Вищий навчальний заклад

«Університет економіки та права «КРОК»
Фаховий коледж

Циклова комісія з інформаційних технологій

Кваліфікаційна робота фахового молодшого бакалавра

на тему: “Розробка 2D гри на платформі Unity. Жанр: аркада. Назва «Бджоліки вперед»”

Виконала _____
(Підпис)
Літошенко Софія Олександрівна
(прізвище, ім’я, по батькові)

_____

Науковий керівник
Уваров Леонід Миколайович
(прізвище, ім’я, по батькові)

_____

_____
(Резолюція «До захисту»)

Попередній захист:

_____
(Висновок: “До захисту в екзаменаційній комісії ”)

Голова циклової комісії _____ _____ _____
(Підпис) (Прізвище, ініціали) (Дата)

Київ – 2025 року

ВИЩИЙ НАВЧАЛЬНИЙ ЗАКЛАД

УНІВЕРСИТЕТ ЕКОНОМІКИ ТА ПРАВА «КРОК»

Фаховий коледж

Циклова комісія з інформаційних технологій

Спеціальність 121 інженерія програмного забезпечення

ЗАТВЕРДЖУЮ:
Голова циклової комісії _____ Леонід УВАРОВ
(підпис)

«____» _____ 2025 року

ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

Здобувач освіти Літошенко Софія Олександрівна

1. Тема роботи Розробка 2D гри на платформі Unity. Жанр: аркада. Назва «Бджоліки вперед»

затверджена наказом по університету від «___» _____ 202__ р. № _____

2. Термін здачі закінченої роботи «30» травня 2025 року
3. Вихідні дані до роботи:
   а) цільова аудиторія – вікова категорія гравців, на яку орієнтована гра (наприклад, підлітки, молодь, дорослі), для розвитку уваги, координації та реакції, жанрові вподобання у цільової аудиторії (аркади, головоломки тощо).
   б) функціональність – гравець керує бджілкою, що летить вперед і має уникати перешкод, реалізувати завдання зібрати якомога більше меду або квіток, не зіштовхнувшись з перешкодою, для підвищення складності реалізувати збільшення швидкість польоту з часом, розробити меню гри, реалізувати паузу, таймер, лічильник балів, рівні, підсилювачі такі як: щит, уповільнення часу, систему балів та рекорд, звукові та візуальні елементи.
   в) технічні вимоги – Unity 2D, Build, основні технології C# для написання скриптів, Sprite-анімація для руху персонажа, Physics2D для колізії, гравітації, Canvas для інтерфейсу; ресурси: спрайти персонажів, фону, перешкод, аудіо звуки збору квітів, зіткнень, музичний супровід, система збереження прогресу JSON-файли.
   г) кращі практики та існуючі розробки в області 2D гри на платформі Unity.

4. Зміст пояснювальної записки
   а) Розділ 1 Теоретична частина. Огляд ігрових рушіїв, огляд інструментів для створення 2D графіки, ·робота з палітрою кольорів, огляд схожих ігор
   б) Розділ 2 Проєктування та розробка. Опис меню, опис рівнів, реалізація лічильника балів, звукові та візуальні ефекти.

в) Розділ 3 Експериментальна частина. Тестування системи (тести, інтеграційні тести, API-тести) та критерії успіху тестування. Оцінка ефективності системи (надійність, продуктивність, масштабованість тощо) та методи (навантажувальне тестування, аналіз логів тощо). Інструкції користувачам.

5. Перелік графічного матеріалу:
Скріншоти існуючих рішень
Скріншоти інтерфейсу продукту
Схеми і таблиці щодо візуалізації аналізу
Блок-схеми алгоритмів
Діаграми щодо проєктування продукту (наприклад, потоків даних, переходів станів, сутність-зв'язок, UML тощо)

Дата видачі завдання «12» лютого 2025 року

Науковий керівник _____ Леонід УВАРОВ
(підпис)

Завдання прийняв до виконання _____ Софія ЛІТОШЕНКО
(підпис)

РЕФАРАТ

Пояснювальна записка: 51 сторінок, 54 рисунків, 1 таблиця, 0 додатків, 12 джерел.

Об’єкт дослідження – використання ігрового рушія Unity.

Мета роботи – розробка аркадної гри з декількома режимами гри.

Методи та засоби розробки – Unity, C#, програмне забезпечення для створення графіки Adobe Illustrator, редактор коду Rider.

У кваліфікаційній роботі розглянуто існуючі методи та засоби розробки 2D ігор. Обґрунтовано вибір технологій, описано архітектуру та алгоритми ігрової системи інтерфейсу користувача, результати тестування.

Сфера застосування – розробка відеоігор, програмне забезпечення для розваг.

Ключові слова: UNITY, C#, RIDER, ІГРОВІ МЕХАНІКИ, СТВОРЕННЯ ІЛЮСТРАЦІЙ. АРКАДНА ГРА, ДЕКІЛЬКА РЕЖИМІВ ГРИ, ОБ’ЄКТНО-ОРІЄНТОВАНЕ ПРОГРАМУВАННЯ.

ABSTRACT

Explanatory note: 51 pages, 54 figures, 1 tables, 0 appendices, 12 sources.

Object of research – the use of the Unity game engine.

Objective of the work – development of an arcade game with multiple game modes.

Methods and tools for development – Unity, C#, graphic design software Adobe Illustrator, code editor Rider.

The qualification note considers the existing methods and tools for developing 2D games.

The choice of technologies is justified, the architecture and algorithms of the game system, user interface, and testing results are described.

Field of application – video game development, software for fun.

Keywords: UNITY, C#, RIDER, GAME MECHANICS, IARCADE GAME, MULTIPLE GAME MODES, OBJECT-ORIENTED PROGRAMMING.

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

ААА-ігор – умовна підмножина відеоігор, створюваних й розповсюджуваних середніми й великими видавництвами, що зазвичай мають більше коштів на розробку й рекламу.

Зміст

ВСТУП

Актуальність задачі. В сучасному світі необхідність у іграх зростає, часами після важких, довгих, сюжетних ігор настає момент, коли хочеться чогось простого і доступного. І саме тому мобільні ігри мають свою популярність.

На прикладі портованих версій ААА-ігор можна побачити, що вони не користуються таким самим попитом. Це пов’язано з особливостями використання мобільних пристроїв. Специфікою їх використання є: швидкий сеанс гри, не бажання мобільного гравця напружуватися, загальна мало-потужність, через це аркадні ігри не втрачають актуальності та продовжують завойовують увагу людей. Вони не вимагають навантаження на мозок і на рухи. Саме тому вони ніколи не зійдуть з рейтингу, а навпаки конкурують з ААА сегментом.

Мета. Розробити 2D аркадну гру використовуючи інструментарій ігрового рушія Unity.

Завдання роботи. Дослідити створення графіки для 2D ігор. Дослідити колористику. Дослідити процес створення сцен, звуку та анімації.

Об’єкт дослідження. Розробка 2D гри за допомогою інструментарію ігрового рушія Unity та створення власної графіки використовуючи Adobe Illustrator.

Предмет Дослідження. Використання 2D графіки у розробці гри.

Методи Дослідження. Unity – найбільш зручний у створенні ігор. Adobe Illustrator – найкращий векторний редактор для створення векторних зображень.

Практичне значення одержаних результатів. Були розроблені ілюстрації, анімації, ігрові механіки(Фабрика об’єктів), логіка ігрового противника.

Структура роботи:

Ключові слова.

2D

Аркадна гра

Unity

Adobe Illustrator

РОЗДІЛ 1.ОГЛЯД ІСНУЮЧИХ РІШЕНЬ

1.1 Огляд ігрових рушиїв

Ігровий рушій – це програма яка дозволяє створювати ігри, вона адаптована для потреб розробників, що значно полегшує процес створення ігор. Особливістю ігрових рушіїв є те, що вони вже мають свої фізичні рушії, системи рендерингу, системи створення та інтеграції скриптів, а також багато інших функцій.

Перед тим як створювати гру, було вирішено обрати саме той рушій який буде зручним у використанні та в якому рівень знань мови програмування буде не менше ніж 50%. Не дивлячись на те, що рушії полегшують процес створення ігор вони все одно потребують знань мов програмування від розробника.

В ході вибору ігрового рушія було розглянуто наступні рушії: Unity, Unreal Engine, Godot.

Unity – найвідоміший рушій. Орієнтований на створення 2D та 3D ігор. Ігровий рушій підтримує такі мови програмування як C# та C++. Підтримує не лише платформу Android, iOS та безліч інших, що збільшує можливості створення продуктів. А саме Windows, Web, macOs, Linux, Playstation4, Playstation5, visionOs.

Також існує безліч безкоштовних інструкцій. Unity має свій Asset Store –це внутрішній магазин, де спільнота розміщує готових персонажів та іншу графіку 2D та 3D форматів, що значно полегшує розробку прототипів.

Одна з переваг ігрового рушія Unity це оптимізація під мобільні пристрої. Це реалізується через зменшення ваги білду, компресію текстур і аудіо, профайлер для мобільних пристроїв та Adaptive resolution(змінення розширення гри прямо в процесі) для збереження продуктивності.

Також важливо зазначити, що Unity має розвинений 2D Toolkit, тобто редактор анімацій для спрайтів і багато чого ще (1).

Unreal Engine – рушій для створення високоякісних та важких проектів, хоч ігровий рушій працює на мові C++, але все одно він використовує дуже багато

ресурсів. Також Unreal Engine підтримує фізику та освітлення на високому рівні. Через потужну систему візуального скриптингу він не підходить для 2D ігор, лише для 3D, VR/AR (2).

Godot – рушій, який орієнтований на 2D ігри. Його особливість полягає в тому, що він з відкритим кодом та одна з його концепцій легкість, як в інтерфейсі так і в розробці, і в плані використання ресурсів. Основними перевагами є:

Відмінна підтримка 2D – Вбудований 2D рендер, підтримка освітлення та анімованих спрайтів

Мова GDScript. Схожа на Python.

Зміна коду і миттєвий результат без повторного запуску.

Після аналізу існуючих рушіїв, можна зробити такий висновок. Unreal Engine орієнтований на 3D кінематографічну графіку. Рушій має ряд особливостей таких як: продвинуті технології освітлення, фізики, рендерингу, оптимізації цих технологій. Але попри такий великий спектор технологій для створення мобільної 2D гри, цей рушій буде надлишково потужним. На противагу потужному, комерційному Unreal Engine існує легкий та з відкритою ліцензією ігровий рушій Godot. Його перевагою є потужна підтримка 2D, хоч і підтримує 3D розробку, а також система рендерингу 2D об’єктів, саме як 2D, що покращує загальну оптимізацію. Godot дійсно є чудовою платформою для створення 2D ігор, але для створення мобільних 2D ігор він не підходить, через відсутність крос-платформи. Unity ігровий рушій, що є еталоном балансу між створенням 2D та 3D ігор з крос-платформою. Він вбирає в себе як і потужні інструменти розробника аналогічні Unreal Engine так і досить широку підтримку 2D розробки, що робить його найкращим вибором для даного проекту (3).

1.2 Огляд інструментів для створення 2D графіки

Векторна графіка – це метод представлення зображень та об’єктів які можна описати математичними виразами. Такі зображення складаються з точок, ліній, кривих та багатокутників. Векторна графіка дає можливість масштабувати зображення будь-якого розміру без втрати якості. Також векторні файли займають

менше місця і кожен елемент можна окремо відредагувати це дозволяє швидко змінювати ілюстрації (4).

Растрова графіка – це зображення, що складаються з пікселів, які певним чином розташовані у прямокутній сітці. Растрові файли втрачають якість при збільшенні також вони займають більше місця через велику кількість пікселів (5).

Для проекту було обрано векторну графіку, тому що саме ця графіка дає можливість швидко редагувати зображення та змінювати розмір в залежності від потреби.

1.2.1 Огляд Adobe Illustrator

Adobe Illustrator – це професійний векторний графічний редактор, який використовується для створення ілюстрацій, логотипів, інтерфейсів та інших видів чіткої, масштабованої графіки. Для UI елементів та об’єктів, які будуть використовуватися на різних екранах дуже важливо, щоб зображення не втрачало якості при змінні розміру. Adobe Illustrator має складну, але водночас потужну систему інструментів для малювання, роботи з текстом, кольорами (6).

1.2.2 Огляд SketchBook

SketchBook – це растовий редактор, який орієнтований на цифровий живопис. Дозволяє швидко створювати малюнки завдяки простому інтерфейсу і природній поведінці пензлів та різних фарб. SketchBook можна використовувати на різних пристроях як: комп’ютер, телефон, планшет (7).

1.2.2 Огляд Krita

Krita – це потужний інструмент з відкритим кодом, який підтримує не лише малювання, а й роботу з анімаціями, текстурами. Цей редактор має можливості для створення покадрової анімації, що робить її дуже зручною для ігор та мультимедійних проектів(8).

1.2.3 Висновок до підпункту

Проаналізувавши сучасні редактори графіки, можна зробити висновок, що для створення графіки цього проекту краще всього підходить Adobe Illustrator, а саме: стабільність роботи, простота експорту та зручність у використанні.

1.3 Кольори

У відео ігровій сфері не менш важливою складовою ігрового досвіду є візуальна частина, а саме:

- Підбір кольорів
- Поєднання кольорів
- Емоції, що вони викликають
- Стилістика та поєднання всього вище переліченого.

1.3.1 Поєднання Кольорів

Підбір кольорів також є необхідним елементом для створення проектів, а саме знання про їх поєднання, про відтінки які не вимагають великого навантаження на сприйняття. Саме тому варто аналізувати, змінювати та поєднувати кольори, щоб дійти до найкращого поєднання. Для цього існує безліч методів поєднань таких як:

Компліментарне поєднання. Яке в свою чергу поєднує кольори, які розташовані на протилежних сторонах колірного круга Іттена. Таке поєднання виглядає жваво та енергійно. Нижче наведено приклад використання.

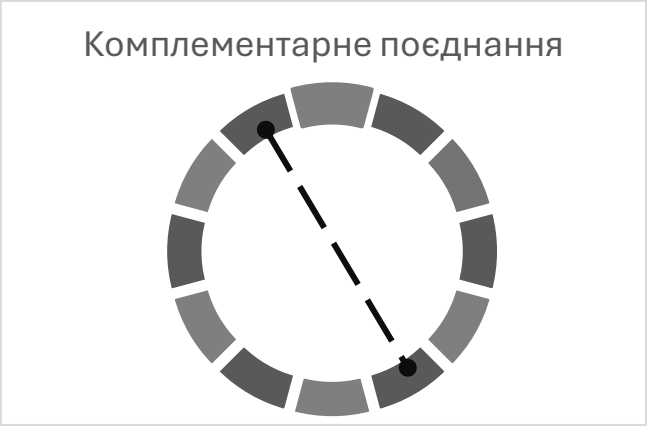

Рисунок 1.1 "Компліментарне поєднання"

Тріада – поєднання трьох кольорів. Основною метою цього методу є підбір кольорів, що лежать на однаковій відстані одне від одного. Поєднання забезпечує контрастність при збережені гармонії. Це поєднання виглядає як трикутник.

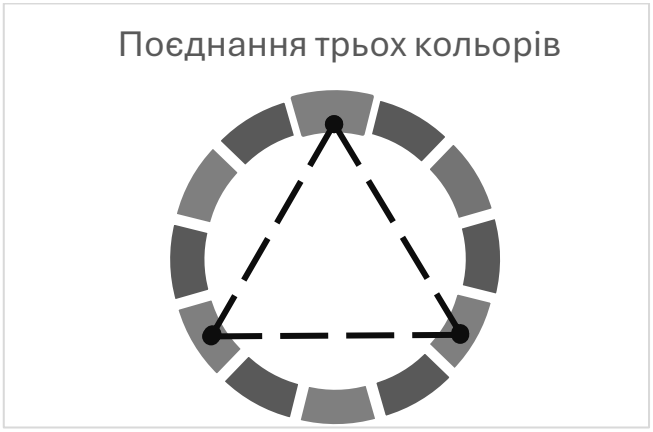

Рисунок 1.2 "Поєднання трьох кольорів"

Аналогічне поєднання –це підбір кольорів які розташовані на одному й тому самому колірному колі. Таке поєднання найчастіше використовується у моїх проектах і не тільки. Бо саме такий підбір кольорів буде нейтральним, приємним і без зайвих витрачених годин на підбір тих чи інших кольорів. Аналогічне поєднання має такий вигляд як зображено на рисунку 1.3

Рисунок 1.3 "Аналогічне поєднання"

Тетрада – поєднання чотирьох кольорів. Такий метод працює таким чином, спочатку обирають основний колір та два кольори, які доповнюють основний та останній який є акцентним.

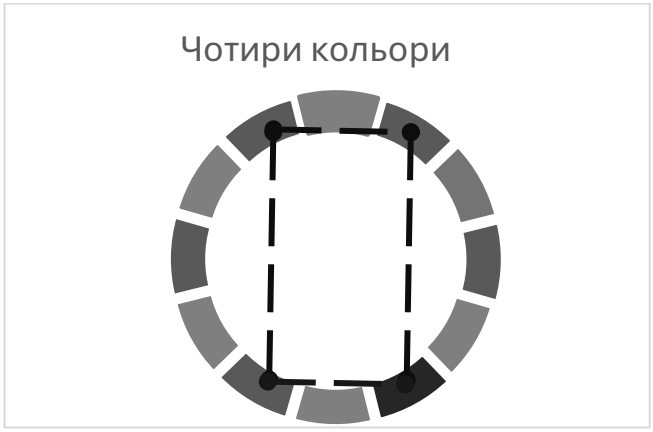

Рисунок 1.3 "Поєднання чотирьох кольорів"

Квадрат – це поєднання кольорів, які рівновіддалені один від одного. Метод цього поєднання має свою унікальність, як і кожен з попередніх методів (9).

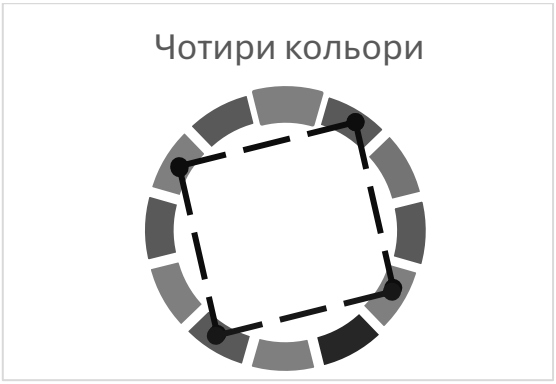

Рисунок 1.4 "Квадрат"

1.3.2 Кольори та їх емоційне значення

Колір може викликати різні емоції. Найчастіше перед тим як обрати кольори для гри, треба зрозуміти який сенс та почуття ця гра має донести до гравця. Чорні та темні кольори відображають старі часи, або похмуру та важку історію. Світлі та яскраві дають можливість відчувати радість та сповнення енергією. Нейтральні передають спокій, а дуже насичені втому та дратівливість.

Кожен сприймає по різному кольори та їх символіку, але у доведених дослідженнях сприйняття кольорів у іграх викликають такі емоцій як:

- Червоний – виражає сильні емоції, такі як гнів, страх. Також використовується, як інструмент привернення уваги гравця до небезпеки.
- Помаранчевий – віддзеркалює радість, розчарування, свіжість та напругу.

- Зелений – безпеку. Його використовують для позначення союзників та при успішному виконання дій.
- Синій – означає спокій, серйозності, холоду. Колір позитивних взаємодій
- Чорний – негативний колір, найчастіше передає страх, зло.
- Білий – нейтральність, невинність, добро.

Але такі кольори не завжди використовуються з таким сенсом (10).

Для того щоб скористатися попередніми підпунктами як 1.3 та 1.3.1 треба обрати загальну стилістику.

1.3.2 Стилістика

Стилістика – це візуальні та художні рішення. Стилістику обирають залежно від жанру гри. Вона створює атмосферу та настрій і має певний вплив на візуальний вигляд гри. Найпоширенішими є:

Піксель-арт – це стилізація під ретро-графіку. Піксель-арт передає мінімалізм, простоту та ностальгію. Варто зазначити, що на створення такої стилізації може бути витрачено багато часу, а також не правильне поєднання кольорів, загальне перевантаження моделей може призвести до несприйняття та розгубленості.

Ручна анімація. Виглядає як намальована ілюстрація, або мультфільм.

Векторна графіка. Складається з простих форм, чітких контурів та схожа на UI-дизайн. Така графіка сучасна та зрозуміла.

Мінімалізм. Зосереджений на механіці, а не на деталях

1.3.3 Висновок

Для проекту було створено візуальний вигляд за допомогою векторної графіки. Підбір кольорів був обраний таким чином, щоб передати весняну та літню атмосферу, а саме ніжні пастельні кольори, що підкреслюються контрастними акцентами.

Для певних об’єктів були використані аналогічні кольорові схеми, але використовувалися інші кольорові рішення.

Кольори та сама стилістика дає відчуття спокою та бажання зануритися у саму гру.

1.4 Огляд схожих ігор

Ринок мобільних ігор має широкий спектр рішень, починаючи від ігор “Три в ряд” закінчуючи портами AAA-ігор. Розглянемо приклади аркадних ігор таких як: MochiCat, Meow Tower, Slidey, BunnyBuns, KleptoCats. За такими критеріями як музика, вибір кольорів, кількість рівнів та їх цікавість.

1.4.1 Огляд гри MochiCat

MochiCat – це 2D гра. Головною метою є догляд за котиком(годування ласощами, виконання квестів).

Ця гра переважно містить ніжні відтінки кольорів, а саме пастельні. Контрастні кольори не дуже помітні, бо зливаються з іншими, але навіть таке кольорове поєднання не втрачає індивідуальних рис об’єктів.

MochiCat має два режими, крім основного. Перший режим “Jenga”, його складність полягає у тому, що він не одразу доступний лише через певну кількість виконаних завдань можна отримати доступ до цього режиму. Головною метою є викласти вертикальну лінію з котів, щоб вони не падали. Другий режим “Dessert”, цей режим полягає у створенні десертів при створенні яких можна отримати нового основного котика.

Гра має лише одну музику для всіх режимів. Мелодія має динамічний ритм, що викликає певний дискомфорт.

Рисунок 1.5 Перший режим MochiCat

Рисунок 1.6 Другий режим MochiCat

1.4.2 Огляд гри Meow Tower

Meow Tower – 2D гра, метою якої є проходження рівня та будування меблів для кота.

Гра має один режим такий як японський кросворд, який влаштований по принципу “Nonogram” де необхідно заповнити правильні клітинки кольором, а на головній сторінці, купувати меблі для котів.

Кольори яскраві та ніжні, що привертають увагу гравця.

Має дві мелодії. Перша спокійна та приємна. Не викликає дискомфорту, а навпаки дає бажання зануритися повністю в гру. Друга змінюється при кожному вході в режим, тому її оцінити важко.

Рисунок 1.7 Головне меню Meow Tower

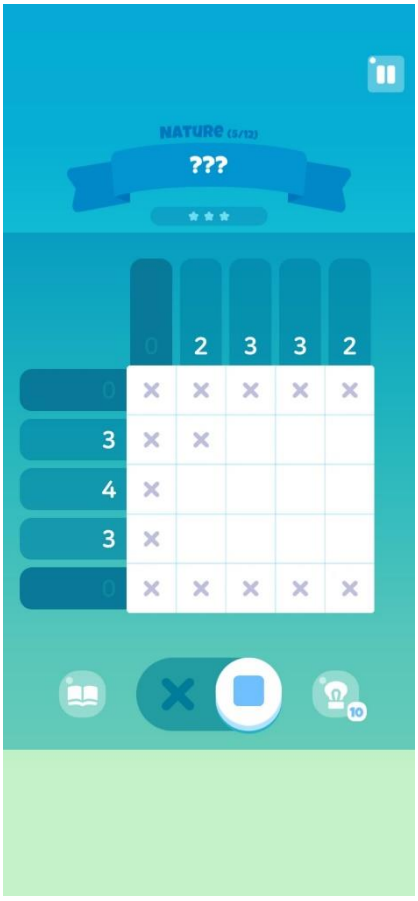

Рисунок 1.8 Перший режим Meow Tower

1.4.3 Огляд гри Slidey

Slidey –це 2D гра. Влаштована по принципу гри “Block”.

Гра має три режими, які відрізняються візуально та мають свою техніку проходження. Перший режим “Event” полягає у тому щоб скласти блоки в певну кількість разів для отримання об’єктів які мають бути на малюнку для колекції. Другий режим – “Classic”. Візуально змінюються блоки які перетворюються на котів, головною метою є скласти блоки так, щоб їх не було переповнено. Третій – “Winter” схожий на попередній режим, але має свої певні умови.

Кольори яскраві та ніжні, часами контрастні.

Музика однакова у всіх режимах, не спокійна, має динамічний ритм. Також не комфортна.

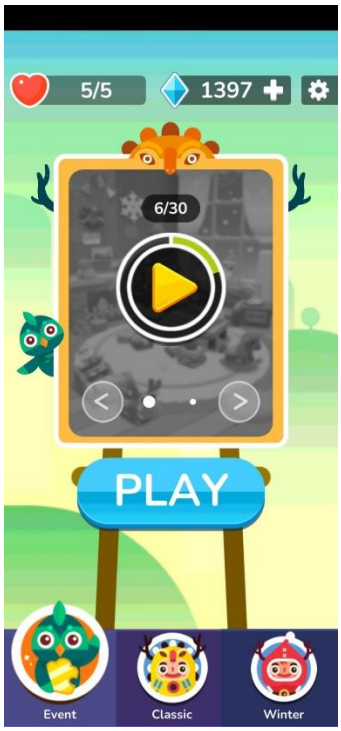

Рисунок 1.9 Режим Event

Рисунок 1.10 Режим Classic

Рисунок 1.11 Режим Winter

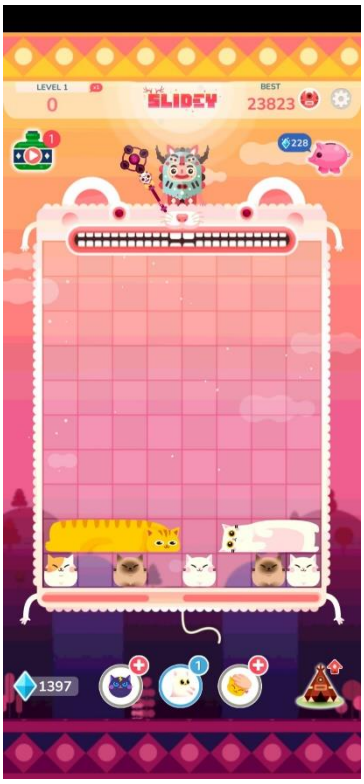

Рисунок 1.12 Вигляд одного з режимів

1.4.4 Огляд гри BunnyBuns

BunnyBuns – 2D гра. Мета геймплею змішувати різні інгредієнти для створення тістечок по замовленню(покупців).

Гра має один режим, де зображено робоче місце головного персонажа та різні інгредієнти. Головним персонажем є кролик, а покупці – різні тваринки та магічні істоти.

Кольори яскраві та теплі, що створюють комфортну атмосферу.

Музика дає відчути себе у італійській кав'ярні. Спокійна насичена.

Рисунок 1.13 Вигляд гри BunnyBuns

1.4.5 Огляд гри KleptoCats

KleptoCats також 2D гра. Головною метою є догляд за котиком, але ця гра відрізняється від MochiCat тим, що гравець відправляє котика на прогулянку і він приносить знайдені речі в колекцію для кімнат.

Ця гра має один режим, де можна вигулювати, годувати, гладити кота та відправляти на прогулянку. Також можна купувати або вигравати нового котика для головного режиму.

Кольори яскраві, різні та контрастні.

Музика має динамічний ритм, також не комфортна.

Рисунок 2.14 Вигляд гри KleptoCats

1.4.6 Порівняння аркадних ігор

Таблиця 1.1

Перелік	**Назви ігор**				
	MochiCat	**Meow Tower**	**Slidey**	**BunnyBuns**	**KleptoCats**
Музика	7/10	8/10	5/10	10/10	4/10
Кількість режимів	2 режими	1 режим	3 режими	1 режим	1 режим
Оцінка кольорової схеми	10/10	10/10	9/10	10/10	7/10
Якість геймплею	10/10	9/10	8/10	10/10	7/10

1.4.7 Висновок

Аркадні ігри бувають різними. Після аналізу вище перелічених ігор можна зробити висновок, що кожна гра має свою стилістику, кольори, мелодії, режими. Але не дивлячись на це варто приділяти більше часу на розробку візуального вигляду, підбирати кольори які будуть приємними для очей та обирати ту мелодію

та звуки, які не будуть дратувати. Тож у підсумку можна сказати, що аркадної гри більш важливу роль відіграє візуальна частина ніж ігрові механікі.

Спираючись на це, під час виконання проекту головною задачею було створення приємного візуального оформлення.

1.5 Висновок розділу

Проаналізувавши ігрові рушії, інструменти для створення 2D графіки, кольори, поєднання кольорів, кольори та їх емоційне значення, стилістику, для створення 2D гри. Були визначені наступні цілі:

Розробка механіки на ігровому рушії Unity

Створення кольорової схеми гри

Створення графіки для гри

Тестування ігрових механік

РОЗДІЛ 2. ПРОЕКТУВАННЯ ТА РОЗРОБКА

2.1 Опис основного меню

В даному пункті буде описано структуру гри та візуальні елементи: основного меню, перший, другий та третій рівні.

При першому запуску гри, гравець бачить основне меню на якому розташована назва по центру “Play”, що закликає до гри, але взаємодія з нею не можлива.

Рисунок 3.1 “Play”

Головне меню створене таким чином, щоб гравець звертав увагу на три основні режими(рівні) гри. Вони розташовані під назвою “Play” мають різний візуальний вигляд, що дає можливість зацікавити користувача у тому чи іншому виборі режиму. Три режими мають однаковий колір фону та мають тип фігури – коло, але кожен з них відрізняється візуально, та підказує гравцю, про що буде той чи інший рівень. На першому режимі зображено бджоліка біля якого розташовано по колу з однієї сторони 3 квітки. Кожна квітка має свою довжину пелюсток та стеблову частину.

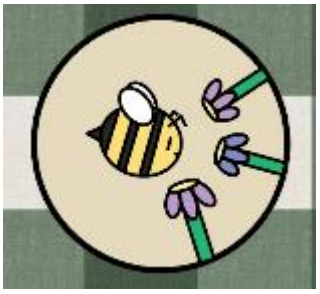

Рисунок 2.2 Перший рівень

Другий режим має бджілку яка несе відро, щоб заповнити медом соти.

Рисунок 2.3 Другий режим

На третьому зображено також три квітки, які не мають стеблову частину, поміж яких знаходиться наш головний персонаж бджолік.

Рисунок 2.4 Третій рівень

При натисканні на будь-яку з трьох іконок режиму гравець переходить до гри. Під доступними режимами розташовано також два не доступні режими які мають водночас контрастний, але пригнічений колір. Що дає гравцю можливість сконцентрувати увагу на доступних режимах, а потім роздивитися все інше.

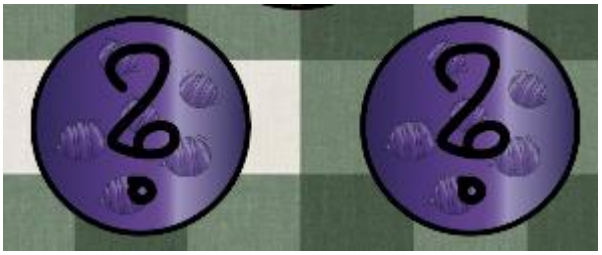

Рисунок 2.5 Не доступні режими(загадкові)

Варто зазначити, що основне меню має не лише різні режими та назву, а також необхідні налаштування, як регулювання гучності мелодій для всіх режимів. Кнопка налаштування має вигляд шестерні при взаємодії з якою можна налаштовувати гучність в середині гри.

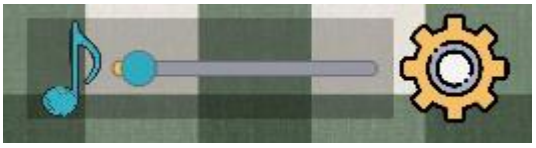

Рисунок 2.6 Вигляд налаштування

2.2 Опис першого рівня

Перший рівень має два вигляди як і інші режими. Фоновим зображенням є хмаринки пастельних кольорів. Також присутній головний персонаж бджолік, який має анімацію руху(розмахування крилами).

Рисунок 2.7 Вигляд головного персонажа

Панель підрахунку підібраних квіток має наступний вигляд:

Рисунок 2.8 Панель підбору квіток

Під час гри гравець може підбирати квітки, які будуть зараховані у панель підрахунку. Вони мають анімацію коливання з однієї сторони в іншу.

Рисунок 2.9 Квітка

З вище наведеної інформації можна зробити висновок, що рівень гри сам по собі є дуже простим, але складність цього режиму полягає у тому, що головний персонаж має пролітати поміж квіток з різною висотою та довжиною стеблової частини та пелюсток. Таким чином гра набуває певної складності.

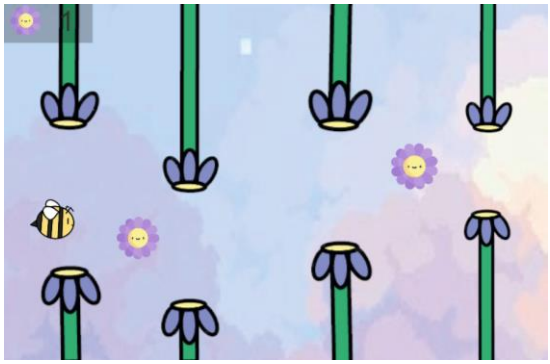

Рисунок 2.10 Вигляд першого рівня

Коли гравець програє, фон першого режиму стає темно-прозорим. На фоні зображуються інформація як “GameOver” та максимальне число квіток, які гравець зміг підібрати.

Рисунок 2.11

Також зображено кнопки як “Повторити спочатку” та “Повернутися до головного меню”.

Рисунок 2.12

2.3 Опис другого рівня

На початку гравець звертає увагу на фон тому що він насичений і має контрастні кольори на відміну від попереднього рівня. Цей рівень має такі об’єкти:

- Бджілки, що тримають відро з медом, літають та виливають його.

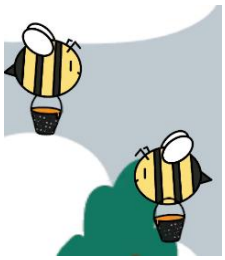

Рисунок 2.13 Бджілки

- Банка з медом, в яку треба збирати мед.

Рисунок 2.14 Банка з медом

Каплі меду та шкідливі каплі(насичено фіолетові), після збору яких гравець програє і розбиває банку.

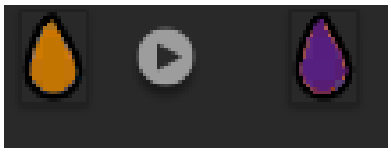

Рисунок 2.15 Каплі меду та шкідливі каплі

При поразці гравець бачить такий самий темно-прозорий фон. На якому зображена інформація: “GameOver” та максимальна кількість підібраних краплин.

Рисунок 2.16

Також на фоні зображено кнопки: “Повторити спочатку” та “Повернутися до головного меню”. Рисунок 2.12

Вигляд самого рівня:

Рисунок 2.17 Вигляд другого рівня

2.4 Опис третього рівня

Фон третього рівня має пастельні та ніжні кольори. Зверху присутня панель з квітками та їх кількістю, яку варто зібрати.

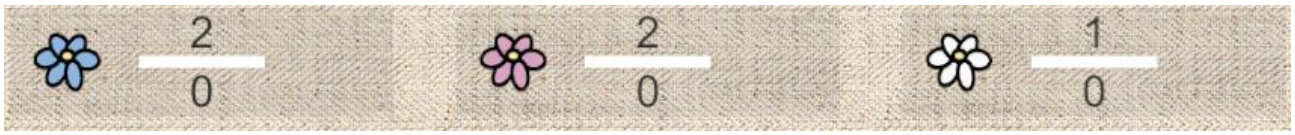

Рисунок 4.18

У цьому режимі присутній головний персонаж та злий бджолік. Він більший за розміром та має акцентний помаранчевий колір. Його головною метою є відбирання квіточок, які з’являються на фоні.

Рисунок 2.18 Злий бджолік

Також біля панельки з квітками є підрахунок рівнів(тобто якщо всі квітки зібрані ми переходимо на новий рівень)

Рисунок 2.19

Самі квіти мають три основні кольори як: ніжно-синій, рожевий та білий. Коли гравець програв з’являється темно-прозорий фон, на якому присутні назви: “GameOver” та рекорд по кількості рівнів.

Рисунок 2.20

Також такі самі кнопки: “Повторити спочатку” та “Повернутися до головного меню”. Рисунок 2.12

2.5 Опис коду головного меню

Unity дозволяє створювати окремі частини/режими для гри на різних сценах та додавати їх, що значно полегшує можливість їх редагування. Для головного меню було створено Scene назва якої “Main”. Частини коду присутні на таких об’єктах: шестерні, трьох режимах. Отже для того, щоб переходити з однієї сцени на іншу треба написати коротенький код на Unity Script, саме його використовують коли код треба під’єднати для об’єкта який розташований на сцені.

Отже необхідно створити поля з якими ми будемо взаємодіяти. Та написати такий код:

```
public void Open()
{
    SceneManager.LoadScene(sceneName, LoadSceneMode.Single);
}
```

Рисунок 2.21

Тут використовується публічне рядкове поле яке було створене раніше для цієї функції та використовуємо LoadSceneMode.Single, щоб відкрити одну сцену.

Для всіх режимів у самому рушії Unity додається цей код як компонент. Він має таке поле як Scene Name і в нього вводиться назва сцени, яка повинна вмикатися.

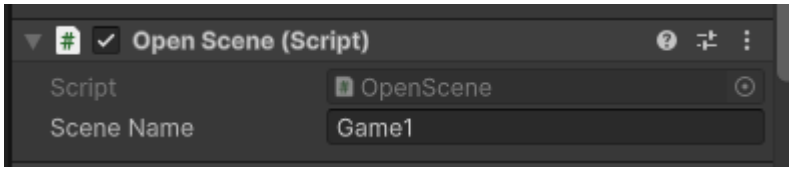

Рисунок 2.22

Та компонент “Button”, щоб гравець міг взаємодіяти з режимами. Також необхідно додати в функцію “On Click()” сам об’єкт перетягнути у доступне поле, а з правої частини обрати OpenScene.Open().

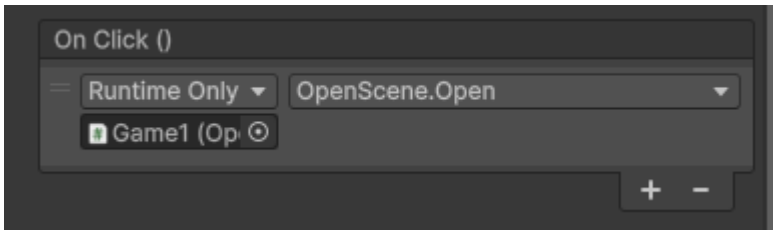

Рисунок 2.23

Цей код можна використовувати для будь-якої сцени та об’єктів. Тому інші режими також мають такі компоненти та код.

Шестерня має компонент “Button” та “Image” в якому розташована картинка шестерні.

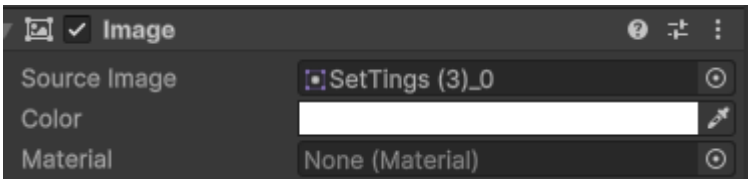

Рисунок 2.24 Компонент Image

Для того, щоб плавно рухалася панель вниз та вверх з регулюванням гучності. Було вирішено створити багато об’єктів у сцені “Main”.

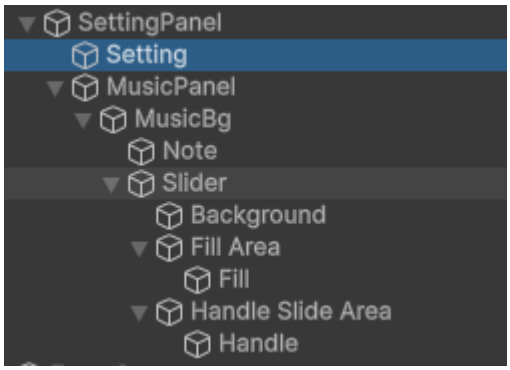

Рисунок 2.25 Об’єкти сцени Main

“SettingPanel” – це порожній об’єкт, який містить компонент “Animator” та компонент коду.

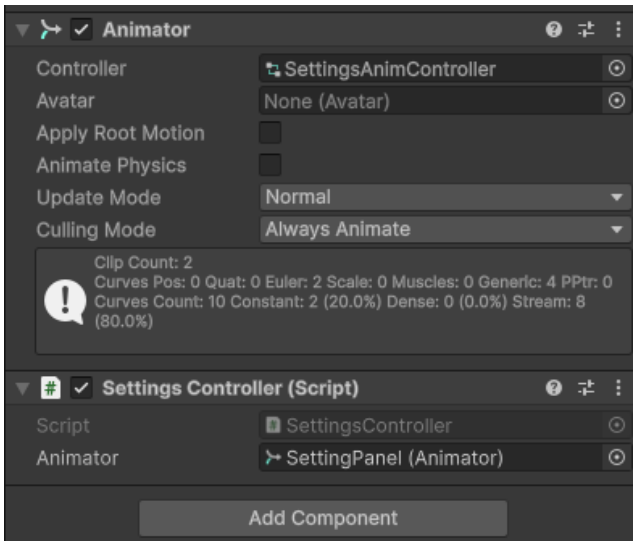

Рисунок 2.26

"Setting" – це зображення самої шестерні, має такі компоненти як "Button" та функцію "On Click()" де використовується об'єкт "SettingsPanel" та з правої частини обрану функцію, яка була створена кодом.

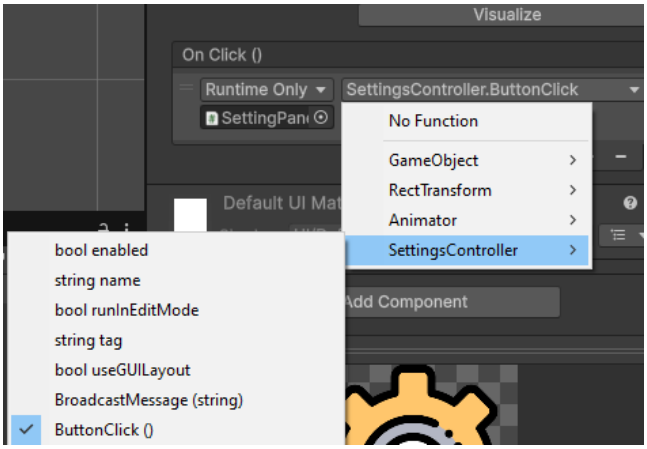

Рисунок 2.27 Функція OnClick()

"MusicPanel" містить компонент "Scripts", а саме код для взаємодії зі звуком. Код має два публічні поля, щоб використати їх у сцені. Виглядає це наступним чином.

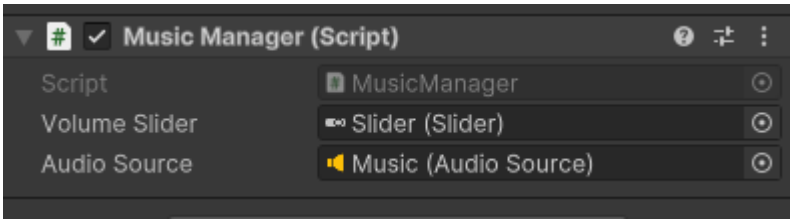

Рисунок 2.28

Код розроблений так, щоб користувач міг взаємодіяти з шестернею та керувати гучністю мелодії. Особливістю є те, що після того, як гравець виставив

гучність та вийшов з гри, при поверненні до гри гучність залишиться такою самою як і останнього разу. Для того щоб все зберіглося використовують PlayerPrefs(сховище даних на пристрої). У об'єкт "Slider" передано в "On Value Changed" самий об'єкт "MusicPanel" де використовується функція, яку було створено у UnityScripts:

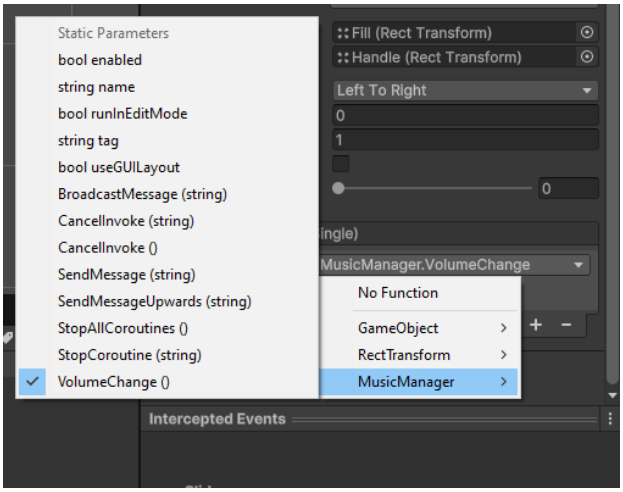

Рисунок 2.29

2.6 Опис коду першого рівня

Бджолік має Animator, Rigidbody 2D, що відповідає за фізику об'єкта, Capsule Collider 2D – це невидимий об'єкт яким можна позначити межі об'єкта. Також містить Script "BeeController" у якого в полях додаються такі об'єкти:

- Rigidbody2D в нього закидаємо наш "Prefab" з бджоли, бо саме на ньому є такий компонент.
- Jump Height = 4 –Задана висота стрибка.
- Coin Manager –елемент для підрахунку підібраних квіток.
- Animator компонент, що передаємо як властивість головного персонажа, для того щоб створити його анімацію.
- Game Over Controller – механізм завершення гри.

Квіти мають Box Collide2D, Rigidbody2D та Flower Controller(Script) у якому можна змінювати швидкість руху квіток з правої частини в іншу. Код містить поле зі значенням швидкості за замовчуванням один.

Update() – це спеціальний метод, що викликається кожен кадр.

GameObject – це об'єкт, до якого прикріплений код.

Transform – компонент, який відповідає за позицію, масштаб об’єкта.

Translate – метод, який переміщує об’єкт на вказаний вектор.

New Vector – створює вектор руху по координатах: x, y, z.

“-flowerSpeed” – означає рух вліво по осі x.

Time.deltaTime – це час між кадрами, який використовують для того, щоб рух був плавним. Нульовими значенням є “0f,0f”, вони відповідають за y та z.

Flower Manager Script – цей код керує створенням нових квіток, коли гравець рухається вперед.

Public GameObject[] flowerPrefabs – масив префабів квіток.

Public List<GameObject> flowers = new() це список створених квіток, які можна додавати та видаляти. Тобто у самій сцені це виглядає наступним чином:

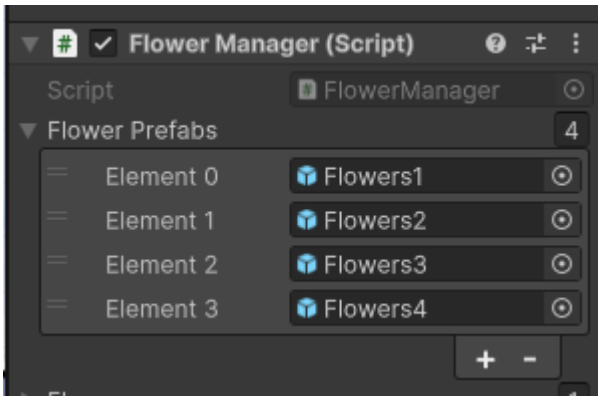

Рисунок 2.30

Далі змінна для зберігання швидкості руху квіток, та public BeeController beeController – це використання beeControllera з класу BeeController.

```
public void CreateNewFlowers(GameObject lastflower)
{
    if (beeController.gameOver) return;
    var index = Random.Range(0, flowerPrefabs.Length);
    var randflower :GameObject = flowerPrefabs[index];
    var flowerRenderer = lastflower.GetComponent<SpriteRenderer>();
    var width :float = flowerRenderer.bounds.size.x;
    var pos :Vector3 = lastflower.transform.position + new Vector3(width * 1.4f, 0, 0);
    Instantiate(randflower, pos, Quaternion.identity);
}
```

Рисунок 2.31 Функція створення нових квіток

Ця функція створює нову квітку праворуч Рисунок 2.31. Коли гравець програє то гра завершується. Тут вибирається випадковий індекс з масиву

префабів квіток Var index = Random.Range(0, flowerPrefabs.Length), у var randflower = flowerPrefabs[index] ми отримуємо випадковий префаб квітки, використовуючи випадковий індекс. Ця частина коду “var flowerRenderer = lastflower.GetComponent<SpriteRenderer>();” відповідає за те, щоб дізнатися ширину квітки, щоб розмістити нову. В перед останньому рядку береться позиція останньої квітки, яка додається вправо за координатами x та множиться на 1.4f. Та останнє це Instantiate(randflower, pos, Quaternion.identity) створює нову квітку на позиції, на позиції pos без обертання.

FlowerCheck відповідає за взаємодію з іншими об’єктами, які мають ключ “NabirFlowers”. Він використовує тригери, щоб створювати та знищувати квітки. За сценою з різних боків розташовано порожній об’єкт який має такі властивості: Box Collider2D та сам код.

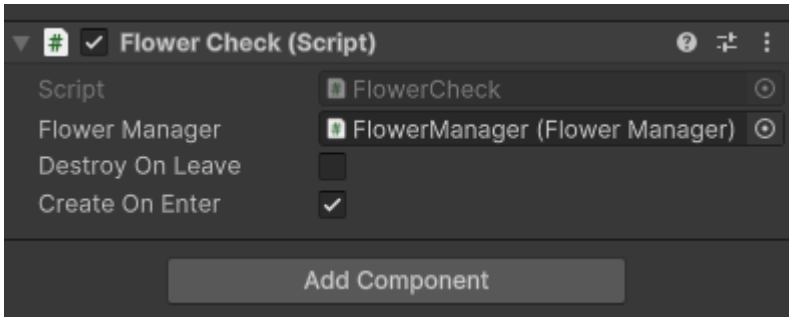

Рисунок 2.31

Для додавання музики, треба було створити порожній об’єкт та додати компонент Audio Source та додати Volume Controller script

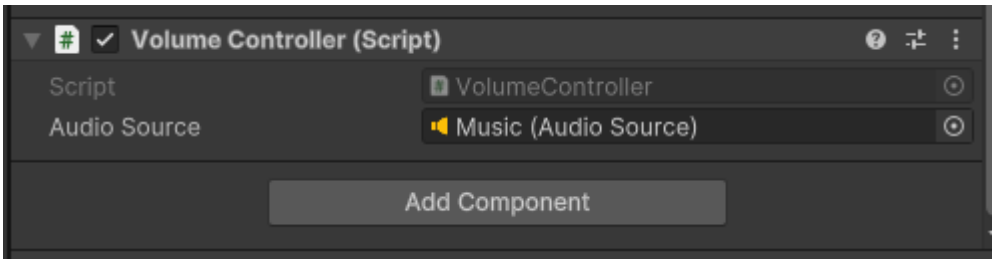

Рисунок 2.32

Для панелі підбору квіток був створений код Coin manager, у якому якщо була підібрана квітка то збільшується на 1 пункт.

Коли гравець програє на екрані відображаються зібрані квітки, максимальне число та викликається об’єкт GameOver який відображає вище сказане. Та доданий Game Over Controller script. GameOver.SetActive(true) робить об’єкт видимим та активним, Потім отримується кількість зібраних монет та зчитується яку максимальну кількість монет гравець коли-небудь назбирав. Якщо було зібрано більше то оновлюється рекорд.

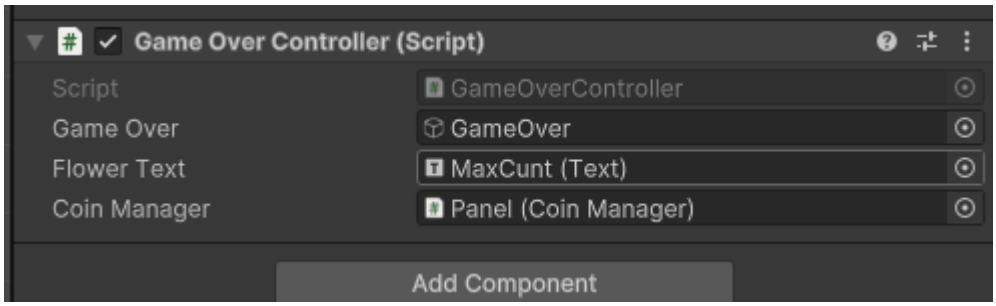

Рисунок 2.33 Game Over Controller Script

2.7 Опис коду другого рівня

На цьому рівні наявні такі об’єкти та їх код:

- Trash – це порожній об’єкт який розташований знизу за екраном. В ньому прописаний код який очищує різні каплини, коли вони потрапляють в об’єкт. Сам код досить простий.

```
private void OnTriggerEnter2D(Collider2D other)
{
    if (other.CompareTag("Kapelka")) Destroy(other.gameObject);
}
```

Рисунок 2.34

OnTriggerEnter2D(Collider2D other) – це вбудований метод, який викликається коли інший об’єкт взаємодіє з колайдером. Other – це той об’єкт який потрапив до Trash. Також створено перевірку, що якщо об’єкт має ключ “Капелька” і він потрапив до Trash, то метод Destroy(other.gameObject) знищує його. Може скластися враження, що немає сенсу від цього механізму, але насправді він є, бо через те що зайві об’єкти залишаються, то гра може почати зависати на мало потужних пристроях.

- SpawnBee – це порожні об’єкти, які розташовані з двох боків та відповідають за створення бджоліків, які рухаються і з ліва, і з права. Вони мають Box Collider 2D, код BeeSpawner в якому є різні поля. Bee Prefab, що дає можливість обрати перший чи другий SpawnBee, speed та id де можна ввести своє значення. Функція void Start() виконується один раз на початку. Також вона має StartCoroutine(SpawnBee()), яка постійно створює нових бджілок. У функції OnTriggerEnter2D(Collider2D other) яка має перевірку, що якщо інший об’єкт має ключ “Bee” то перевіряється його id через BeeMoverController, якщо id не збігається то вона знищується. Наступна функція IEnumerator SpawnBee(), яка має перевірку, що допоки гра не завершена, створюються бджоли на випадковій висоті, яка раніше була зазначена в певних межах. Якщо швидкість менше нуля то бджола змінює свій стан на 180 градусів, щоб здавалося що вона летить в ту чи іншу сторону. Потім отримуючи з BeeMoveController в якому викликається StartSpawn(time) та задається id та speed. Після чого yield return new WaitForSeconds(5) робить затримку для створення наступної бджоли.
- Jar – це об’єкт, який має вигляд банки з медом. В яку треба збирати мед, який виливають бджоли. Але вони час від часу виливають або краплі меду, або шкідливі краплі. При контактуванні банки з отруйною каплею вмикається анімація тріснутої банки та завершення гри, а при зборі звичайної в панель підрахунку капель збільшується рахунок. Jar має такі компоненти, як Sprite Renderer де обирається вигляд. Animator та Box Collider 2D. Та код Jar Controller, який має метод GetCorrectPosition(float pos) він обмежує горизонтальний рух гравця в межах екрана. Повертає Mathf.Clamp(pos,minx,maxx), що гарантує, що по координаті x jar не вийде за межі екрану, також враховано розмір об’єкта, щоб він не виходив за край. Метод Update() отримує позицію курсора(миші) або пальця, та перетворює їх у координати гри. Переміщує гравця по осі x не змінюючи інші. Метод Start() приховує курсор на екрані. OnCollisionEnter2D(Collision2D other)

метод який має різні перевірки. Якщо гра завершена то нічого не повертається, якщо відбулося зіткнення з об'єктом яке має ключ "Kapelka" то викликається метод Catch() та збільшується число зібраних капель. Якщо ж відбулося зіткнення з об'єктом "BadK" то запускається анімація тріснутої банки та завершується гра, знищується "BadK" та зупиняється створення нових бджіл.

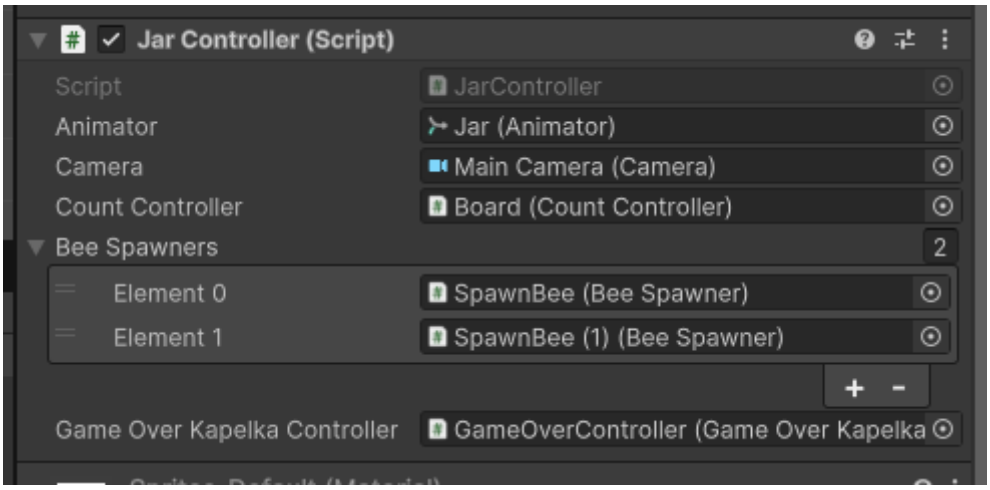

Рисунок 2.35 Jar Controller

- Board – це панель підрахунку краплин меду. За що відповідає CountController у якому є функція CatchKapelka() де count збільшується на один, результат оновлюється на панелі, та запускається анімація капельки на панелі. Метод GetCount() дає можливість отримати скільки капель було спіймано.
- GameOverController має доступ до GameOverKrapelkaController в якому є функція GameOver(), що відповідає за завершення гри, коли гравець спіймав шкідливу краплину. Public GameObject game over це посилання на панель, яка відобразиться після завершення гри. Public Text kapelkaText – це текстове поле у якому буде показана кількість краплин. Public CountController countController з нього отримуємо об'єкт класу, який підраховує скільки крапель спіймано. "gameOver.SetActive(true)" вмикає екран завершення гри. Var coin = countController.GetCount() з відси отримуємо кількість спійманих крапель. Var maxKapelka = PlayerPrefs.GetInt("MaxKapelka",0) тобто

максимальна кількість крапель зберігається в локальному сховищі під ключем “MaxKapelka”, якщо збереження немає то повертає нуль. Код містить перевірку, чи зібрана кількість монет більша за максимальну кількість минулого разу. Якщо новий рекорд то відбувається зберігання у PlayerPrefs та вивід найкращого результату.

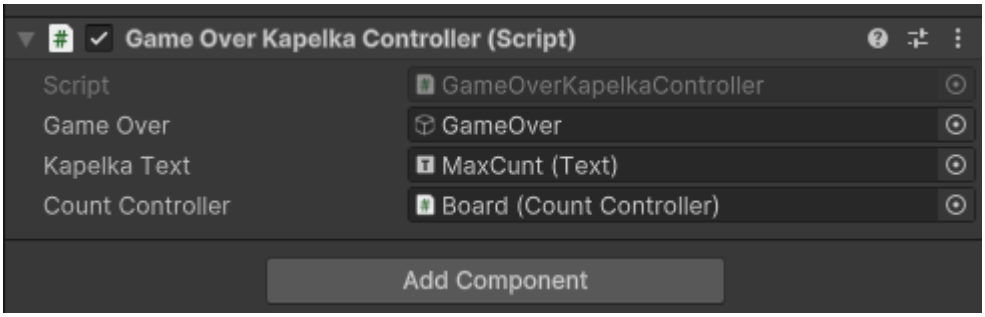

Рисунок 2.36 GameOverCappelkaController Script

- Музика додана так само як і в попередньому режимі.

2.7 Опис коду третього рівня

Третій режим має такі об’єкти та контролери:

- KwitkaPanel створена таким чином, щоб за потреби можна було додавати, додаткові квіти у панель і не тільки.

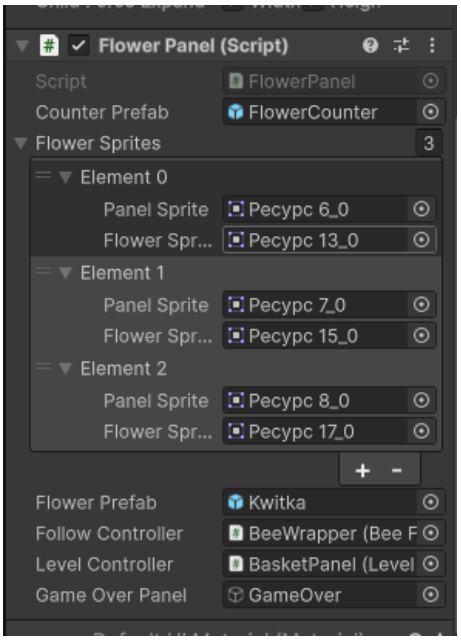

Рисунок 2.37

KwitkaPanel використовує FlowerPanel script в якому є метод Update() в ньому присутні перевірки, як: якщо гра завершена повертається нічого. В перевірці if(AllFlowersCatch()) перевіряється чи було

зібрано всі квітки, якщо так то рівень збільшується та функцією Random змінюються кількість квіток, які гравець має підібрати. Перевірка else if(BlackFlowersCatch()) перевіряє чи було зібрано достатню кількість негативних квіток гравцем, якщо так – то гра завершується, бджола зупиняється та з'являється вікно gameOverPanel. Метод AllFlowersCatch() перевіряє чи були зібрані квітки. Метод BlackFlowersCatch() підраховує скільки квіток стали чорними, якщо дві то повертає true. Start() – це метод який створює FlowerCounter для кожного SpriteTuple, який додається для підрахунку квіток. Також запускається StartCoroutine для створення квіток на сцені. І самий Ienumerator SpawnFlowers() який в своє чергу перевіряє, що якщо гра не завершена то генерується випадкова позиція на екрані по координатах де Screen.height/6 відповідає за те, щоб квітки не створювалися на самій панелі квіток. Далі обирається одна пара спрайтів із масиву, потім отримується спрайт квітки, який буде використовуватися для створення нової квітки на сцені також отримується відповідний лічильник квіток для квітки зі словника. Створюється нова квітка з префаба та розміщується по позиції pos без обертання. Для того, щоб гравець міг взаємодіяти з квіткою, використовується TapOnFlower tap = flower.GetComponent <TapOnFlower>() цей скрипт реагує на взаємодію гравця з квіткою, та після нього у коді йде налаштування нової квітки. Також варто зазначити, що кожна нова квітка створюється у проміжку 1.5 секунд між собою та видаляється через певний час, щоб уникнути перевантаження.

- Об'єкт BasketPanel використовує LevelController script. Його код містить публічне текстове поле, щоб у рушії Unity можна було накласти його на об'єкт, та приватне поле де початкове значення рівня нульове. Також властивість public int Level => level, яка

дозволяє іншим скриптам дізнатися поточний рівень. Функція LevelUp() збільшує рівень на один та оновлює текст у BasketPanel.

- Головний герой бджолік має такі компоненти: Bee Follow Controller script, Rigidbody2D, Circle Collider 2D. Bee Follow Controller відповідає за рух, зміну напрямку, анімацію поразки та літання, а також знищення об’єкта після завершення гри. BeeFollowController використовує метод Update() у якому бджолік плавно рухається до target Position(квітки) за допомогою Lerp(лінійна інтерполяція). Time.deltaTime гарантує однакову швикість, а MovePosition(…) змінює позицію імітуючи фізику. Також для того щоб бджола не літала однією стороною було зроблено перевірку, що якщо квітка праворуч, то бджола дивиться вправо, якщо ліворуч – бджола віддзеркалюється на 180 градусів. Метод Start() у якому бджола немає цілі, тому вона розташована на поточній позиції, вона не рухається допоки гравець не натисне на квітку. Функція BeeGameOver() викликає анімацію поразки, а функція End() знищує об’єкт.
- AngryBee – це зла бджола, яка наслідує клас бджоли, але у неї є власна функція Random() яка кожні 5 секунд змінює свій напрям руху та забирає квітки. AngryBee має такі компоненти: Rigidbody 2D, Circle Collider 2D та сам AngryBeeController script. Як було зазначено вище, клас AngryBeeController наслідує клас BeeFollowController. Метод Start() у якому об’єкт встановлює свою початкову позицію, як targetPosition, та запускається корутина, яка керує періодичними змінами. Ienumerator RandomCords() – це корутина, через яку: допоки гра не завершена то створюється затримка на декілька секунд. Потім екранні координати перетворюються у координати сцени, вибирається випадкова точка на екрані та віднімається Screen.height/6, щоб позиція не була за межами доступної частини

екрану. Після цього створюється нова позиція де координата z залишається не змінною.

- Музика додана так само як і у попередніх рівнях.

Також у цьому рівні присутні ще додаткові класи та їх методи:

- FlowerCounter цей клас відображає квітки, показує поточну кількість зібраних квіток і скільки треба зібрати. Змінює також колір квіток на чорний у панелі, якщо гравець зібрав більше квіток ніж задано. Клас містить перевірки чи було зібрано всі квітки гравцем, чи було більше ніж задано. Містить метод Reset(), який використовується для скидання підрахунку квітів при новому рівні, та зберігає те саме розташування та колір квіток. Метод SetUp() налаштовує лічильник. Метод CatchFlower() викликається, коли гравець зібрав квітку, та має перевірку чи було зібрано більше ніж було задано, якщо так то змінює вигляд квітки на чорну, якщо ні то збільшує лічильник.
- RecordManager відповідає за збереження та відображення рекорду рівня. Цей клас має метод Start() у якому отримується збережений рекорд під ключем “Game3Record”, якщо не було збереження то використовується значення за замовчуванням тобто нуль. У цьому методі присутня перевірка, що якщо поточний рівень гравця більший за збережений рекорд, то оновлюється рекорд, та зберігається нове значення, після чого виводиться оновлений або попередній рекорд на екрані.
- SpriteTuple – це простий структурований контейнер для двох спрайтів, які використовуються для зв’язку між panelSprite та flowerSprite. Також використовується атрибут [Serializable], який дозволяє зберігати або передавати атрибути класів та структур.
- TapOnFlower – це контролер квітки, який з’являється на сцені. Клас взаємодіє з гравцем, бджолою, анімацією збору квіток та автоматичним знищенням після певного часу. Метод

OnTriggerEnter2D відповідає за збір квіток, де: якщо квітка зібрана то повертається нічого, інакше викликається анімація квітки та якщо було зіткнення бджоли з самою квіткою то оновлюється лічильник квіток. Метод SetUp() Викликається коли створюється нова квітка, задає спрайт, зберігає об’єкт для підрахунку квіток, та зберігає followController, щоб знати хто летить до квітки. Метод End() видаляє квітку з гри. On Click() перевіряє, що якщо гравець натискає на квітку то бджолік летить до неї. Корутина DeletFlower() через 10 секунд після появи, квітка автоматично зникає. Метод Start() запускає автоматичне зникнення квітки. Update() використовує перевірку чи гравець натиснув на квітку і якщо так то викликається OnClick(), щоб бджолік полетів до квітки. Також визначається позиція натискання на квітку та перетворення координат з екрану в сцену.

2.8 Висновок

У даному розділі було розглянуто структуру коду та візуальне оформлення гри. Візуальний стиль створює комфортну атмосферу, що сприяє глибшому зануренню гравця в ігровий процес. Дизайн інтерфейсу є простим, інтуїтивно зрозумілим та зручним у використанні.

Для оптимальної організації проекту кожен ігровий режим було розміщено у відповідних папках, що значно полегшує орієнтацію в коді та забезпечує зручність його доопрацювання у разі потреби.

Програмна логіка реалізована без використання штучного інтелекту. Натомість поведінка об’єктів ґрунтується на механізмах випадковості, а їхня кількість контролюється шляхом автоматичного видалення після певного часу, що сприяє оптимізації роботи гри.

РОЗДІЛ 3. ТЕСТУВАННЯ ТА АНАЛІЗ

3.1 Огляд видів тестування

У межах проекту було застосовано кілька типів тестування, зокрема вбудовані інструменти перевірки в кодовому редакторі Rider, засоби налагодження в ігровому рушії Unity, а також мануальне тестування.

Мануальне тестування – це процес перевірки роботи застосунку, який виконується розробником з метою виявлення помилок, дефектів та недоліків у програмному середовищі (11).

Для проекту було проведено такі види ручного тестування:

- Функціональне тестування – це аналіз та перевірка на відповідність між реальною поведінкою об’єктів і вимогами, визначеними розробником. Тобто перевірка того, чи відповідає програма вказаним функціональним вимогам.
- Тестування зручності використання – це оцінка інтерфейсу та перевірка зручності використання застосунку та об’єктів для забезпечення позитивного досвіду користувача. Зокрема оцінювалося наскільки зручно розташовані елементи інтерфейсу та чи мають елементи достатню ширину для коректної взаємодії.
- Кросплатформне тестування – це перевірка роботи гри на різних типах пристроїв, зокрема на декількох моделях мобільних телефонів та на ноутбуці, з метою забезпечення стабільності та однакового користувацького досвіду.
- Тестування інтерфейсу – це перевірка візуального інтерфейсу гри: кольорової гами, контрастності кольорів, розміщення об’єктів на екрані для загального сприйняття.

3.2 Аналіз виявлених помилок(багів)

Під час тестування гри було зроблено висновок, що відсутність кнопки завершення гри ускладнює процес та змушує шукати додаткові рішення, щоб

вийти з гри. Звісно, гравець зможе це обійти, але кнопка виходу справді необхідна для головного меню.

У кожному рівні, а також у головному меню відтворюється музика з мелодіями за замовчуванням. Це може викликати певний дискомфорт у гравця, адже мелодія не завжди відповідає вподобанням користувача. Саме тому було вирішено додати налаштування гучності музики для всіх режимів у головному меню.

Регулювання гучності реалізоване за допомогою повзунка, який дає змогу змінювати гучність фонової музики в реальному часі. Крім того, якщо гравець змінює рівень гучності то це налаштування зберігається і застосовується під час наступного запуску гри.

Раніше, щоб завершити кожен режим гри, необхідно було спочатку програти, аби на екрані з’явилася панель з кнопками “Повернення на головне меню” та “Почати спочатку”, що дуже не зручно в ситуаціях, коли гравець випадково обирав не той режим, або ж хотів перезапустити рівень. Саме тому було реалізовано функціональність кнопок “Пауза” та “Продовжити”. Під час паузи гравець може повернутися до головного меню, почати гру спочатку або тимчасово зупинити гру, щоб відволіктися і не втратити поточний прогрес.

Під час тестування першого рівня було виявлено, що головний персонаж – бджолік – має фізику, яка дозволяє йому падати. Однак через те, що на початку не було враховано фізичні обмеження, персонаж падав і виходив за межі екрану. Для вирішення цієї проблеми були створені порожні об’єкти, розміщені зверху та знизу екрану, які запобігають виходу персонажа за його межі. Ці об’єкти мають властивість BoxCollider 2D завдяки яким гравець не зможе вилетіти за межі ігрового простору.

Також під час розробки першого рівня виникли певні труднощі зі створенням квіток, крізь які має пролітати головний персонаж – бджолік. Під час тестування першого режиму з’ясувалося, що необхідно зменшити розміри BoxCollider2D та збільшити відстань між квітками. Це було важливо для того, щоб уникнути надто тісного розташування квіток, та забезпечити вільний проліт

персонажа між ними. Проте після внесення цих змін, було виявлено, що квітки лише створюються, але не видаляються. Це призвело до накопичення об’єктів (квіток) у сцені рушія Unity, що вже за кілька секунд спричиняло зависання гри. Для вирішення цієї проблеми був створений додатковий порожній об’єкт із прикріпленим скриптом, який перевіряє чи торкнулася квітка цього об’єкта. У разі зіткнення квітка автоматично видаляється. Таке вирішення проблеми забезпечує стабільну продуктивність гри та її функціонування.

Після тестування другого режиму на мобільному пристрої було вирішено збільшити розмір банки з медом. Це покращення дозволило гравцю краще орієнтуватися в ігровому просторі та точніше переміщувати банку в місце, куди має впасти краплина. Також під час тестування другого рівня було виявлено, що з обох боків одночасно створюються бджілки які летять у протилежних напрямках, у результаті вони накопичувалися, що призводило до зниження продуктивності гри. Щоб вирішити цю проблему було реалізовано перевірку: якщо бджілки летять з правої сторони та торкаються лівого спавнера, вони автоматично знищуються, і навпаки – при русі зліва направо. Це рішення дозволило уникнути надмірного накопичення бджілок у сцені та покращило стабільність гри.

Для уникнення накопичення краплин які гравець не встиг зловити під час гри, було створено порожній об’єкт, який перевіряє, чи торкнулася краплина його. Якщо так – краплина автоматично видаляється. Це рішення було впроваджено з метою забезпечити стабільність проходження цього рівня.

Під час тестування третього режиму гри було виявлено, що зла бджілка зникала за фоном через те, що її позиція була встановлена на значення – 10. Цю проблему не вдалося виправити в рушиї Unity, тому довелося переглянути та змінити код, після внесених змін зла бджілка нарешті почала відображатися на екрані. Через певний час тестування було виявлено, що зла бджілка вилітає за межі екрану та заходить на панель підрахунку квітів, що ускладнює ігровий процес. Для усунення цієї проблеми до коду було додано обмеження руху злої бджілки, завдяки чому вона тепер літає правильно. Також було виявлено, що зла

бджілка підбирає квіти і це зараховується гравцеві, що не має бути, так як зла бджілка навпаки забирає квітки собі і не повинна допомагати гравцеві. Ця помилка була виправлена кодом.

Також квітки з’являлися на панелі, що ускладнювало процес проходження рівня. Цю проблему вдалося вирішити так само, як і у випадку зі злою бджілкою за допомогою обмеження позиції об’єктів на сцені.

Під час тестування рівня було виявлено, що коли гравець відволікається від гри, то на екрані накопичується забагато квіток, що ускладнює ігровий процес. Для вирішення цієї проблеми було реалізовано автоматичне видалення квіток через 5-10 секунд після їх появи на екрані, що сприяє покращенню продуктивності гри та зручності сприйняття гравця.

3.3 Атрибути усунення дефектів

Для визначення послідовності усунення дефектів, які були виявлені під час тестування програмного забезпечення використовують такі атрибути як Severity та Priority (12).

Severity – це атрибут, котрий характеризує вплив дефекту на загальну функціональність програмного забезпечення що тестується. Він визначає, наскільки сильно дефект впливає на систему та її користувачів. Severity має таку класифікацію як:

- Blocker – блокуюча помилка, унеможливлює всю подальшу роботу з системою. Для відновлення роботи необхідне виправлення Blocker.
- Critical – критична помилка, порушує роботу основного функціоналу. Баг проявляється постійно і унеможливлює використання ключових функцій системи.
- Major – значна помилка, ускладнює (але не блокує) роботу основного функціоналу, або унеможливлює використання другорядних функцій.

- Minor – незначний баг, на функціонал системи впливає відносно мало, ускладнює використання другорядних функцій. Для обходу цього бага існують очевидні шляхи.
- Trivial – тривіальний баг, не впливає на функціонал проекту, але погіршує загальне враження від роботи з продуктом.

Priority – це атрибут, що визначає важливість виправлення дефекту та послідовність, в якій баги повинні бути виправлені командою розробників. *Priority* встановлюється на основі бізнес-потреб та вимог замовника. Preority має таку класифікацію як:

- Highest – найвищий пріоритет. Призначається екстреним ситуаціям, які дуже негативно впливають на продукт, чи навіть бізнес компанії. Такі баги слід усувати негайно.
- High – високий пріоритет. Призначається багам, які мають бути усунені якнайшвидше.
- Medium – звичайний пріоритет, призначається за замовчуванням. Ці баги усуваються в другу чергу, в штатному порядку.
- Low – низький пріоритет. Призначається багам, які не мають серйозного впливу на функціонал та якість продукту. Виправлення таких багів відбувається в останню чергу, якщо є час та ресурси.

3.5 Класифікація багів за типом тестування та пріоритетом

- “Відсутність кнопки завершення гри” Severity(Major), Priority(***Medium***)
- “Відсутність регулювання гучності” Severity(***Trivial***), Priority(***Low***)
- “Відсутність кнопки пауза та продовження” Severity(***Critical***), Priority(***Highest***)
- “Відсутність кнопки пауза та продовження” Severity(***Critical***), Priority(***Highest***)
- “Відсутність колайдеру на першому рівні” Severity(***Blocker***), Priority(***Highest***)

- “Відсутність достатньої відстані між квітками” Severity(***Blocker***), Priority(***Highest***)
- “Відсутність очищення квіток на першому рівні” Severity(***Critical***), Priority(***Highest***)
- “Невідповідність розміру банки” Severity(***Trivial***), Priority(***Low***)
- “Відсутнє видалення зайвих бджілок” Severity(***Blocker***), Priority(***Highest***)
- “Відсутність колайдеру на другому рівні” Severity(***Blocker***), Priority(***Highest***)
- “Невидимість злої бджілки” Severity(***Major***), Priority(***Highest***)
- “Відсутність обмеження розташування злої бджілки” Severity(***Trivial***), Priority(***Low***)
- “Відсутність обмеження розташування злої бджілки” Severity(***Trivial***), Priority(***Low***)
- “Не правильна робота злої бджілки” Severity(***Critical***), Priority(***High***)
- “Не правильна робота злої бджілки” Severity(***Critical***), Priority(***High***)
- “Відсутність обмеження з’явлення квіток на третьому рівні” Severity(***Trivial***), Priority(***Low***)
- “Не контрольований потік квіток” Severity(***Critical***), Priority(***Medium***)

3.6 Висновок

Тестування знаходить помилки, які були зроблені як в процесі створення коду, а мануальне тестування дозволило зрозуміти, які частини програми не відповідають вимогам та очікуванням. Також без виправлення багів користування програмою було б проблематичним. Тому тестування є важливим процесом для правильної поведінки продукту.

ВИСНОВКИ

Під час проведення кваліфікаційної роботи було створено 2D аркадну гру з трьома доступними режимами, проведено аналіз рушіїв, аналіз колористики та аналіз вже наявних продуктів. Були розроблені вимоги до власного проекту та сама реалізація. Було досягнуто таких цілей:

- Розроблена кольорова схема проекту
- Створено набір ілюстрацій та спрайтів
- Розроблено головну сторінку
- Реалізовано механізм переходу від головної сторінки до режиму гри
- Реалізовано три режими гри
- Створено алгоритм руху противника (зла бджілка)
- Розроблено механізми оптимізації роботи
- Було проведено тестування з виявленням критичних та некритичних багів у роботі проекту
- Була проведена робота по усуненню критичних багів

Отримані результати свідчать про успішне виконання реалізацію функціональних вимог та проект вже може використовуватися в розважальних цілях. Практичне значення розробленого проєкту полягає в створенні повноцінного програмного продукту – мобільної 2D аркадної гри “Бджоліки вперед”, розробленої за допомогою рушія Unity.

Отримані результати можуть бути застосовані при подальшій розробці мобільних ігор, зокрема у сфері казуальних або аркадних застосунків. Розроблені компоненти можуть бути повторно використані в інших проектах, що підвищує ефективність розробки.

Розробка гри не обмежується лише трьома режимами, присутні також два додаткових режима, які будуть реалізовані згодом. У планах розробки вже присутні такі режими: режим в стилі “Hide and Seek” та в стилі побудови вежі. Також вибір створення режимів не обмежується такими типами, бо саме настрій

та ідеї змінюють більшу частину проекту. Також в планах викласти проект на таких платформах як Google Play Market та Steam та Itch.

Після виконаної роботи та аналізу, можна підсумувати, що правильний підбір рушія відіграє важливу роль у проектах і саме тому перед тим як створювати гру, варто звернути увагу на жанр гри і який рушій краще відповідає вимогам для реалізації проекту. Також правильний підбір кольорів відіграє досить важливу роль у створенні візуальних відчутів. Тестування також відіграє велику роль перед випуском проекту тому, що воно виявляє технічні не справності проекту.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1.Офіційний сайт Unity:

Unity Technologies. Unity Engine. URL: https://unity.com/products/unity-engine (дата звернення: 14.05.2025)

2. Офіційний сайт Unreal Engine:

Epic Games. Unreal Engine. URL: https://www.unrealengine.com/ (дата звернення: 14.05.2025).

3. Офіційний сайт Godot Engine:

Godot Engine. Godot Engine. URL: https://godotengine.org/ (дата звернення: 14.05.2025).

4. Стаття на сайті Run-It:

Run IT. Що таке векторна графіка? Run IT. URL: https://www.run-it.com.ua/shcho-take-vektorna-hrafika/ (дата звернення: 14.05.2025).

5. Стаття на сайті Run-It:

Run IT. Що таке растрова графіка? Run IT. URL: https://www.run-it.com.ua/shcho-take-rastrova-hrafika/ (дата звернення: 14.05.2025).

6. Офіційний сайт Adobe Illustrator:

Adobe. Adobe Illustrator pricing & plans. Adobe. URL: https://www.adobe.com/pl/products/illustrator/campaign/pricing.html (дата звернення: 15.05.2025).

7. Офіційний сайт Sketchbook:

Sketchbook, Inc. Sketchbook: Digital art app for desktop and mobile devices. Sketchbook. URL: https://www.sketchbook.com/apps (дата звернення: 15.05.2025).

8. Офіційний сайт Krita:

Krita Foundation. Krita: Цифрове малювання. Творча свобода. URL: https://krita.org/uk/ (дата звернення: 15.05.2025).

9. Стаття про поєднання кольорів:

Marathon Print. Поєднання кольорів. URL: https://marathon-print.com.ua/uk/statti/poyednannya-koloriv/ (дата звернення: 22.05.2025).

10. Стаття про кольори та їх емоційне значення:

VOKI Games. Гейм-палітра: роль кольору в сучасних мобільних іграх. URL: https://vokigames.com/ua/rol-koloru-v-suchasnyh-mobilnyh-igrah/ (дата звернення: 22.05.2025).

11. Мануальне тестування:

BuildApps. Мануальне тестування. URL: https://buildapps.pro/uk/blog/manual-testing (дата звернення: 22.05.2025).

12. Стаття Severity та Priority:

IT-notes. Severity та Priority дефектів (багів). URL: https://www.it-notes.wiki/software-testing/severity-and-priority-defects/ (дата звернення: 22.05.2025).