                    StartCoroutine(SpawnHatObjects(false, HatButton[i], 5, 4.7522155f));
                }
            }
        }
    }

    private void StartSound(string SoundName)
    {
        if (PlayerPrefs.GetInt("IsSound") == 0)
        {
            SoundSetting.PlaySound(SoundName);
```

```
        }
    }
}
```

Клас Trap_2

```
using UnityEngine;
public class Trap_2 : MonoBehaviour
{
    [Tooltip("Starting position of the object")] private Vector3 startPos;
    [Tooltip("The target object to which the movement begins")] [SerializeField] Transform target;
    [Tooltip("Object movement speed")] [SerializeField] float speed;
    [Tooltip("Checking in which direction the object is moving")] private bool moveUp;
    void Start()
    {
        startPos = transform.position;
        moveUp = true;
    }
    void Update()
    {
        float step = speed * Time.deltaTime;
        if (transform.position == target.position)
        {
            moveUp = false;
        }
        else if (transform.position == startPos)
        {
            moveUp = true;
        }
        if (moveUp == false)
        {
            transform.position = Vector3.MoveTowards(transform.position, startPos, step);
        }
        else if (moveUp)
        {
            transform.position = Vector3.MoveTowards(transform.position, target.position, step);
        }
    }
}
```

Клас LevelDone

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

public class LevelDone : MonoBehaviour
{
    public static int LevelFinish = 0;
    [SerializeField] CanvasGroup FinishPanel;
    [SerializeField] GameObject[] LockStar;
    [SerializeField] GameObject[] UnLockStar;
    [SerializeField] CanvasGroup Transition;
    [SerializeField] GameObject NextLevelButton;
    [Tooltip("Check if it's a secret level")] [SerializeField] bool SecretLevel;
    [SerializeField] int SecretLevelNumber;
    public static bool IsDoneLevel;
```

```
void Start()
{
    IsDoneLevel = false;
    GameController.isPaused = false;
    Transition.gameObject.SetActive(false);
    Transition.alpha = 0;
    FinishPanel.gameObject.SetActive(false);
    FinishPanel.alpha = 0;
    LevelFinish = 0;
}

private void OnTriggerEnter2D(Collider2D collision)
{
    if(collision.CompareTag("Player"))
    {
        IsDoneLevel = true;
        StartSound("PortalTransition");
        LockStart();
        LevelFinish = 1;
        if(SecretLevel == false)
        {
            UnLockLevel();
        }
        else
        {
            PlayerPrefs.SetInt("SecretLevelDone" + SecretLevelNumber, 1);
        }
    }
}

public void LockStart()
{
    GameController.isPaused = true;
    FinishPanel.gameObject.SetActive(true);
    FinishPanel.LeanAlpha(1, 0.2f);
    StartCoroutine(StartAnim());
}

IEnumerator OneStar()
{
    yield return new WaitForSeconds(0.3f);
    StartSound("Transition");
    StartCoroutine(AnimButton(UnLockStar[0], 0.2f, 1.1f, 1f));
    LeanTween.moveY(NextLevelButton.gameObject.GetComponent<RectTransform>(), -249f, 0.2f);
}

IEnumerator TwoStar()
{
    yield return new WaitForSeconds(0.3f);
    StartSound("Transition");
    StartCoroutine(AnimButton(UnLockStar[0], 0.2f, 1.1f, 1f));
    yield return new WaitForSeconds(0.3f);
    StartSound("Transition");
    StartCoroutine(AnimButton(UnLockStar[1], 0.2f, 1.1f, 1f));
    LeanTween.moveY(NextLevelButton.gameObject.GetComponent<RectTransform>(), -249f, 0.2f);
}

IEnumerator ThreeStar()
```

```
{
    yield return new WaitForSeconds(0.3f);
    StartSound("Transition");
    StartCoroutine(AnimButton(UnLockStar[0], 0.2f, 1.1f, 1f));
    yield return new WaitForSeconds(0.3f);
    StartSound("Transition");
    StartCoroutine(AnimButton(UnLockStar[1], 0.2f, 1.1f, 1f));
    yield return new WaitForSeconds(0.3f);
    StartSound("Transition");
    StartCoroutine(AnimButton(UnLockStar[2], 0.2f, 1.1f, 1f));
    LeanTween.moveY(NextLevelButton.gameObject.GetComponent<RectTransform>(), -249f, 0.2f);
}

private void StartSound(string SoundName)
{
    if (PlayerPrefs.GetInt("IsSound") == 0)
    {
        SoundSetting.PlaySound(SoundName);
    }
}

IEnumerator StartAnim()
{
    yield return new WaitForSeconds(0.3f);
    StartCoroutine(AnimButton(LockStar[0], 0.2f, 2.228451f, 1.796268f));
    yield return new WaitForSeconds(0.3f);
    StartCoroutine(AnimButton(LockStar[1], 0.2f, 2.228451f, 1.796268f));
    yield return new WaitForSeconds(0.3f);
    StartCoroutine(AnimButton(LockStar[2], 0.2f, 2.228451f, 1.796268f));
    yield return new WaitForSeconds(0.4f);
    if (StarOnLevelUI.StarCount == 1)
    {
        StartCoroutine(OneStar());
    }
    else if (StarOnLevelUI.StarCount == 2)
    {
        StartCoroutine(TwoStar());
    }
    else if (StarOnLevelUI.StarCount == 3)
    {
        StartCoroutine(ThreeStar());
    }
    else
    {
        LeanTween.moveY(NextLevelButton.gameObject.GetComponent<RectTransform>(), -249f, 0.2f);
    }
}

IEnumerator AnimButton(GameObject obj, float Time, float StartSize, float EndSize)
{
    LeanTween.scale(obj, new Vector3(StartSize, StartSize, StartSize), 0.2f);
    yield return new WaitForSeconds(Time);
    LeanTween.scale(obj, new Vector3(EndSize, EndSize, EndSize), 0.2f);
}


public void UnLockLevel()
{
```

```
        int currentLevel = SceneManager.GetActiveScene().buildIndex;

        if (currentLevel >= PlayerPrefs.GetInt("levels"))
        {
            PlayerPrefs.SetInt("levels", currentLevel + 1);
        }
    }


    public void NextLevel(string SceneName) // Turn on dimming and go to NEXT LEVEL
    {
        IsDoneLevel = false;
        StartSound("Click");
        StartCoroutine(NextLevelStart(SceneName));
    }
    IEnumerator NextLevelStart(string SceneName)
    {
        Transition.gameObject.SetActive(true);
        Transition.LeanAlpha(1, 0.2f);
        yield return new WaitForSeconds(0.3f);
        SceneManager.LoadScene(SceneName);
    }
}
```

ДОДАТОК Б

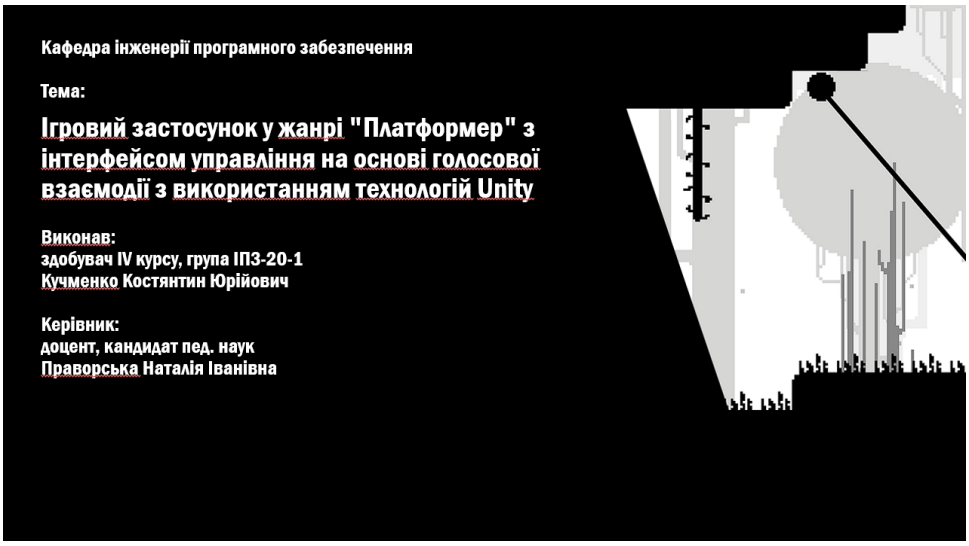


Рисунок 1 – слайд презентації №1

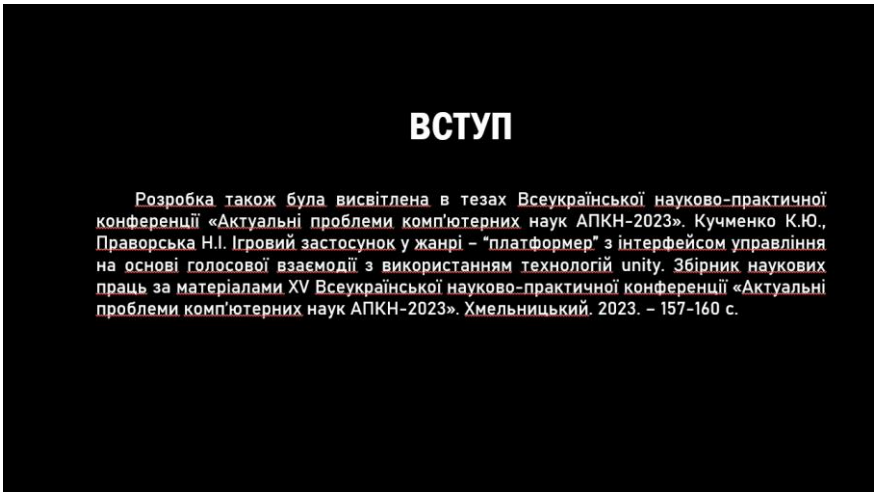


Рисунок 2 – слайд презентації №2

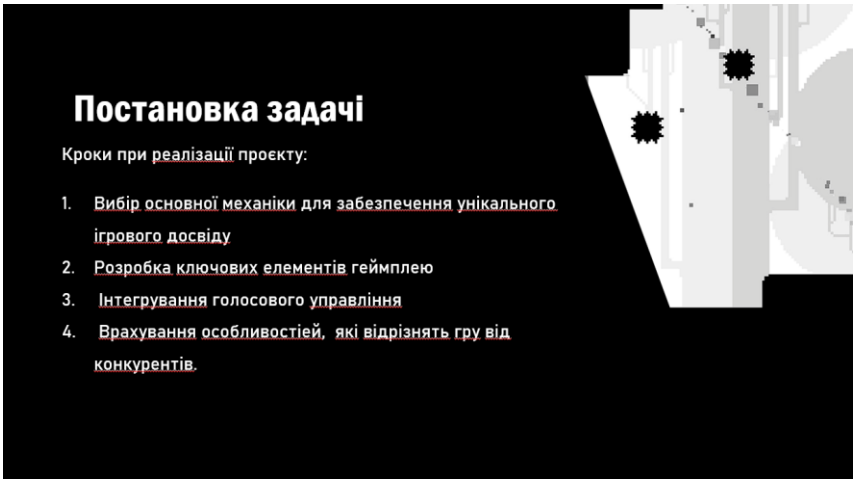


Рисунок 3 – слайд презентації №3

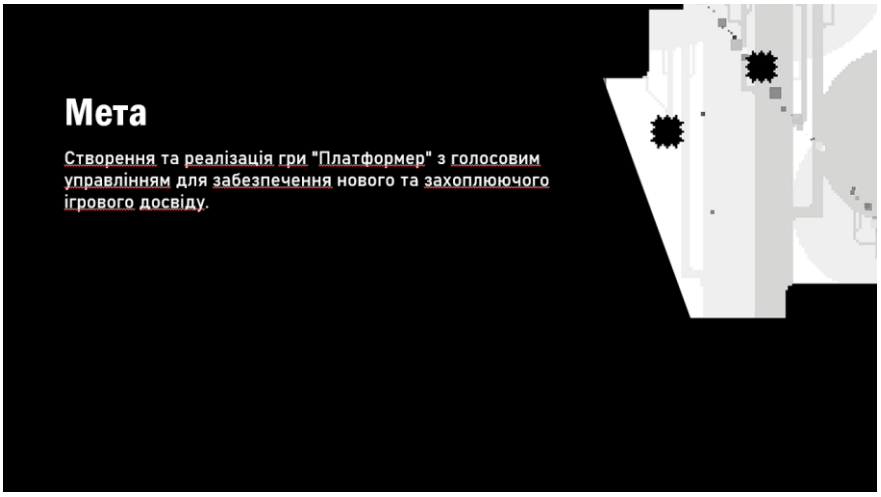


Рисунок 4 – слайд презентації №4

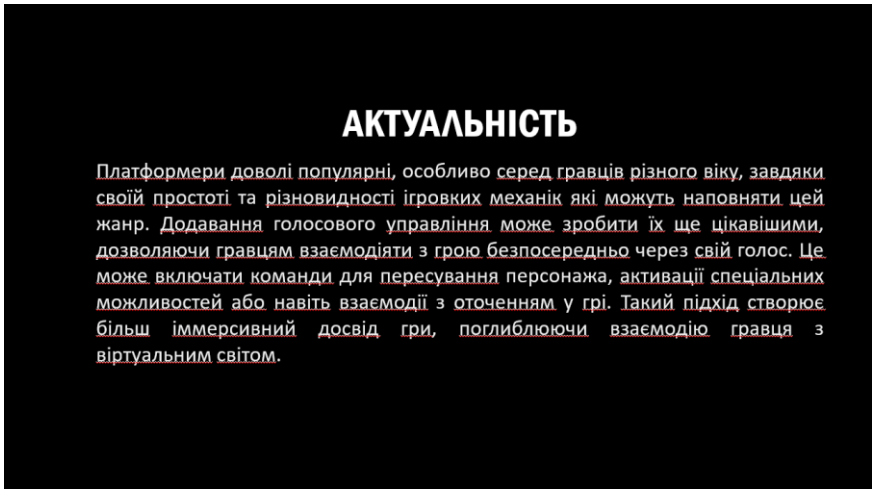


Рисунок 5 – слайд презентації №5

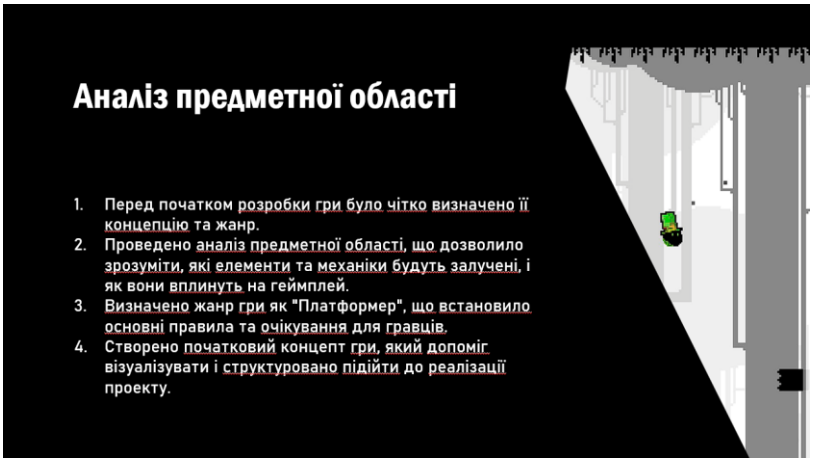


Рисунок 6 – слайд презентації №6

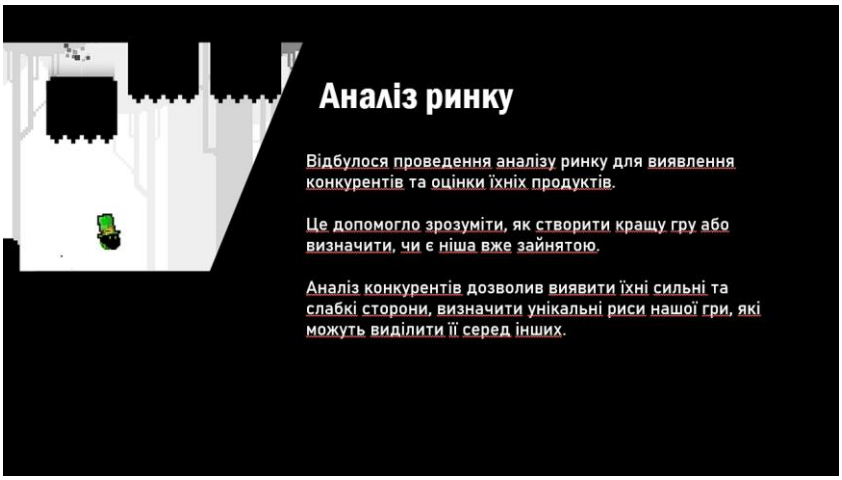


Рисунок 7 – слайд презентації №7

Рисунок 8 – слайд презентації №8

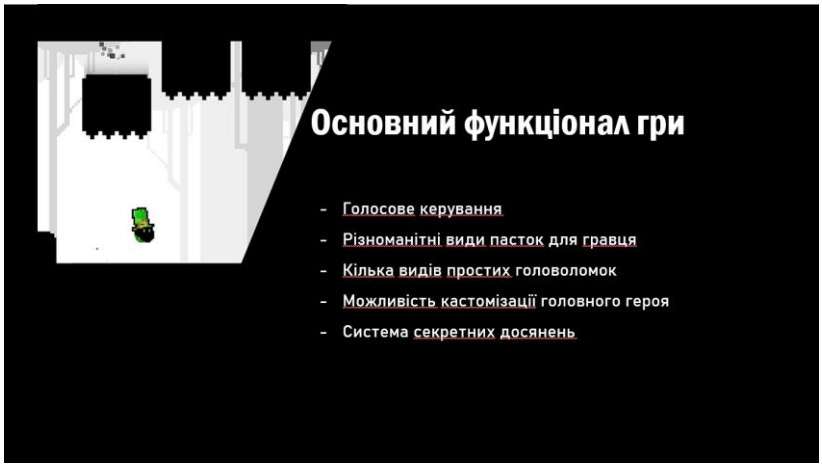


Рисунок 9 – слайд презентації №9

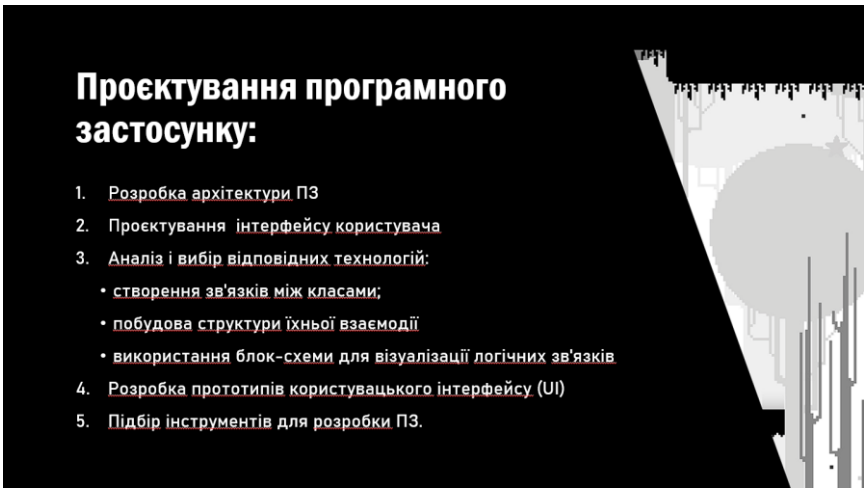


Рисунок 10 – слайд презентації №10

Рисунок 11 – слайд презентації №11

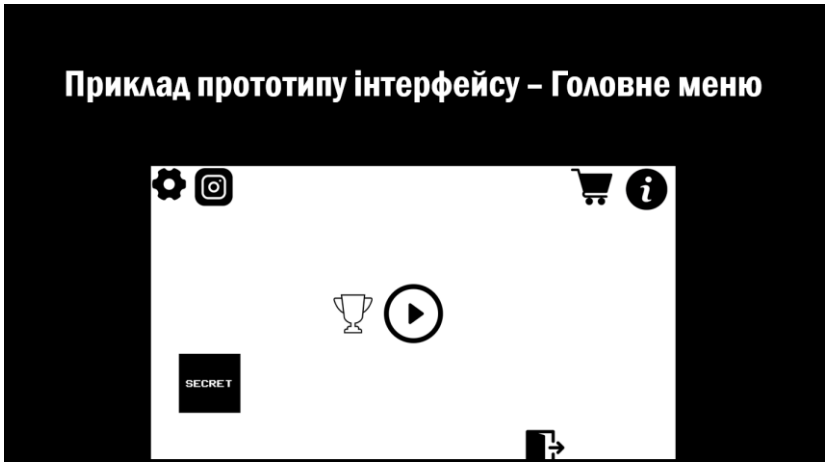


Рисунок 12 – слайд презентації №12

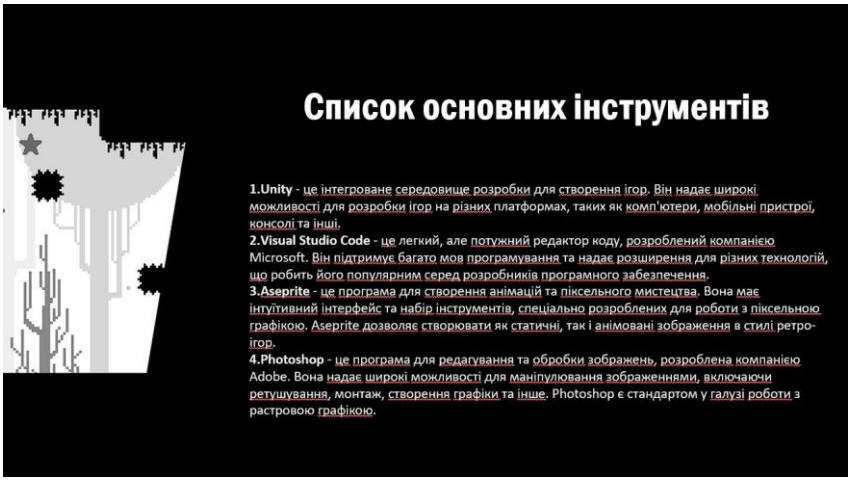


Рисунок 13 – слайд презентації №13

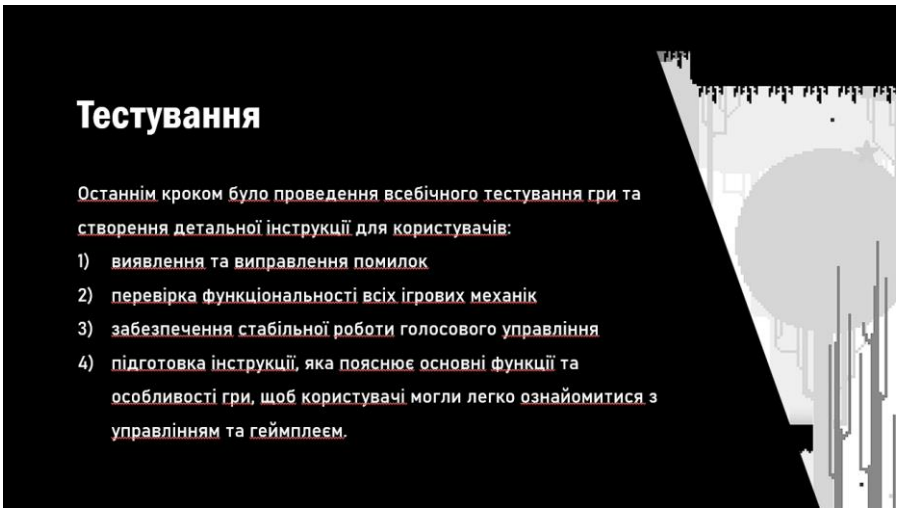


Рисунок 14 – слайд презентації №14

Рисунок 15 – слайд презентації №15

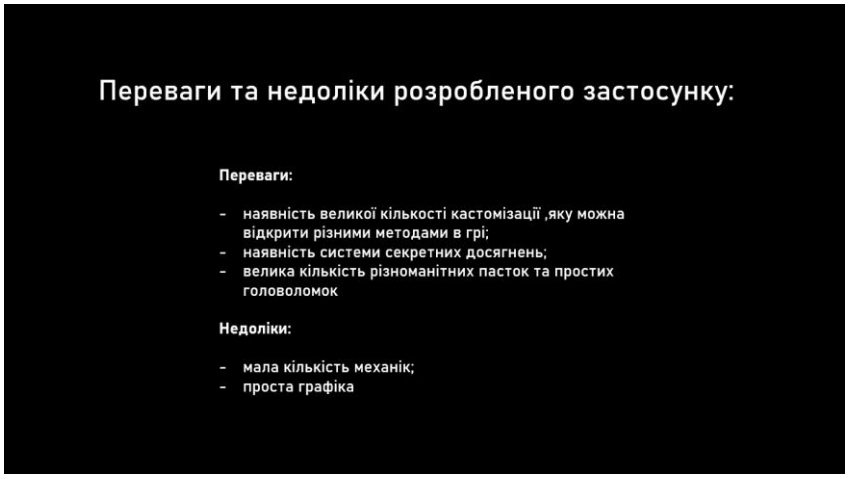


Рисунок 16 – слайд презентації №16

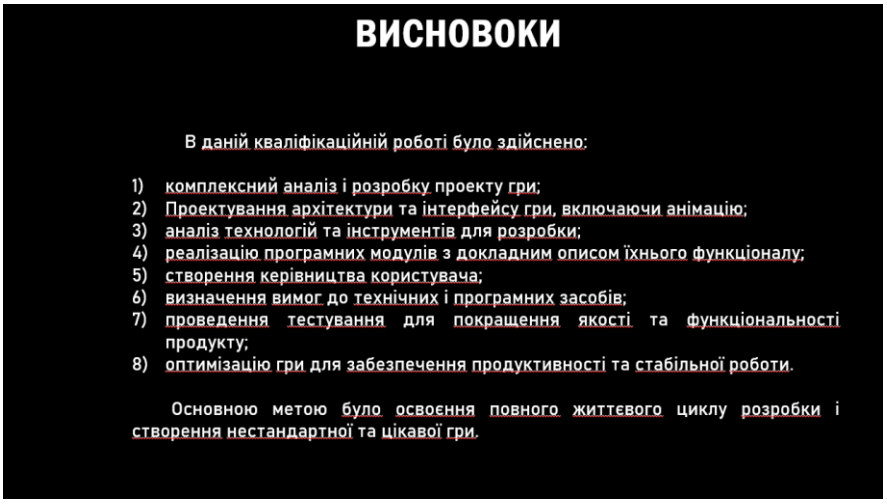
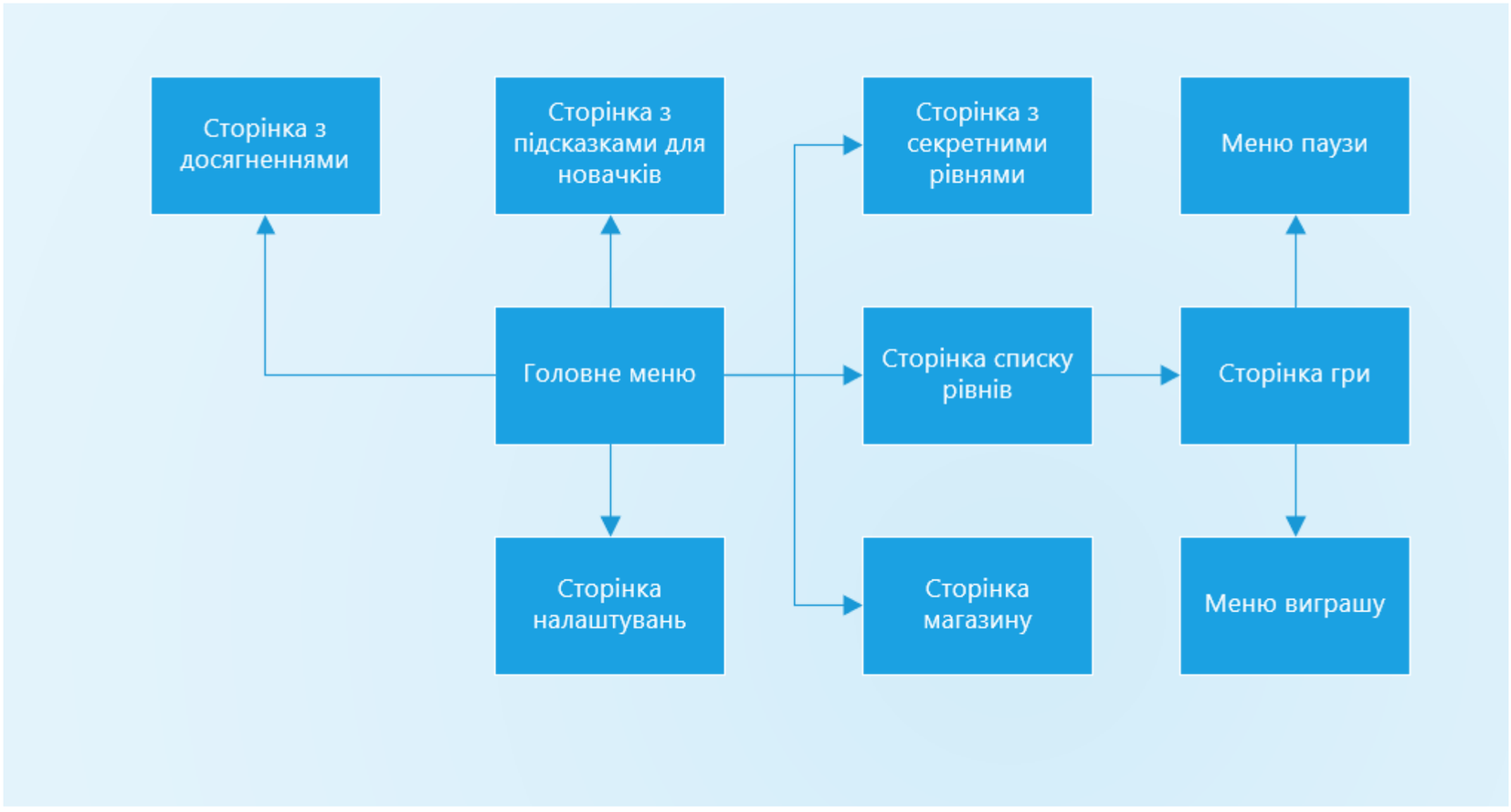
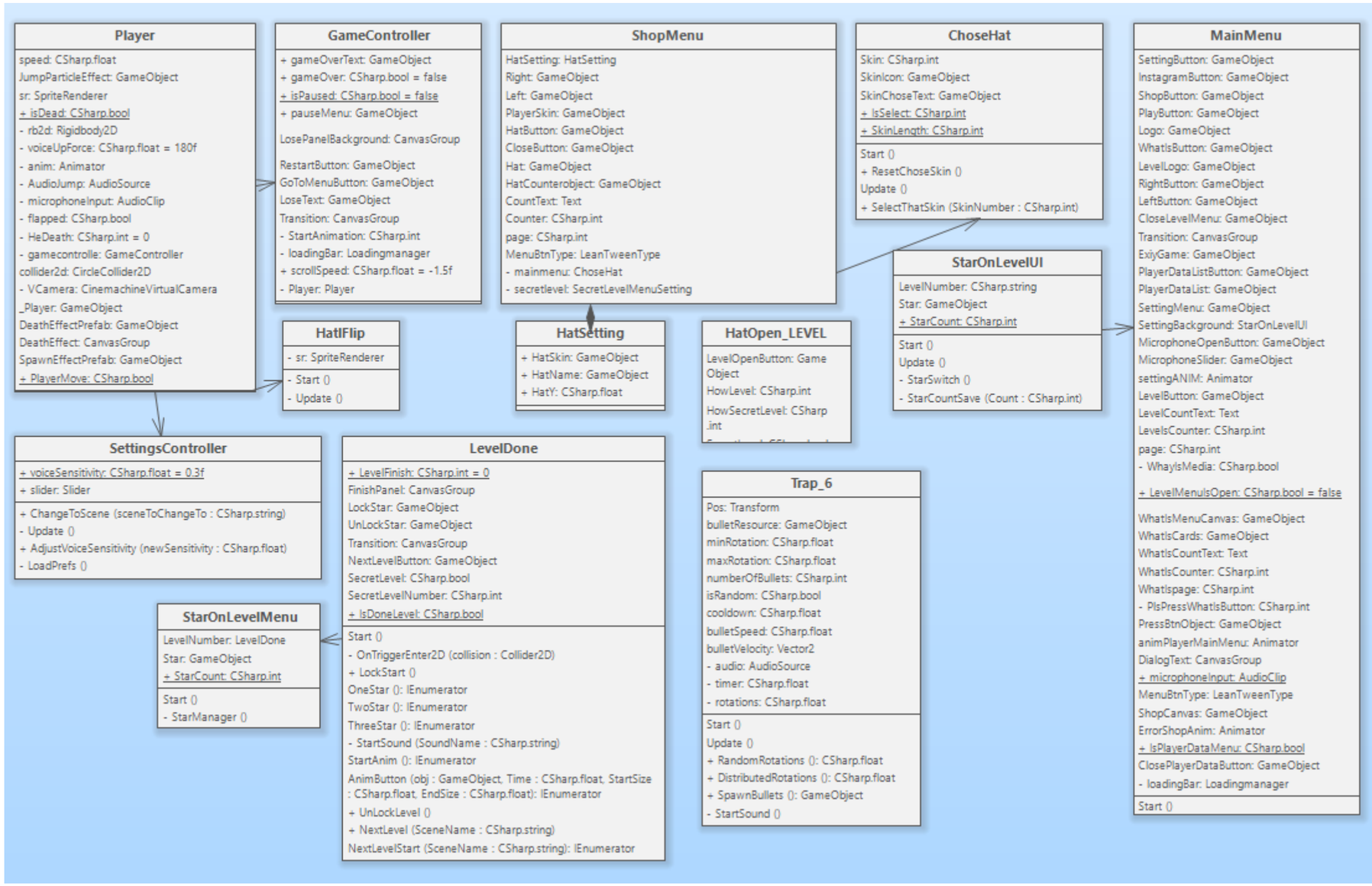


Рисунок 17 – слайд презентації №17

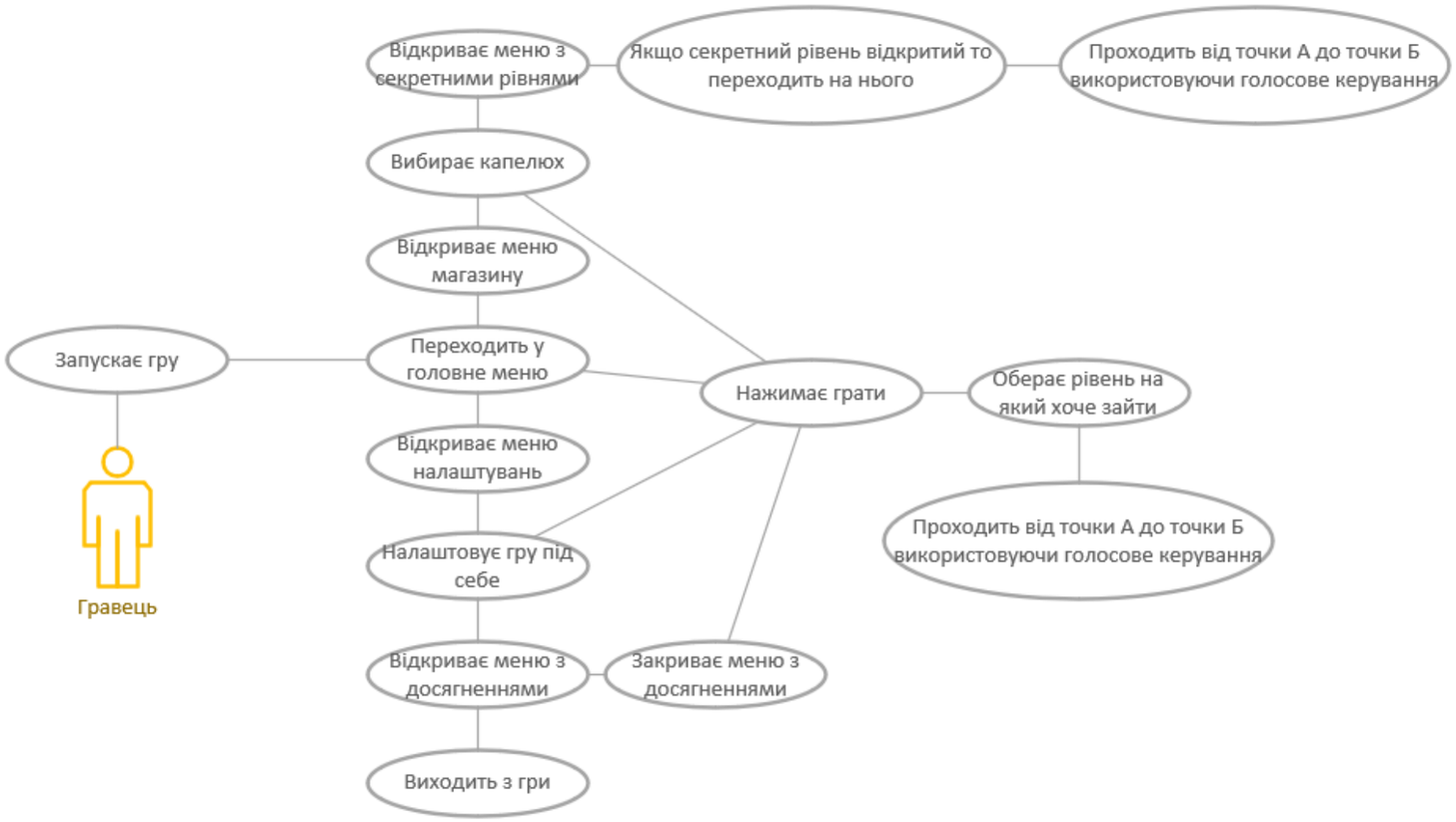
ГРАФІЧНА ЧАСТИНА



					КвРІПЗ. 200166.01.13.ПЗ			
					Ігровий застосунок у жанрі "Платформер" з інтерфейсом управління на основі голосової взаємодії з використанням технологій Unity	Літера	Маса	Масштаб
Зм.	Арк.	№ докум.	Підпис	Дата				
Розробив		Кучменко К.Ю.		04.04.24	Відомість документів			
Керівник		Праворська Н.І.		04.04.24		Аркуш	Аркушів	
Консульт.								
Н. Контр.		Бедратюк Г.І.		04.04.24	Діаграма вікон інтерфейсу			
Зав. каф.		Бедратюк Л.П.		04.04.24				



					КвРІПЗ. 200166.01.13.ПЗ	
					Ігровий застосунок у жанрі "Платформер" з інтерфейсом управління на основі голосової взаємодії з використанням технологій Unity	Літера / Маса / Масштаб
Зм.	Арк.	№ докум.	Підпис	Дата		
Розробив		Кучменко К.Ю.		04.04.24	Відомість документів	Аркуш / Аркушів
Керівник		Праворська Н.І.		04.04.24		
Консульт.					ER діаграма класів	
Н. Контр.		Бедратюк Г.І.		04.04.24		
Зав. каф.		Бедратюк Л.П.		04.04.24		



					КвРІПЗ. 200166.01.13.ПЗ	
					Ігровий застосунок у жанрі "Платформер" з інтерфейсом управління на основі голосової взаємодії з використанням технологій Unity Відомість документів	Літера: — Маса: — Масштаб: —
Зм.	Арк.	№ докум.	Підпис	Дата		Аркуш: — Аркушів: —
Розробив		Кучменко К.Ю.		04.04.24		
Керівник		Праворська Н.І.		04.04.24		
Консульт.					UML діаграма варіантів використання	
Н. Контр.		Бедратюк Г.І.		04.04.24		
Зав. каф.		Бедратюк Л.П.		04.04.24		

СУПРОВІДНІ ДОКУМЕНТИ

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

здобувача вищої освіти
Кучменка К. Ю.
Прізвище, ініціали

факультет ІТ, 4 курс, група ІПЗ-20-1

ЗАЯВА

З правилами чинного Положення «Про систему забезпечення академічної доброчесності в Хмельницькому національному університеті», згідно з яким виявлення академічного 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 наявність академічного плагіату оповіщений та надаю свою згоду на обробку й збереження університетом моєї роботи в інституційному репозитарії Хмельницького національного 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 текстових збігів у роботах.

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

06.06.24
дата

підпис

UNICHECK
by Turnitin

Ім'я користувача:
ІПЗ

ID перевірки:
1016316943

Дата перевірки:
04.06.2024 02:58:02 EEST

Тип перевірки:
Doc vs Internet + Library

Дата звіту:
04.06.2024 09:03:46 EEST

ID користувача:
100012953

Назва документа: **БКР_Ігровий застосунок у жанрі Платформер з інтерфейсом управління на основі голосово...**

Кількість сторінок: **67** Кількість слів: **12582** Кількість символів: **95393** Розмір файлу: **1.20 MB** ID файлу: **1016114513**

4.5%
Схожість

Найбільша схожість: **1.73%** з джерелом з Бібліотеки (ID файлу: **1016114512**)

3.42% Джерела з Інтернету	376	Сторінка 69
2.91% Джерела з Бібліотеки	215	Сторінка 71

1.34% Цитат

Цитати	8	Сторінка 72
Посилання	1	Сторінка 72

0%
Вилучень

Немає вилучених джерел

Anti-Plagiarism v-15.257

Максимальне співпадіння з одним документом 1.0%

Словники перевірки: en_US, ru_RU, ua_UA. **Помилок в документах: 11%**

ID: 128475 Назва: БКР Ігровий застосунок у жанрі "Платформер" з інтерфейсом управління на основі голосової взаємодії з використанням технологій Unity Додано в БД: 2024-06-05 Автора: Кучменко К.Ю. Керівники: Праворська Н.І., канд.пед.наук Консультанти: Опоненти:	Документ		Сумарний збіг по Базі Даних	
	Символи	Лексеми	Символи	Лексеми
	83565	697	2208 (3%)	26 (4%)

Джерело плагіату

ID	Опис	Наявність плагіату в документі	
		Символи	Лексеми

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

РЕЦЕНЗІЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ
освітнього ступеня «Бакалавр»

Дипломник Кучменко Костянтин Юрійович

Тема Ігровий застосунок у жанрі "Платформер" з інтерфейсом управління на основі голосової взаємодії з використанням технологій Unity

Спеціальність 121 – Інженерія програмного забезпечення

Обсяг кваліфікаційної роботи:

Кількість листів креслень 3 ; кількість сторінок записки 55

1.Короткий зміст пояснювальної записки та прийнятих рішень У В рамках кваліфікаційної роботи було проведено дослідження та аналіз предметної області. Були вивчені вже існуючі на ринку рішення, проаналізовані їхні переваги та недоліки. На основі цього було визначено вимоги до розробки програмного забезпечення. Далі, було досліджено різноманітні технології, розроблено архітектуру програмного продукту та прототип інтерфейсу користувача. Цей прототип згодом був використаний при розробці програми.Після завершення розробки було проведено ряд тестувань.За результатами тестувань можна зробити висновок, що програмне забезпечення готове до експлуатації.

2. Висновок про відповідність роботи поставленому завданню Всі поставлені вимоги та завдання, що стосувалися кваліфікаційної роботи, були успішно виконані.

3. Характеристика виконання кожного розділу роботи, ступінь використання останніх досягнень науки і техніки та передових методів роботи У першому розділі проведено грунтовний аналіз жанру "Платформер" та його особливостей. Виявлено, що динаміка та різноманітність цього жанру роблять його ідеальним для інтеграції голосової взаємодії. Також було досліджено ринок існуючих ігор з голосовим керуванням, зокрема в жанрі "Платформер". Аналіз цих ігор дозволив виявити їхні сильні та слабкі сторони, що стало важливим для подальшого проектування.У другому розділі визначено необхідний функціонал гри та розроблено архітектуру програмного забезпечення. Описано основні компоненти та класи, що відповідають за різні аспекти функціоналу гри. Архітектура спроектована з урахуванням принципів модульності та розширюваності.Далі, було проведено проєктування інтерфейсу користувача, що включало створення різноманітних меню для зручного доступу до функцій гри. Також було описано інструменти, що використовувалися для розробки гри, від ігрового рушія Unity до графічних програм для створення піксельної графіки та анімацій.Завершальним етапом було тестування гри, що включало функціональне тестування, ігрове тестування та збір зворотного зв'язку. Це дозволило виявити та виправити можливі недоліки перед випуском продукту.В результаті виконаної роботи було розроблено програмний продукт - гру з голосовим керуванням в жанрі "Платформер". Продукт відповідає всім поставленим вимогам та завданням.

4. Позитивні сторони роботи Тема даної кваліфікаційної роботи є актуальною, що було чітко обгрунтовано в роботі.Під час аналізу існуючих на ринку програмного

забезпечення рішень було використано статистичні дані і з різних джерел інформації. При виборі засобів розробки було проведено грунтовний аналіз доступних технологій. Були описані їхні переваги та недоліки, а також принципи роботи.

5. Негативні сторони роботи Мала кількість механік у грі, що може спричинити бистре привикання до гри і поступове набридання. Щоб зробити гру більш привабливою та цікавою, рекомендується додати більше механік, заснованих на голосовому управлінні.

6. Оцінка графічного оформлення та пояснювальної записки Візуальне оформлення кваліфікаційної роботи, яке включає різноманітні рисунки які відображають те як виглядає гра та діаграми, відповідає темі дослідження.Пояснювальна записка до роботи складена відповідно до всіх вимог та чинних стандартів.

7. Відгук про кваліфікаційну роботу в цілому Кваліфікаційна робота заслуговує позитивну оцінку.Матеріал пояснювальної записки чітко структурований, послідовний та детально описаний. Що робить матеріал в записці читабельним. Також наведені графічні матеріали наочно демонструють всі аспекти виконаної роботи.

8. Інші зауваження

9. Оцінка кваліфікаційної роботи Кваліфікаційна робота виконана в повному обсязі, з дотриманням усіх поставлених вимог та завдань. Тому вона заслуговує на оцінку
Задовільно

РЕЦЕНЗЕНТ Мартинюк Валерій Володимирович, кандидат технічних наук, доцент кафедри комп'ютерної інженерії та інформаційних систем (ККІС) ХНУ.

"6" 06 2024 р. (підпис)

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

Підтверджуємо ознайомлення з результатом звіту подібності щодо роботи, генерованого системою виявлення текстових збігів/ідентичності/схожості:

Назва: «Ігровий застосунок у жанрі "Платформер" з інтерфейсом управління на основі голосової взаємодії з використанням технологій Unity»

Автор: Кучменко Костянтин Юрійович
Спеціальність: 121 – Інженерія програмного забезпечення
Освітня програма: Освітньо-професійна програма «Інженерія програмного забезпечення»
Науковий керівник: Праворська Наталія Іванівна, кандидат технічних 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 посиланання 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 4,5% і адресується до 591 джерела, що, з урахуванням наведених обгрунтувань, відповідає характеру теми і свідчить на користь кваліфікаційної роботи.

Дата 5.06.2024

Завідувач кафедри — Леонід БЕДРАТЮК

Гарант освітньої програми — Леонід БЕДРАТЮК

Керівник кваліфікаційної роботи — Наталія ПРАВОРСЬКА