Міністерство освіти і науки України
Університет митної справи та фінансів

Факультет інноваційних технологій
Кафедра комп'ютерних наук та інженерії програмного забезпечення

Кваліфікаційна робота бакалавра

на тему : «Розробка 2D гри з елементами процедурної анімації»

Виконав: студент групи ІПЗ20-1
Спеціальність 121 «Інженерія програмного забезпечення»
Янчук Владислав Дмитриевич
(прізвище та ініціали)
Керівник к.т.н., доц. Ульяновська Ю.В.
(науковий ступінь, вчене звання, прізвище та ініціали)
Рецензент Університет митної справи та фінансів
(місце роботи)
Доцент кафедри кібербезпеки та інформаційних технологій
(посада)
к.т.н., доцент Савченко В.Ю.
(науковий ступінь, вчене звання, прізвище та ініціали)

Дніпро – 2024

АНОТАЦІЯ

Янчук В.Д. Розробка 2D гри з елементами процедурної анімації.

Кваліфікаційна робота на здобуття освітнього ступеня бакалавр за спеціальністю 121 «Інженерія програмного забезпечення» – Університет митної справи та фінансів, Дніпро, 2024.

Кваліфікаційна робота присвячена розробці 2D гри з процедурною анімацією, що є актуальним завданням у контексті сучасного розвитку індустрії відеоігор. Процедурна анімація, заснована на математичних алгоритмах, дозволяє створювати реалістичні та інтерактивні ігрові середовища, зменшуючи обсяг ручної роботи та підвищуючи гнучкість створення анімацій, що відкриває нові можливості для розробників ігрових додатків. Створюючи більш захоплюючий і реалістичний ігровий досвід для користувачів.

В процесі дослідження було проведено аналіз сучасних методів та підходів до розробки ігрових додатків, а також обрано оптимальні програмні засоби для реалізації проекту. Розробка проводилася із застосуванням ігрового рушія Unity та графічного редактора Krita, що забезпечило створення високоякісних анімацій та інтерактивних елементів гри.

Проект включає розробку алгоритму процедурної анімації, створення ігрових об'єктів та їх взаємодію. Реалізовані функції забезпечують інтерактивність, динамічність та гнучкість гри, дозволяючи створювати реалістичні анімації, які реагують на дії користувача та змінюються в реальному часі.

Ключові слова: 2D гра, процедурна анімація, Unity, Krita, алгоритми, ігрові рушії, анімація.

ABSTRACT

Yanchuk V.D. Development of a 2D Game with Procedural Animation Elements.

Bachelor's thesis for obtaining a degree in Software Engineering, specialty 121. – University of Customs and Finance, Dnipro, 2024.

The qualification work is dedicated to the development of a 2D game with procedural animation, which is a relevant task in the context of the modern development of the video game industry. Procedural animation, based on mathematical algorithms, allows for the creation of realistic and interactive gaming environments, reducing the amount of manual work and increasing the flexibility of creating animations, thus opening new opportunities for game developers. It creates a more engaging and realistic gaming experience for users.

During the research process, an analysis of modern methods and approaches to game development was conducted, and optimal software tools for the project implementation were selected. The development was carried out using the Unity game engine and the Krita graphic editor, which ensured the creation of high-quality animations and interactive game elements.

The project includes the development of procedural animation algorithms, the creation of game objects, and their interaction. The implemented features provide interactivity, dynamism, and flexibility to the game, allowing for the creation of realistic animations that respond to user actions and change in real time.

Keywords: 2D game, procedural animation, Unity, Krita, algorithms, game engines, animation.

ЗМІСТ

ВСТУП

Актуальність проблеми. Сучасна індустрія відеоігор є однією з найдинамічніших галузей, що постійно розвивається завдяки технологічним інноваціям та зростаючому попиту серед користувачів.

Світова індустрія відеоігор демонструє вражаючі темпи зростання. Цього року загальний дохід галузі склав 164,6 мільярди доларів, що на 9,3% більше, ніж у 2022 році. Експерти передбачають, що до 2025 року ця цифра перевищить 300 мільярдів доларів. Основним драйвером такого зростання є постійний розвиток технологій. Сьогодні люди активно грають не тільки на традиційних ПК та ігрових консолях, таких як Microsoft Xbox, Sony PlayStation, та Nintendo Switch, але й на мобільних пристроях, зокрема на смартфонах та планшетах. Наразі мобільні ігри генерують більше ніж половину всіх світових доходів від відеоігор, і ця тенденція має тенденцію до зростання.

Розробка ігрових додатків є складним та багатогранним процесом, який охоплює різноманітні методи та підходи. З розвитком технологій та інструментів для розробки ігор з'являються нові підходи, що дозволяють спростити процес створення ігор, підвищити їх якість та скоротити час розробки.

Розробка 2D ігор з процедурною анімацією є важливим напрямком, що дозволяє створювати більш реалістичні та інтерактивні ігрові середовища. Процедурна анімація, яка базується на використанні математичних алгоритмів для генерації рухів та ефектів, надає можливість зменшити обсяги ручної роботи та забезпечує високу гнучкість у створенні анімацій.

Процедурна анімація дозволяє розробникам створювати складні динамічні системи, які можуть автоматично адаптуватися до змін у ігровому середовищі. Це відкриває нові можливості для креативності, дозволяючи створювати унікальні ігрові світи з мінімальними затратами часу та ресурсів.

Крім того, використання фізичних симуляцій та математичних моделей для створення анімацій сприяє підвищенню якості ігрового досвіду, роблячи його більш захоплюючим та реалістичним.

Зростаючий інтерес до відеоігор як серед розробників, так і серед користувачів, створює значний попит на нові підходи та технології у розробці ігор. Використання гнучких підходів та передових технологій дозволяє створювати інноваційні продукти, що відповідають сучасним вимогам та очікуванням гравців. При виборі програмного забезпечення необхідно враховувати такі фактори, як зручність використання, функціональні можливості інструментів, підтримка різних платформ, наявність активної спільноти користувачів та вартість. У процесі розробки 2D ігор важливим кроком є вибір відповідного ігрового рушія. Кожен рушій має свої унікальні переваги та недоліки, які можуть вплинути на успіх проекту. Вибір програмного забезпечення повинен враховувати різноманітні фактори, такі як зручність використання, функціональні можливості, підтримка різних платформ, наявність активної спільноти користувачів та вартість. В цьому контексті, використання процедурної анімації стає надзвичайно актуальним, адже вона дозволяє створювати високоякісні ігрові продукти, що відповідають сучасним вимогам та очікуванням гравців. Таким чином, дослідження та розробка 2D ігор з процедурною анімацією є важливим кроком у розвитку індустрії відеоігор, сприяючи створенню інноваційних та конкурентоспроможних продуктів на ринку.

Метою роботи є розробка 2D гри з процедурною анімацією.

Методи дослідження: обробка та аналіз інформації, методи проектування та розробки ігор, процедурна анімація.

У відповідності до поставленої мети в кваліфікаційній роботі поставлені наступні завдання дослідження:

1. Проаналізувати технічні засоби, що застосовуються для розробки 2D ігор.

2. Провести проектування гри.

3. Розробити 2D гри з елементами процедурної анімації.

4. Провести тестування.

Об'єктом дослідження є розробка програмного забезпечення для створення ігрових застосунків.

Предметом дослідження є апаратно-програмне забезпечення для розробки ігор з алгоритми процедурної анімації.

Структура роботи:

- Розділ 1 Аналіз існуючих рішень.
- Розділ 2 Аналіз засобів реалізації 2d гри
- Розділ 3 Розробка алгоритму процедурної анімації

Робота складається зі вступу, 3-х розділів, висновків, списку використаних джерел з 15 найменувань. Обсяг роботи 55 сторінок кваліфікаційної роботи, 46 рисунків, 1 таблиці.

РОЗДІЛ 1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1. Аналіз публікацій щодо розробки ігрових додатків

Останнім часом світова індустрія відеоігор демонструє вражаючі темпи зростання. Цього року загальний дохід галузі склав 164,6 мільярди доларів, що на 9,3% більше, ніж у 2022 році. Експерти передбачають, що до 2025 року ця цифра перевищить 300 мільярдів доларів. Основним драйвером такого зростання є постійний розвиток технологій. Сьогодні люди активно грають не тільки на традиційних ПК та ігрових консолях, таких як Microsoft Xbox, Sony PlayStation, та Nintendo Switch, але й на мобільних пристроях, зокрема на смартфонах та планшетах.

Наразі мобільні ігри генерують більше ніж половину всіх світових доходів від відеоігор, і ця тенденція має тенденцію до зростання. Таке зростання можливе завдяки тому, що більш ніж 40% населення світу має власні смартфони або планшети і доступ до швидкісного мобільного інтернету. Крім того, технічний прогрес сприяє підвищенню якості ігрового досвіду на мобільних пристроях.

У 2023 році ігрові додатки стали найпопулярнішою категорією у Google Play, становлячи 13,35% усіх доступних додатків. 2020 року у світі було 2,2 мільярда мобільних геймерів, з яких 203 мільйони знаходилися у США, а 459 мільйонів – у Китаї.

На західних ринках домінують гіперказуальні ігри, екшн, аркади та головоломки, в той час як в Азії популярні мультиплеєрні ігри, включаючи партійні екшн-баталії та MMORPG, які займають топові позиції серед мобільних ігрових жанрів.

Гра – це структурована діяльність, що полягає в імітації іншого виду діяльності з метою розваги. Вона відрізняється від інших видів діяльності, таких як робота, тим, що не має на меті приносити «користь» гравцю. Проте,

ця межа не є абсолютно чіткою, оскільки існують ігри, які можна вважати роботою.

Люди можуть грати в ігри як для задоволення, так і для досягнення певних винагород. Деякі ігри призначені для одиночної гри, інші - для командної або змагань з іншими гравцями. Основними елементами гри є цілі, правила, виклики та взаємодія. Зазвичай ігри включають фізичну або розумову стимуляцію, а часто і те, і інше.

Відеогра – це гра, яка являє собою комп'ютерну програму та використовує мультимедійні можливості комп'ютера. Пристрої для відеоігор включають ПК, консолі, смартфони тощо. Для керування відеоіграми використовують геймпади, мишки, клавіатури, сенсорні екрани та інші контролери.

Ігрова механіка – це правила або обмеження, за якими діє гравець, а також реакції гри на його дії. Сукупність ігрових механік формує організацію ігрового процесу та може визначати жанр гри [2].

Ігровий цикл (або core-gameplay, core-loop) – це набір основних ігрових механік, що повторюються та формують основний ігровий досвід. Це ті дії, які гравець виконує протягом всієї гри. Core-loop може змінюватися та ускладнюватися, додаючи нові механіки або еволюціонуючи старі.

Ігровий цикл має бути:

- зрозумілим;
- легко виконуваним;
- гнучким та різноманітним;
- пов'язаним з іншими ігровими циклами;
- приносити задоволення гравцю.

Мета гра (або meta-gameplay) – це набір механік та систем, які існують навколо основного ігрового циклу, створюючи додаткові або основні цілі для залучення гравця. Вони можуть впливати на core-loop, але не є його частиною. Прикладом є система щоденних завдань у мобільних іграх, яка стимулює гравця заходити в гру щодня.

Meta гра не є обов'язковою частиною відеоігор, адже існує багато ігор, побудованих виключно навколо основного ігрового циклу.

Roguelike – це піджанр відеоігор, що походить від гри «Rogue», випущеної в 1980 році Гленном Віхманом та Майклом Тойєм. Гра стала культовою. Основними характеристиками цього жанру є процедурна генерація рівнів та «перманентна смерть», коли після програшу гравець втрачає весь прогрес і починає знову. Перевагою жанру є висока реіграбельність.

Ігровий рівень – це віртуальний простір, доступний гравцю для виконання ігрових цілей. Після завершення цілей рівня гравець переходить на наступний. Зазвичай кожен наступний рівень складніший. Наповнення рівнів залежить від жанру гри та її механік. Щоб підтримувати інтерес гравця, кожен новий рівень повинен містити новий контент та перешкоди.

Процедурна генерація – це автоматичне створення ігрового контенту за допомогою алгоритмів. Цим методом можуть створюватися текстури, предмети, квести, музика, але найчастіше – ігрові рівні. Процедурна генерація використовує випадкові або псевдовипадкові значення для створення різноманітного контенту. Алгоритми повинні забезпечувати надійність, швидкість, контрольованість та різноманітність згенерованого контенту [4].

1.2. Аналіз методів та підходів розробки ігрових додатків

Розробка ігрових додатків є складним та багатогранним процесом, який охоплює різноманітні методи та підходи. З розвитком технологій та інструментів для розробки ігор з'являються нові підходи, що дозволяють спростити процес створення ігор, підвищити їх якість та скоротити час розробки.

Одним із традиційних методів є водоспадний метод, який передбачає лінійну послідовність етапів: аналіз вимог, проектування, розробка, тестування, впровадження та підтримка. Кожен етап розпочинається після

завершення попереднього, що забезпечує чітку структуру процесу та полегшує управління проектом. Проте цей метод може виявитися занадто жорстким для ігрових проектів, оскільки вимагає чіткого визначення вимог на початку проекту.

Гнучкі методи, або Agile, акцентують увагу на гнучкості та швидкому реагуванні на зміни. Основні принципи Agile включають ітеративну розробку, постійну взаємодію з клієнтом, швидке отримання зворотного зв'язку та самоуправління командою. Найбільш популярні методології Agile – Scrum та Kanban. Ці методи дозволяють швидко адаптуватися до змін у вимогах та покращити якість кінцевого продукту.

Спіральний метод поєднує елементи водоспадного методу та ітеративного підходу. Він складається з кількох циклів, кожен з яких включає етапи планування, аналізу ризиків, розробки та тестування. Цей підхід дозволяє ефективно управляти ризиками та поступово вдосконалювати продукт.

Одним із важливих підходів до розробки ігрових додатків є прототипування. Цей підхід передбачає створення ранньої версії гри з основними функціями для тестування ідей та концепцій. Прототипування дозволяє швидко оцінити життєздатність ігрової механіки та отримати зворотний зв'язок від користувачів на ранніх етапах розробки, що допомагає уникнути великих витрат на неперспективні ідеї.

Ітеративний підхід передбачає розробку гри через серію циклів, кожен з яких завершується створенням робочої версії продукту з новими функціями та поліпшеннями. Цей підхід дозволяє постійно вдосконалювати гру, реагувати на зворотний зв'язок від користувачів та адаптуватися до змін у вимогах.

Важливим елементом сучасної розробки ігор є використання ігрових рушіїв. Ігровий рушій — це середовище розробки програмного забезпечення, призначене для створення відеоігор. Сучасні ігрові рушії значно спрощують процес розробки відеоігор, надаючи рівень абстракції, що дозволяє експортувати гру на різні платформи з мінімальними змінами. Ігрові рушії

забезпечують основні інструменти та сервіси для створення ігор, що дозволяє зосередитися на створенні контенту та ігрової механіки. Основні компоненти ігрових рушіїв включають:

1) Вхідні дані
2) Графіка
3) Аудіо
4) Фізика
5) Штучний інтелект

Приклади популярних ігрових рушіїв включають:

Godot — це безкоштовний крос-платформний ігровий рушій для створення 2D та 3D ігор, орієнтованих на ПК, мобільні та веб-платформи. Його архітектура побудована навколо вузлів, які організовані в сцени. Godot підтримує мови програмування C++, C#, GDScript та інші. Графічний рушій використовує OpenGL ES 3.0 або OpenGL ES 2.0 і включає окремий 2D-графічний рушій. Основні недоліки Godot включають недостатню документацію, проблеми з імпортом 3D-ресурсів та слабкий фізичний рушій. Godot є безкоштовним і має ліцензію MIT.

Unity — крос-платформенний ігровий рушій від Unity Technologies, вперше представлений у 2005 році. Unity підтримує більше 25 платформ, включаючи мобільні пристрої, ПК, консолі та віртуальну реальність. Рушій надає можливість створювати 2D та 3D ігри, використовуючи C# та візуальне сценаріювання Bolt. Unity має розвинений 2D-рендеринг та підтримує різноманітні ефекти для 3D-ігор. Unity Asset Store дозволяє розробникам купувати та продавати ассети. Недоліки Unity включають слабкі графічні можливості та високі вимоги до пам'яті. Unity має безкоштовну версію Personal, а також платні версії Plus, Pro та Enterprise.

Unreal Engine 5 – крос-платформенний ігровий рушій від Epic Games, відомий створенням високотехнологічних AAA проектів. Unreal Engine 5 використовує мову візуального програмування Blueprint та мову C++. Unreal

Engine 5 безкоштовний, але розробники повинні віддавати 5% доходу компанії Epic Games, якщо їхні доходи перевищують 5000$ за квартал.

Phaser — ігровий фреймворк для створення 2D HTML5 відеоігор для ПК та мобільних пристроїв. Phaser використовує WebGL та Canvas рендерери, а ігри запускаються як JavaScript у браузері. Переваги Phaser включають відмінну підтримку мобільних ігор та стабільність, але він обмежений тільки 2D-графікою.

GameMaker — крос-платформенний ігровий рушій від YoYo Games, призначений для розробників-початківців. GameMaker Studio 2 дозволяє створювати ігри за допомогою мови візуального програмування або скриптової мови, схожої на JavaScript. Переваги GameMaker Studio 2 включають підтримку багатьох платформ та простоту програмування, але він більше орієнтований на 2D-графіку і є платним.

Використання крос-платформних інструментів, таких як Unity або Unreal Engine, дозволяє розробникам створювати один код, який потім може бути експортований на різні платформи з мінімальними змінами. Це значно знижує витрати на розробку та підтримку ігор.

1.3. Висновок до першого розділу

Аналіз публікацій та існуючих рішень у сфері розробки ігрових додатків показав, що сучасна індустрія відеоігор стрімко розвивається. Основними драйверами цього зростання є постійний технологічний прогрес, розширення ринку мобільних ігор та широке розповсюдження швидкісного мобільного інтернету. Відеоігри вже стали не тільки розвагою, але й важливим елементом цифрової економіки, приносячи значні доходи та створюючи нові можливості для розвитку.

У процесі дослідження було розглянуто різні методи розробки ігрових додатків, включаючи водоспадний метод, гнучкі методи (Agile), спіральний метод та прототипування. Водоспадний метод забезпечує чітку послідовність

етапів розробки, але є менш гнучким у порівнянні з Agile, який дозволяє швидко реагувати на зміни у вимогах. Спіральний метод поєднує переваги обох підходів, акцентуючи увагу на управлінні ризиками. Прототипування дозволяє швидко оцінити життєздатність ігрових концепцій на ранніх етапах розробки, що допомагає уникнути великих витрат на неперспективні ідеї.

Важливим аспектом розробки ігрових додатків є вибір ігрового рушія. Сучасні рушії, такі як Unity, Unreal Engine, Godot, Phaser та GameMaker Studio 2, надають розробникам широкий спектр інструментів для створення високоякісних ігор. Кожен з них має свої переваги та недоліки, що дозволяє обирати оптимальний рушій залежно від специфіки проекту. Unity відзначається своєю універсальністю та підтримкою крос-платформенності, Unreal Engine – потужним графічним рушієм, Godot – безкоштовністю та відкритим кодом, Phaser – простотою використання для веб-ігор, а GameMaker Studio 2 – зручністю для початківців [13].

Використання гнучких підходів та передових технологій дозволяє створювати інноваційні продукти, що відповідають сучасним вимогам та очікуванням гравців.

РОЗДІЛ 2. АНАЛІЗ ЗАСОБІВ РЕАЛІЗАЦІЇ 2D ГРИ

2.1. Вибір програмних засобів для реалізації проекту

При виборі програмного забезпечення необхідно враховувати такі фактори, як зручність використання, функціональні можливості інструментів, підтримка різних платформ, наявність активної спільноти користувачів та вартість. У сучасних умовах розробники можуть скористатися різноманітними програмними засобами, такими як Unity, Unreal Engine, Godot, Phaser та GameMaker Studio 2. Кожен із цих інструментів має свої особливості, переваги та недоліки, які необхідно ретельно розглянути перед початком розробки. Крім того, важливо звернути увагу на додаткові програми для роботи з графікою, зокрема Adobe Photoshop та Krita.

Unity є одним із найпотужніших ігрових рушіїв, який підтримує розробку як 2D, так і 3D ігор. Його основними перевагами є інтуїтивно зрозумілий інтерфейс, що дозволяє новачкам швидко освоїтися, та розвинений редактор для роботи зі спрайтами, анімаціями та фізикою 2D об'єктів. Unity також забезпечує підтримку крос-платформенності, що дозволяє розробляти ігри для більш ніж 25 платформ, включаючи ПК, мобільні пристрої та консолі. Велика спільнота користувачів та розробників надає доступ до численних навчальних матеріалів та підтримки. Проте Unity має свої недоліки: високі вимоги до ресурсів можуть призвести до проблем із продуктивністю на слабких пристроях, а також платність деяких версій для комерційних проектів [1].

Unreal Engine також підтримує розробку 2D ігор, хоча основний акцент рушія зроблено на 3D ігри. Перевагами Unreal Engine є потужний графічний рушій, що дозволяє створювати високоякісну графіку навіть у 2D іграх, а також інструменти для візуального сценаріювання, зокрема мова Blueprint, яка дозволяє створювати логіку гри без написання коду. Проте Unreal Engine має

більш складний інтерфейс та інструментарій, що може бути складним для новачків, а також високі вимоги до ресурсів [15].

Godot є безкоштовним та відкритим ігровим рушієм, який спеціалізується як на 2D, так і на 3D іграх. Godot вирізняється зручним інтерфейсом та інструментами для розробки 2D ігор, підтримкою багатьох мов програмування, зокрема C++, C#, GDScript та інших. Важливою перевагою Godot є його повна безкоштовність та ліцензія MIT, що дозволяє використовувати рушій без будь-яких комерційних обмежень. Однак недоліками Godot є недостатня документація та проблеми з імпортом 3D-ресурсів, що може бути критично для проектів, які поєднують 2D та 3D елементи [12].

Phaser є фреймворком для створення 2D HTML5 ігор, який ідеально підходить для веб-ігор. Його основними перевагами є простий та зрозумілий API, підтримка веб-технологій, зокрема WebGL та Canvas для рендерингу, що забезпечує хорошу продуктивність на різних пристроях. Phaser є відкритим та безкоштовним фреймворком, що робить його доступним для широкого кола розробників. Недоліками Phaser є його обмеженість лише 2D графікою та відсутність розширених інструментів для розробки, які пропонують такі рушії, як Unity чи Unreal Engine [10].

GameMaker Studio 2 є популярним інструментом для розробки 2D ігор, орієнтованим на початківців. Його основними перевагами є простота використання, можливість створювати ігри за допомогою візуального сценаріювання або скриптової мови, схожої на JavaScript, а також підтримка багатьох платформ, включаючи ПК, мобільні пристрої та консолі. Однак GameMaker Studio 2 більше орієнтований на 2D графіку, хоча й підтримує 3D, і є платним інструментом, хоча пропонує безкоштовну пробну версію.

У процесі розробки 2D ігор важливим кроком є вибір відповідного ігрового рушія. Кожен рушій має свої унікальні переваги та недоліки, які можуть вплинути на успіх проекту. Вибір програмного забезпечення повинен враховувати різноманітні фактори, такі як зручність використання,

функціональні можливості, підтримка різних платформ, наявність активної спільноти користувачів та вартість. У таблиці 2.1 нижче наведено аналіз основних ігрових рушіїв, що використовуються для розробки 2D ігор, з зазначенням їхніх ключових переваг та недоліків.

Таблиця 2.1

Порівння ігрових рушіїв

Ігровий рушій	Переваги	Недоліки
Unity	Інтуїтивно зрозумілий інтерфейс, розвинений редактор для 2D, підтримка крос-платформенності, велика спільнота	Високі вимоги до ресурсів, платність деяких версій для комерційних проектів
Unreal Engine	Потужний графічний рушій, інструменти для візуального сценаріювання, високоякісна графіка	Складний інтерфейс та інструментарій, високі вимоги до ресурсів
Godot	Зручний інтерфейс, підтримка багатьох мов програмування, повна безкоштовність, ліцензія MIT	Недостатня документація, проблеми з імпортом 3D-ресурсів
Phaser	Простий та зрозумілий API, підтримка WebGL та Canvas, безкоштовність	Обмеженість лише 2D графікою, відсутність розширених інструментів для розробки
GameMaker Studio 2	Простота використання, можливість створювати ігри за допомогою візуального сценаріювання, підтримка багатьох платформ	Більше орієнтований на 2D графіку, платний інструмент

Крім основних ігрових рушіїв, важливу роль у розробці 2D ігор відіграють програми для роботи з графікою. Adobe Photoshop та Krita є двома провідними інструментами, які використовуються для створення та обробки графічних елементів.

Adobe Photoshop є широко відомим професійним інструментом для обробки зображень та графіки. Photoshop надає потужні інструменти для

створення спрайтів, текстур та інших графічних елементів. Основними перевагами Photoshop є його багатий функціонал, підтримка широкого спектру форматів файлів та можливість роботи з шарами, що дозволяє створювати складні графічні композиції. Однак Photoshop є комерційним програмним забезпеченням, що може бути дорогим для індивідуальних розробників та невеликих студій.

Krita є безкоштовним та відкритим програмним забезпеченням для малювання та обробки графіки. Krita спеціалізується на цифровому живопису та анімації, що робить його ідеальним інструментом для створення художніх елементів ігор. Krita підтримує роботу з шарами, має потужні інструменти для малювання та анімації, а також багатий набір кистей та текстур. Основними перевагами Krita є його безкоштовність та відкритий вихідний код, що дозволяє розробникам адаптувати програму під свої потреби. Недоліками Krita можуть бути менша кількість професійних функцій порівняно з Photoshop та менш розвинена підтримка форматів файлів.

2.2. Програмні засоби для розробки 2D гри

Для реалізації проекту з розробки 2D гри було обрано ігровий рушій Unity із використанням мови програмування C#, а також програму для роботи з графікою Krita. Вибір цих інструментів зумовлений їхніми потужними можливостями, зручністю використання та широкою підтримкою спільноти. Детальний аналіз цих інструментів допоможе зрозуміти їхні ключові переваги, функціональні можливості та причини, через які вони були обрані для даного проекту.

Unity є одним із найпопулярніших ігрових рушіїв, що використовується для розробки як 2D, так і 3D ігор. Його універсальність та потужні інструменти роблять його ідеальним вибором для широкого спектру ігрових проектів. Основні аспекти Unity, що роблять його привабливим для розробників:

1) Інтуїтивно зрозумілий інтерфейс

Unity має зручний і зрозумілий інтерфейс, який дозволяє новачкам швидко освоїтися та почати працювати над проектами. Інтерфейс включає зручні інструменти для роботи зі спрайтами, анімаціями та фізикою 2D об'єктів.

2) Підтримка крос-платформенності

Рушій дозволяє розробляти ігри для більш ніж 25 платформ, включаючи ПК, мобільні пристрої, консолі та VR/AR, що забезпечує широкі можливості для розповсюдження гри.

3) C#

Використання мови програмування C# дозволяє створювати складну ігрову логіку, використовуючи сучасні підходи до програмування. Вона є потужною та гнучкою мовою, що підтримує об'єктно-орієнтоване програмування.

4) Активна спільнота та навчальні матеріали

Unity має велику та активну спільноту користувачів, що надає доступ до численних навчальних ресурсів, форумів та документації. Це значно полегшує процес навчання та вирішення проблем під час розробки.

5) Розвинений редактор

Unity Editor надає розширені можливості для налаштування ігрових сцен, розміщення об'єктів, налаштування фізики та анімацій. Інструменти для налагодження та профілювання допомагають оптимізувати продуктивність гри [8].

Krita є потужним безкоштовним інструментом для цифрового малювання та обробки графіки. Ця програма спеціалізується на створенні художніх елементів для ігор та анімацій. Основні переваги Krita, що роблять її ідеальним вибором для роботи з графікою у 2D іграх:

1) Безкоштовність та відкритий код

Krita є повністю безкоштовною програмою з відкритим кодом, що дозволяє її використовувати без будь-яких комерційних обмежень. Це робить Krita доступною для всіх розробників, незалежно від бюджету проекту.

2) Потужні інструменти для малювання

Krita надає широкий набір інструментів для цифрового малювання, включаючи різноманітні кисті, текстури та інструменти для обробки зображень. Програма підтримує роботу з шарами, що дозволяє створювати складні графічні композиції.

3) Інструменти для анімації

Krita має вбудовані інструменти для створення анімацій, що дозволяє легко створювати анімовані спрайти для ігор. Інтерфейс для анімації є зручним та інтуїтивно зрозумілим.

4) Підтримка різних форматів файлів

Krita підтримує широкий спектр форматів файлів, що дозволяє легко експортувати графічні елементи для використання в Unity. Це забезпечує зручний робочий процес між створенням графіки та інтеграцією її в ігровий рушій.

5) Розширені можливості налаштування

Krita дозволяє користувачам налаштовувати інтерфейс та інструменти під свої потреби, що забезпечує гнучкість у роботі та підвищує продуктивність.

Вибір Unity із використанням C# та Krita для реалізації проекту з розробки 2D гри є обґрунтованим рішенням, яке базується на їхніх потужних можливостях та зручності використання.

2.3 Висновок до другого розділу

У цьому розділі було проведено аналіз засобів реалізації 2D гри, що включає вибір програмних інструментів, які найкраще відповідають вимогам проекту. Основним акцентом аналізу став вибір ігрового рушія та графічних інструментів, які забезпечать високий рівень продуктивності та якісної графіки.

В процесі аналізу було розглянуто кілька популярних платформ і технологій для розробки 2D ігор, серед яких Unity, Unreal Engine, Godot, Phaser та GameMaker Studio 2. Після ретельного порівняння було обрано Unity як основний ігровий рушій. Його переваги включають широку функціональність, підтримку крос-платформенності, а також активну спільноту користувачів, що забезпечує доступ до великої кількості ресурсів та додаткових інструментів.

Для створення графічних елементів гри було обрано Adobe Photoshop та Krita. Krita було вибрано завдяки її безкоштовності та потужним можливостям для цифрового малювання та анімації, що дозволяє створювати високоякісні графічні ресурси для гри. Photoshop залишився у резерві як додатковий інструмент для редагування та доопрацювання графічних матеріалів.

Таким чином, проведений аналіз підтвердив доцільність обраних інструментів та підходів для реалізації проекту 2D гри. Використання Unity та Krita забезпечить створення ефективного, надійного та зручного у використанні ігрового продукту, що відповідає сучасним вимогам гравців і дозволяє досягти високого рівня якості. Цей вибір сприятиме максимальній ефективності використання наявних ресурсів та забезпечить успішну реалізацію проекту.

РОЗДІЛ 3. РОЗРОБКА 2D ГРИ З ЕЛЕМЕНТАМИ ПРОЦЕДУРНОЇ АНІМАЦІЇ

3.1. Концепція алгоритму процедурної анімації

Розробка процедурної анімації, порівняно з традиційними способами є має незвичайну реалізацію, але даний принцип має ряд свої власних особливостей завдяки яким залишається актуальним під час проектування сучасних відео-ігор. У порівнянні з традиційними методами, які вимагають значних зусиль для ручного створення кожного кадру, процедурна анімація дозволяє використовувати математичні алгоритми для генерації рухів та ефектів.

Процедурна анімація забезпечує високу гнучкість і адаптивність. Алгоритмічний підхід дозволяє легко змінювати параметри анімації, що важливо для створення інтерактивних середовищ у реальному часі, що використовуються у відеоіграх і віртуальній реальності. Це відкриває нові можливості для креативності, дозволяючи художникам і розробникам експериментувати з різними варіантами анімації без необхідності створювати нові кадри з нуля.

Процедурна анімація також сприяє створенню більш реалістичних і фізично правдоподібних ефектів. Використання фізичних симуляцій та математичних моделей дозволяє створювати природні явища, такі як вода, вогонь, дим та інші. Це не тільки підвищує загальну якість візуальних ефектів, але й допомагає створювати більш захоплюючі та реалістичні віртуальні світи.

Сучасні програмні інструменти, такі як Houdini, Unreal Engine та Unity, активно підтримують процедурні методи, що дозволяє професіоналам у галузі графіки легко інтегрувати ці техніки у свої проекти. Це робить процедурну анімацію важливим елементом сучасного виробничого процесу, сприяючи створенню інноваційного та високоякісного контенту.

3.2. Етапи створення анімації

Для створення алгоритму процедурної анімації потрібно розподілити завдання на декілька етапів (рис. 3.1).

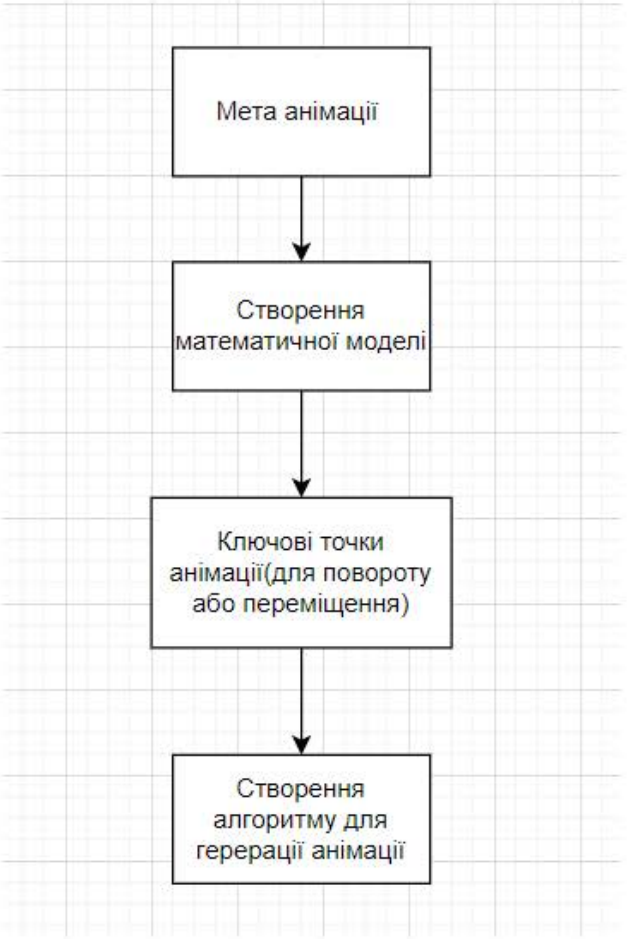

Рисунок 3.1 – Етапи створення процедурної анімації

1) Спочатку потрібно, визначити що потрібно анімувати.

2) Далі потрібно створити математичну модель для створення анімації. Математична модель це сукупність математичних алгоритмів, що виконуються в певному поряду для створення анімації або збільшення її плавності.

3) Ключові точки використовуються для анімації частини об’єкта: тобто перемістити на певне місце, повернути або обернути навколо цієї точки (рис. 3.2).

4) Створення алгоритму для анімації – це функція, яка взаємодіє з ключовими точками об’єкта анімації.

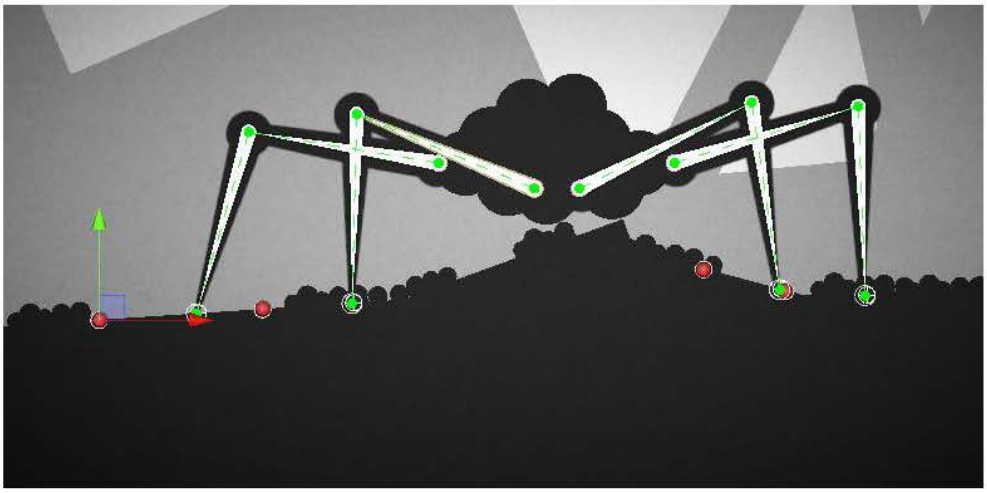

Рисунок 3.2 - Приклад ключових точок анімації

Тобто для демонстрації процедурної анімації розробнику потрібно опрацювати наступні етапи:

1. Визначити інструменти розробки для створення процедурної анімації
2. З’ясувати, які об’єкти та яким чином вони будуть анімовані
3. Створити математичні моделі для генерування анімації
4. Створити ігрові механіки взаємодії при яких буде використовуватися об’єкти анімації для їх демонстрації.

3.3. Інструменти розробки

Для розробки даного проекту потрібно обрати інструменти, які б надали зручне середовище для створення процедурної анімації. Для цього краще

використовувати – ігровий рушій. Серед популярних рушіїв можна виділити наступні: Godot, Unreal Engine, Unity.

Для розробки було обрано Unity. Unity – це потужний ігровий рушій, який надає розробникам широкий спектр інструментів для створення анімації (рис. 3.3).

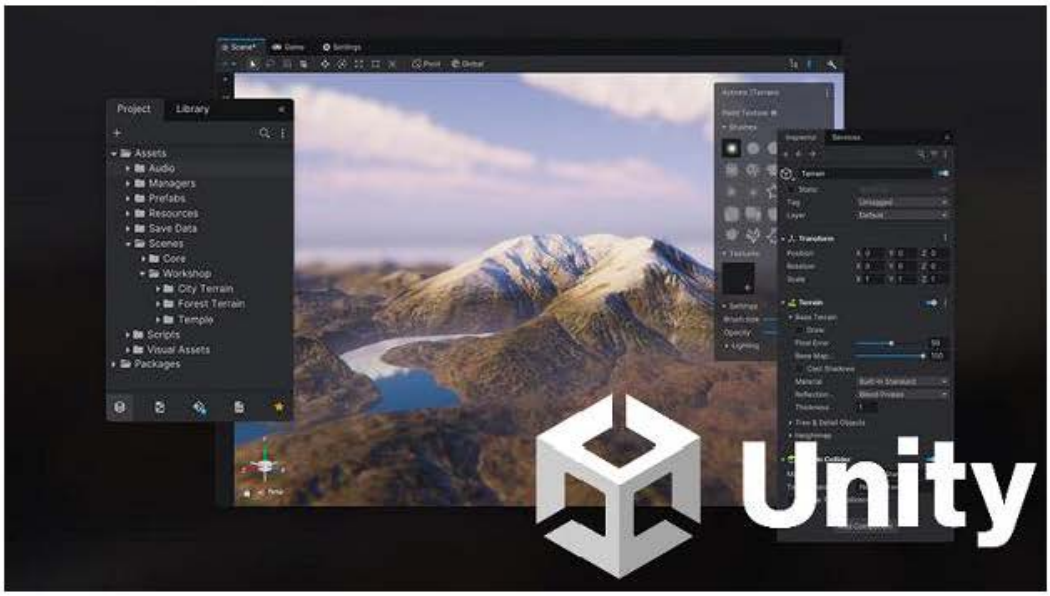

Рисунок 3.3 – Ігровий рушій Unity

В контексті процедурної анімації Unity вирізняється своєю гнучкістю, інтегрованими можливостями та підтримкою різних платформ, що робить його ідеальним вибором для реалізації складних анімаційних проектів. Однією з ключових особливостей Unity є його інтегрована система фізики, яка дозволяє створювати реалістичні анімації, базовані на фізичних законах. Завдяки використанню PhysX та інших фізичних рушіїв, розробники можуть легко симулювати динаміку рідин, зіткнення об'єктів, гравітацію та інші фізичні явища. Це особливо корисно для створення процедурних анімацій, де рухи генеруються автоматично на основі фізичних параметрів.

Unity також пропонує потужні інструменти для створення скриптів, зокрема використання мов програмування таких як C#. За допомогою даної мови розробники можуть створювати складні алгоритми процедурної анімації,

які реагують на взаємодії користувача або змінюються в реальному часі. Це дозволяє створювати інтерактивні та адаптивні анімаційні системи, які можуть змінюватися залежно від умов гри або середовища.

Мова програмування C# має наступні переваги:

- розробники можуть створювати класи, об'єкти та інтерфейси, що спрощує управління складними ігровими системами та анімаціями
- C# інтегрується з платформою .NET, що забезпечує доступ до великої бібліотеки класів і функцій для роботи з файлами, мережами, базами даних та іншими важливими аспектами розробки додатків.
- Unity підтримує інтеграцію з Visual Studio – одним з найпотужніших середовищ розробки для C#. Visual Studio надає потужні інструменти для відлагодження, автодоповнення коду, управління версіями та багато іншого, що значно підвищує продуктивність розробку проекту [9].

Для створення текстур в проекті використовується графічний редактор Krita.

Krita – це безкоштовний та відкритий графічний редактор, який широко використовується для створення цифрових малюнків, концепт-арту, коміксів та текстур для ігор. Він надає потужні інструменти для художників, а також можливості для створення високоякісних текстур, які можуть бути використані в Unity та інших ігрових рушіях.

3.4 Розробка проекту

Щоб розпочати розробку, необхідно створити новий проект в Unity, створити декілька об’єктів та скрипти для їх анімації та контролю. Загалом структура класів проекту для контролю об’єктів виглядає наступним чином (рис. 3.4) [11]:

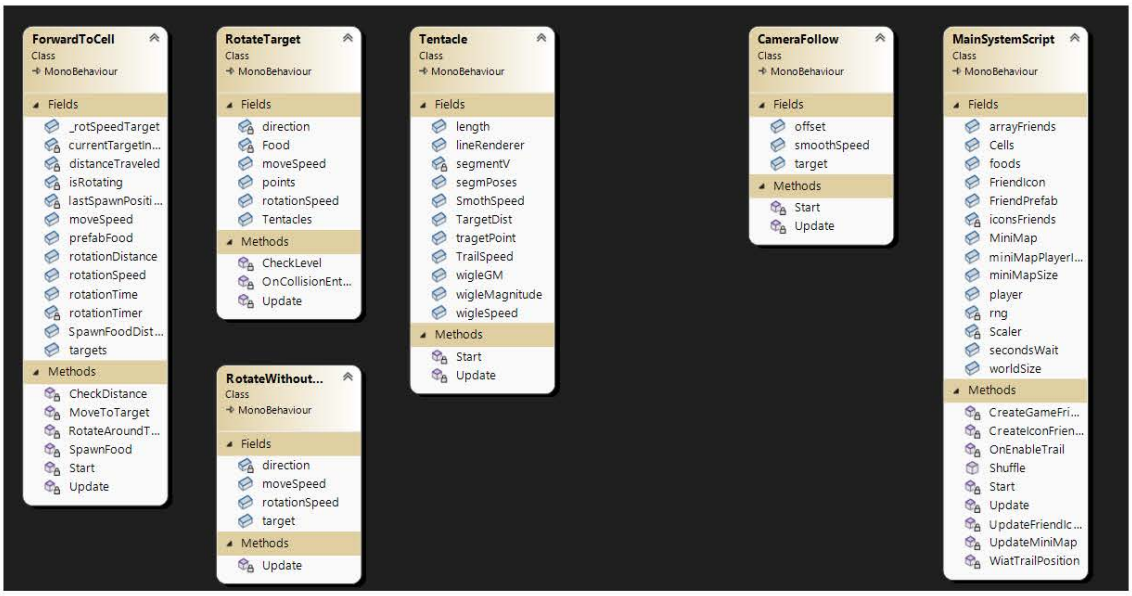

Рисунок 3.4 – Діаграма класів проекту

Потрібно розібрати кожен клас, його функції, та як вони працюють.

Перший клас MainSystemScript - це головний клас проекту, який виконує роль «адміністратора». Контролює яка кількість об’єктів на сцені, та створює додаткові об’єкти або обмежує їх створення, якщо це потрібно.

```
public class MainSystemScript : MonoBehaviour
{
    // Start is called before the first frame update
    public float secondsWait;
    public List<Transform> Cells;
    public GameObject FriendPrefab;//Об'єкт який будем спамити
    public List<GameObject> arrayFriends;
    private  System.Random rng = new System.Random();

    //MiniMap
    public Transform player; // Посилання на гравця
    public Transform MiniMap;
    public RectTransform miniMapPlayerIcon; // Іконка гравця на міні-карті
    public GameObject FriendIcon;
    private List<GameObject> iconsFriends;
    public float worldSize = 81f; // Розмір світу
    public float miniMapSize = 100f; // Розмір міні-карти
    private float Scaler;
```

Рисунок 3.5 – Вхідні дані класу MainSystemScript

Розглянемо функцію CreateGameFriends. Вона створює та ініціалізує ігрові об'єкти "friends" на сцені.

```
void  CreateGameFriends()
{
    for (int i = 0; i<Cells.Count; i++)
    {
        GameObject go = Instantiate(FriendPrefab, new Vector2(0,20), Quaternion.identity);
        CreateIconFriendMap(new Vector2(0, 20));
        go.name = "Friend " + i.ToString();
        ForwardToCell forwardToCell = go.GetComponent<ForwardToCell>();
        List<Transform> shuffledCells = Shuffle(new List<Transform>(Cells));
        foreach (var cell in Cells)
        {
            Debug.Log(cell.name);
        }
        Debug.Log("///////");
        forwardToCell.targets = shuffledCells;
        arrayFriends.Add(go);
    }

}
```

Рисунок 3.6 – Функція CreateGameFriends

Вона проходить через всі кординати у списку Cells, і для кожної позиції створює новий об'єкт на основі шаблону FriendPrefab з початковою позицією Vector2(0, 20). Після створення об'єкта також створюється іконка для міні-карти через виклик функції CreateIconFriendMap. Ім'я кожного нового об'єкта встановлюється як "Friend" з відповідним номером. Функція отримує компонент ForwardToCell з об'єкта, та перемішує список позицій Cells і встановлює його як цільові позиції для компонента ForwardToCell. В кінці, створений об'єкт додається до списку arrayFriends для подальшого керування.

Функція UpdateMiniMap відповідає за оновлення позиції іконки гравця на міні-карті, щоб відображати його поточне розташування в ігровому світі.

```
1 reference
void UpdateMiniMap()
{
    // Отримання позиції гравця
    Vector3 playerPosition = player.position;

    // Масштабування позиції до розмірів міні-карти

    // Перетворення позиції в координати міні-карти
    Vector2 miniMapPos = new Vector2(playerPosition.x * Scaler, playerPosition.y * Scaler);

    // Оновлення позиції іконки гравця на міні-карті
    miniMapPlayerIcon.anchoredPosition = miniMapPos;
}
```

Рисунок 3.7 – Функція UpdateMiniMap

Спочатку функція отримує поточну позицію гравця за допомогою параметра player.position. Далі ця позиція масштабується відповідно до розмірів міні-карти за допомогою параметра Scaler, щоб координати ігрового світу відповідали координатам міні-карти. Після цього позиція гравця перетворюється в координати міні-карти і зберігається у змінній miniMapPos. Далі функція оновлює позицію іконки гравця на міні-карті, встановлюючи властивість anchoredPosition об'єкта miniMapPlayerIcon на значення miniMapPos. Таким чином, іконка гравця на міні-карті переміщується відповідно до його рухів у грі.

Дані на міні-карті оновлюються в реальному часі за допомогою функції Update (рис. 3.8):

```
void Update()
{
    UpdateMiniMap();
    UpdateFriendIconsMaps();
}
```

Рисунок 3.8 – Функція Update

Функція Update в Unity є одним з основних методів, який викликається на кожному кадрі гри. Вона використовується для реалізації логіки, яка повинна виконуватися постійно під час роботи гри, таких як обробка вводу користувача, оновлення стану об'єктів, виконання анімацій та фізичних обчислень.

Функція UpdateFriendIconsMaps відповідає за оновлення позицій іконок друзів на міні-карті відповідно до їх поточного розташування у світі гри.

```
void UpdateFriendIconsMaps()
{
    for (int i = 0; i< arrayFriends.Count; i++)
    {
        Vector2 friendPosition = arrayFriends[i].transform.position;
        Vector2 miniMapIcon = new Vector2(friendPosition.x * Scaler, friendPosition.y * Scaler);
        RectTransform rectTransform = iconsFriends[i].GetComponent<RectTransform>();
        rectTransform.anchoredPosition = miniMapIcon;
    }
}
```

Рисунок 3.9 – Функція UpdateFriendIconsMaps

Функція проходить через всі об'єкти в списку arrayFriends, використовуючи цикл for. Для кожного об’єкту отримується його поточна позиція у світі за допомогою arrayFriends[i].transform.position. Це значення зберігається у змінній friendPosition. Позиція об’єкту у світі масштабується до координат міні-карти. Це робиться шляхом множення координат x і y на значення Scaler, яке відповідає співвідношенню між розмірами світу і міні-карти. Результат зберігається у змінній miniMapIcon. Позиція іконки на міні-карті оновлюється відповідно до масштаборованих координат. Це робиться шляхом встановлення властивості anchoredPosition об'єкта RectTransform на значення miniMapIcon (рис. 3.10).

```
1 reference
void CreateIconFriendMap(Vector2 position)
{
    GameObject _iconFriendMap = Instantiate(FriendIcon, new Vector2(position.x * Scaler, positio
    _iconFriendMap.transform.SetParent(MiniMap, false);
    iconsFriends.Add(_iconFriendMap);
}
```

Рисунок 3.10 – Функція CreateIconFriendMap

Функція CreateIconFriendMap створює відповідні знаки на мапі, які показують приблизне положення об’єкту в ігровому світі.

Остання функція даного класу це Shuffle.

```
1 reference
public  List<Transform>  Shuffle(List<Transform> list)
{
    for (int i = list.Count - 1; i > 0; i--)
    {
        int j = Random.Range(0, i + 1);
        Transform temp = list[i];
        list[i] = list[j];
        list[j] = temp;
    }
    return list;
}
```

Рисунок 3.11 – Функція Shuffle

Дана функція використовується для перемішування об’єктів масиву у випадковому порядку для об’єктів, які переміщуються між різними координатами.

Наступний клас це ForwardToCell. Даний клас має собі функції, що використовуються для навігації та переміщення об’єкту за певним маршрутом (рис. 3.12).

```
public class ForwardToCell : MonoBehaviour
{
    public List<Transform> targets = new List<Transform>(); // Масив об'єктів для переміщення
    public float moveSpeed = 5f; // Швидкість переміщення
    public float rotationSpeed = 50f; // Швидкість обертання
    public float rotationDistance = 10f; // Відстань обертання
    public float rotationTime = 50f; // Час обертання
    public float _rotSpeedTarget;

    private int currentTargetIndex = 0;
    private bool isRotating = false;
    private float rotationTimer = 0f;

    //FoodSpawn
    public float SpawnFoodDistace = 10f;
    public GameObject prefabFood;
    private float distanceTraveled;
    private Vector3 lastSpawnPosition;
    // Start is called before the first frame update
```

Рисунок 3.12 – Клас ForwardToCell

Функція MoveToTarget відповідає за переміщення об'єкта до цільової позиції, обраної з масиву цільових точок **targets** (рис. 3.13).

```
1 reference
void MoveToTarget()
{
    Transform target = targets[currentTargetIndex];
    //rotation
    Vector2 _direction = target.position - transform.position;
    float angle = Mathf.Atan2(_direction.y, _direction.x) * Mathf.Rad2Deg;
    Quaternion rotation = Quaternion.AngleAxis(angle, Vector3.forward);
    transform.rotation = Quaternion.Slerp(transform.rotation, rotation, _rotSpeedTarget * Time.deltaTime);

    //
    Vector2 direction = (target.position - transform.position).normalized;
    float distance = Vector2.Distance(transform.position, target.position);

    if (distance > rotationDistance)
    {
        Vector2 newPosision = direction * moveSpeed * Time.deltaTime;
        transform.position += new Vector3(newPosision.x, newPosision.y, 0);
    }
    else
    {
        isRotating = true;
        rotationTimer = rotationTime;
    }
}
```

Рисунок 3.13 – Функція MoveToTarget

Вона обробляє як обертання об'єкта навколо цілі, так і його переміщення до нього. На початку функції визначається цільова позиція, до якої об'єкт повинен переміщатися. Це робиться за допомогою Transform target, де targets[currentTargetIndex] визначає поточну ціль. Спочатку функція обчислює напрямок до цілі за допомогою Vector2 _direction, що є різницею між позиціями цілі та об'єкта. Потім обчислюється кут обертання за допомогою Mathf.Atan2. На основі цього кута створюється Quaternion обертання Quaternion rotation, і функція плавно обертає об'єкт у напрямку до цілі за допомогою Quaternion.Slerp, враховуючи швидкість обертання _rotSpeedTarget.

Функція RotateAroundTarget відповідає за обертання об'єкта навколо цільової точки протягом певного часу. Вона також контролює перехід до наступної цілі після завершення обертання (рис. 3.14).

```
1 reference
void RotateAroundTarget()
{
    Transform target = targets[currentTargetIndex];
    rotationTimer -= Time.deltaTime;

    if (rotationTimer > 0)
    {
        float angle = rotationSpeed * Time.deltaTime;
        Vector3 offset = transform.position - target.position;
        transform.position = target.position + Quaternion.Euler(0, 0, angle) * offset;
    }
    else
    {
        isRotating = false;
        currentTargetIndex = (currentTargetIndex + 1) % targets.Count;
    }
}
```

Рисунок 3.14 – Функція RotateAroundTarget

Спочатку визначається поточна цільова позиція, навколо якої об'єкт буде обертатися. Відбувається аналогічна операція як в попередньому методі. Таймер обертання rotationTimer зменшується коли віднімається час, що пройшов з останнього кадру, використовуючи Time.deltaTime. Це забезпечує поступове зменшення таймера з кожним кадром. Якщо таймер обертання більший за нуль, об'єкт продовжує обертатися навколо цілі. Обчислюється кут обертання float angle, який залежить від швидкості обертання rotationSpeed і часу, що пройшов. Вектор offset визначає зміщення об'єкта відносно цілі. Нове положення об'єкта обчислюється шляхом застосування Quaternion обертання Quaternion.Euler(0, 0, angle) до зміщення, і це значення додається до позиції цілі target.position. Якщо таймер обертання досягає нуля або стає меншим, об'єкт припиняє обертання. Значення isRotating встановлюється на false, що сигналізує про завершення обертання.

Функція CheckDistance відповідає за обчислення пройденої відстані об'єктом і створення нових об'єктів (їжі) після подолання певної відстані (рис. 3.15).

```
void CheckDistance()
{
    distanceTraveled += Vector3.Distance(transform.position, lastSpawnPosition);

    // Спавнити новий об'єкт, якщо пройдена відстань перевищує spawnDistance
    if (distanceTraveled >= SpawnFoodDistace)
    {
        SpawnFood();
        distanceTraveled = 0f; // Скинути лічильник пройденої відстані
    }

    lastSpawnPosition = transform.position; // Оновити останню позицію спавну
}
```

Рисунок 3.15 – Функція CheckDistance

На початку функція збільшує значення distanceTraveled на відстань, яку об'єкт пройшов від останньої точки спавну, використовуючи Vector3.Distance(transform.position, lastSpawnPosition). Якщо пройдена відстань перевищує визначене значення SpawnFoodDistace, викликається функція SpawnFood для створення нового об'єкта їжі, і значення distanceTraveled скидається до нуля. Нарешті, оновлюється значення lastSpawnPosition, щоб зберегти поточну позицію об'єкта як останню точку генерації.

```
1 reference
void SpawnFood()
{
    if(MainSystemScript.foods.Count >100)
    {
        return;
    }
    GameObject go =  Instantiate(prefabFood, transform.position, Quaternion.identity);
    MainSystemScript.foods.Add(go);
}
```

Рисунок 3.16 – Функція SpawnFood

Функція SpawnFood відповідає за створення нових об'єктів їжі у грі. В першу чергу перевіряється, чи кількість об'єктів їжі в основній системі MainSystemScript.foods не перевищує 100. Якщо кількість об'єктів більше 100, функція просто повертається без спавну нових об'єктів. Якщо кількість об'єктів менша, створюється новий об'єкт їжі за допомогою префабу

prefabFood на поточній позиції об'єкта з Quaternion.identity (тобто без обертання). Новий об'єкт додається до списку foods в основній системі MainSystemScript.

Наступний клас є відносно не великим, це – CameraFollow (рис. 3.19).

```
Unity Script (1 asset reference) | 0 references
public class CameraFollow : MonoBehaviour
{
    // Start is called before the first frame update
    public Transform target; // Гравець
    public Vector3 offset; // Зміщення камери відносно гравця
    public float smoothSpeed = 0.125f; // Швидкість згладжування
    Unity Message | 0 references
    void Start()
    {

    }

    // Update is called once per frame
    Unity Message | 0 references
    void Update()
    {
        Vector3 desiredPosition = target.position + offset;
        Vector3 smoothedPosition = Vector3.Lerp(transform.position, desiredPosition, smoothSpeed);
        transform.position = smoothedPosition;

    }
}
```

Рисунок 3.17 – Функція CameraFollow

Даний клас використовується, щоб камера слідкувала постійно за об’єктом гравця. Має лише 1 працюючу функцію Update, що кожен раз відслідковує позицію гравця, та переміщує камеру в напрямку гравця.

Далі наступний клас RotateTarget, який відповідає за керуванням об’єктом гравця, та його реакцію під час зіткнення з іншими об’єктами (рис. 3.18).

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

Unity Script (1 asset reference) | 0 references
public class RotateTarget : MonoBehaviour
{
    // Start is called before the first frame update
    public float rotationSpeed;
    private Vector2 direction;

    public float moveSpeed;

    //Food
    private int Food = 0;
    //Ui
    public Text points;
    //Levels
    public GameObject[] Tentacles;
    Unity Message | 0 references
    private void Update()
    {
        direction = Camera.main.ScreenToWorldPoint(Input.mousePosition) - transform.position;
        float angle = Mathf.Atan2(direction.y, direction.x) * Mathf.Rad2Deg;
        Quaternion rotation = Quaternion.AngleAxis(angle, Vector3.forward);
        transform.rotation = Quaternion.Slerp(transform.rotation, rotation, rotationSpeed * Time.deltaTime);

        Vector2 courPos = Camera.main.ScreenToWorldPoint(Input.mousePosition);
        Vector2 newPosition = Vector2.MoveTowards(transform.position, courPos, moveSpeed * Time.deltaTime);
        newPosition.x = Mathf.Clamp(newPosition.x, -80f, 80f);
        newPosition.y = Mathf.Clamp(newPosition.y, -80f, 80f);
        transform.position = newPosition;
    }
    Unity Message | 0 references
```

Рисунок 3.18 – Клас RotateTarget

Даний клас має 3 функції: Update, OnCollisionEnter2D, CheckLevel.

Функція Update в даному коді відповідає за оновлення позиції та обертання об'єкта на кожному кадрі гри, базуючись на положенні курсора миші (рис. 3.21).

```
private void Update()
{
    direction = Camera.main.ScreenToWorldPoint(Input.mousePosition) - transform.position;
    float angle = Mathf.Atan2(direction.y, direction.x) * Mathf.Rad2Deg;
    Quaternion rotation = Quaternion.AngleAxis(angle, Vector3.forward);
    transform.rotation = Quaternion.Slerp(transform.rotation, rotation, rotationSpeed * Time.deltaTime);

    Vector2 courPos = Camera.main.ScreenToWorldPoint(Input.mousePosition);
    Vector2 newPosition = Vector2.MoveTowards(transform.position, courPos, moveSpeed * Time.deltaTime);
    newPosition.x = Mathf.Clamp(newPosition.x, -80f, 80f);
    newPosition.y = Mathf.Clamp(newPosition.y, -80f, 80f);
    transform.position = newPosition;
}
```

Рисунок 3.19 – Функція Update

Функція обчислює напрямок до курсора миші від поточної позиції об'єкта. Для цього використовується наступна функція: Camera.main.ScreenToWorldPoint(Input.mousePosition). Вектор напряму використовується для обчислення кута обертання в радіанах за допомогою Mathf.Atan2(direction.y, direction.x). Об'єкт плавно обертається до нової позиції

курсора за допомогою Quaternion.Slerp(transform.rotation, rotation, rotationSpeed * Time.deltaTime), де rotationSpeed визначає швидкість обертання. Далі використовуючи функцію Vector2.MoveTowards, об'єкт переміщується до нової позиції, визначеної курсором миші, з заданою швидкістю moveSpeed. Це забезпечує плавний рух об'єкта до курсора.

Наступна функція OnCollisionEnter2D. Функція відповідає за обробку зіткнень об'єкта з іншими об'єктами в двовимірному просторі (2D). Вона реагує на зіткнення з об'єктами, що мають тег "Food" (рис. 3.20).

```
Unity Message | 0 references
private void OnCollisionEnter2D(Collision2D collision)
{
    if(collision.gameObject.tag == "Food")
    {
        Debug.Log("З'їв");
        Food++;
        points.text = Food.ToString();
        Destroy(collision.gameObject);
        for (int i = MainSystemScript.foods.Count - 1; i >= 0; i--)
        {
            if (MainSystemScript.foods[i] == null)
            {
                MainSystemScript.foods.RemoveAt(i);
            }
        }
        CheckLevel();
    }
}
```

Рисунок 3.20 – Функція OnCollisionEnter2D

Функція перевіряє тег об'єкта, з яким сталося зіткнення, використовуючи collision.gameObject.tag == "Food". Збільшується значення змінної Food на одиницю, що відображає кількість зібраної їжі. Оновлюється текстовий елемент points, щоб відобразити нове значення лічильника їжі, використовуючи points.text = Food.ToString().Об'єкт їжі, з яким сталося зіткнення, видаляється з сцени за допомогою Destroy(collision.gameObject). Після видалення, також очищається список **foods.**

Функція CheckLevel відповідає за перевірку кількості зібраної їжі (значення Food) і зміну розміру об'єкта, а також активацію додаткових елементів (щупалець) залежно від рівня їжі (рис. 3.21).

```
1 reference
void CheckLevel()
{
    if(Food > 10 && Food < 50)
    {
        gameObject.transform.localScale = new Vector3(0.35f, 0.35f, 0);

    }
    if(Food > 50 && Food < 100)
    {
        gameObject.transform.localScale = new Vector3(0.36f, 0.36f, 0);
        Tentacles[1].SetActive(true);
    }
    if (Food >100)
    {
        gameObject.transform.localScale = new Vector3(0.37f, 0.37f, 0);
        Tentacles[2].SetActive(true);
    }
}
```

Рисунок 3.21 – Функція CheckLevel

Наступний невеликий клас RotateWithoutMouth. Виконує тільки обертання об’єкту відносно об’єкту гравця (рис. 3.22).

```
Unity Script (1 asset reference) | 0 references
public class RotateWithoutMouth : MonoBehaviour
{
    // Start is called before the first frame update
    public float rotationSpeed;
    public Transform target;
    private Vector2 direction;

    public float moveSpeed;
    Unity Message | 0 references
    private void Update()
    {
        direction = target.position - transform.position;
        float angle = Mathf.Atan2(direction.y, direction.x) * Mathf.Rad2Deg;
        Quaternion rotation = Quaternion.AngleAxis(angle, Vector3.forward);
        transform.rotation = Quaternion.Slerp(transform.rotation, rotation, rotationSpeed * Time.deltaTime);

    }
}
```

Рисунок 3.22 – Клас RotateWithoutMouth

Для створення процедурної анімації використовується клас Tentacle. Клас Tentacle в Unity відповідає за створення та анімацію щупальця, яке

складається з кількох сегментів. Клас використовує компонент LineRenderer для візуалізації щупальця як лінії з сегментів, які плавно рухаються та слідують за цільовою точкою (рис. 3.23).

```
Unity Script (38 asset references) | 0 references
public class Tentacle : MonoBehaviour
{
    public int length;
    public LineRenderer lineRenderer;
    public Vector3[] segmPoses;
    private Vector3[] segmentV;
    // Start is called before the first frame update
    public Transform tragetPoint;
    public float TargetDist;
    public float SmothSpeed;
    public float TrailSpeed;

    public float wigleSpeed;
    public float wigleMagnitude;
    public Transform wigleGM;
    Unity Message | 0 references
    void Start()
    {
        lineRenderer.positionCount = length;
        segmPoses = new Vector3[length];
        segmentV = new Vector3[length];
    }

```

Рисунок 3.23 – Клас Tentacle

Поле length визначає кількість сегментів щупальця. Це значення використовується для встановлення кількості позицій у LineRenderer та створення масиву позицій і векторів сегментів. LineRenderer відповідає за рендеринг щупальця як лінії. Масив segmPoses зберігає позиції всіх сегментів щупальця. Початкова позиція задається як позиція об'єкта, до якого прив'язане щупальце.

Метод Start виконується один раз при запуску. Він ініціалізує кількість позицій у LineRenderer відповідно до значення length. Також створюються масиви segmPoses та segmentV (рис. 3.24).

```
Unity Message | 0 references
void Start()
{
    lineRenderer.positionCount = length;
    segmPoses = new Vector3[length];
    segmentV = new Vector3[length];
}
```

Рисунок 3.24 – Функція Start

Метод Update викликається на кожному кадрі гри. Спочатку він обчислює обертання щупальця для створення ефекту коливання, використовуючи wigleSpeed і wigleMagnitude. Перший сегмент щупальця встановлюється в позицію об'єкта, до якого прив'язане щупальце. Далі кожен сегмент щупальця плавно рухається до нової позиції, використовуючи функцію Vector3.SmoothDamp, яка забезпечує згладжений рух сегментів з урахуванням параметрів SmothSpeed і TrailSpeed. Кінцеві позиції всіх сегментів встановлюються у LineRenderer для візуалізації щупальця (рис. 3.25).

```
Unity Message | 0 references
void Update()
{
    wigleRot localRotation = Quaternion.Euler(0, 0, Mathf.Sin(Time.time * wigleSpeed) * wigleMagnitude);
    segmPoses[0] = transform.position;
    for(int i = 1; i < segmPoses.Length; i++)
    {
        segmPoses[i] = Vector3.SmoothDamp(segmPoses[i], segmPoses[i - 1] + targetPoint.right * targetDist, ref segmentV[i], SmothSpeed+i/trailSpeed);
    }
    lineRenderer.SetPositions(segmPoses);
}
```

Рисунок 3.25 – Функція Update

Наступний крок, це розробка самої сцени разом з об’єктами та встановлення на них відповідних скриптів (рис. 3.26).

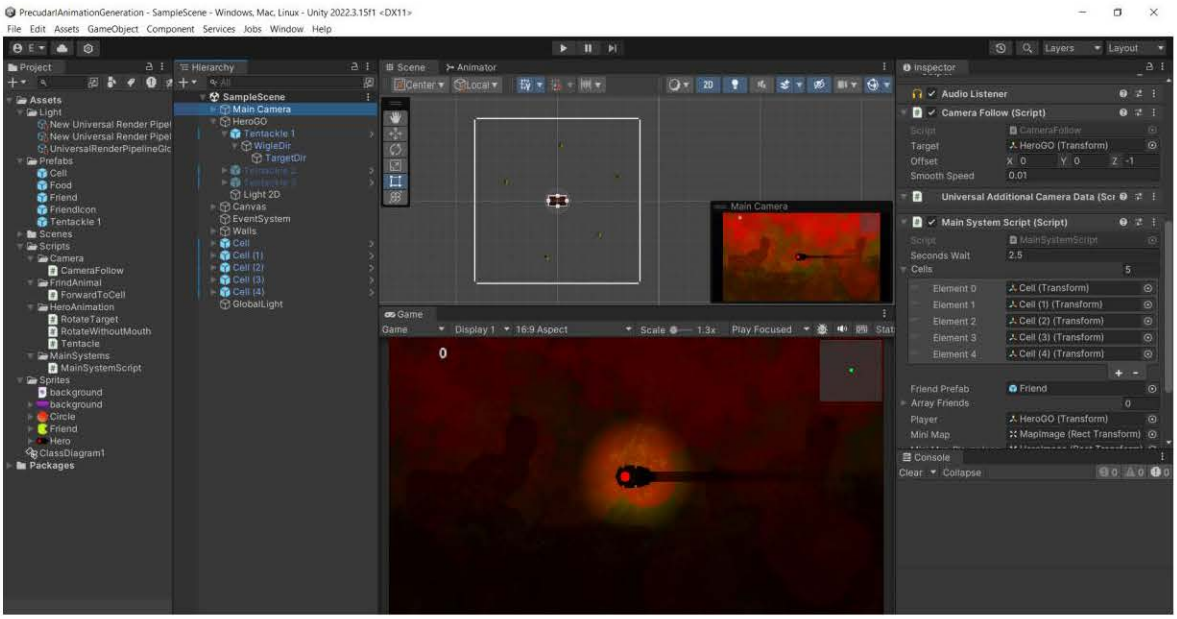

Рисунок 3.26 – Робочий інтерфейс Unity

Потрібно розпочати з Камери даної сцени, адже через неї гравець бачить всі процеси та анімації (рис. 3.27):

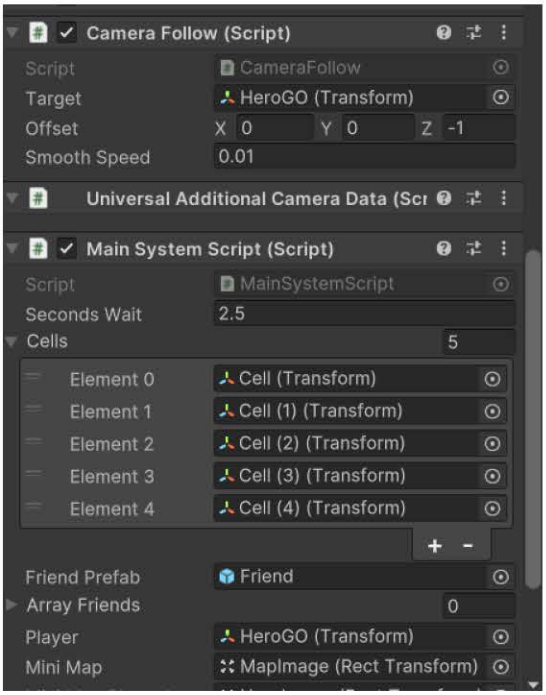

Рисунок 3.27 – Скрипти, що має об’єкт Camera

Зазвичай при розробці ігрових додатків, найважливіші скрипти, що відповідають за ігрову систему прикріплюються до камери, тому що вона

гарантовано залишиться на сцені. Без неї неможливо відобразити та редерити певне зображення чи текстури.

Для відображення заднього фону, яке завжди відображається позаду сцени не залежно від місця положення гравця, використовується Canvas (рис. 3.28):

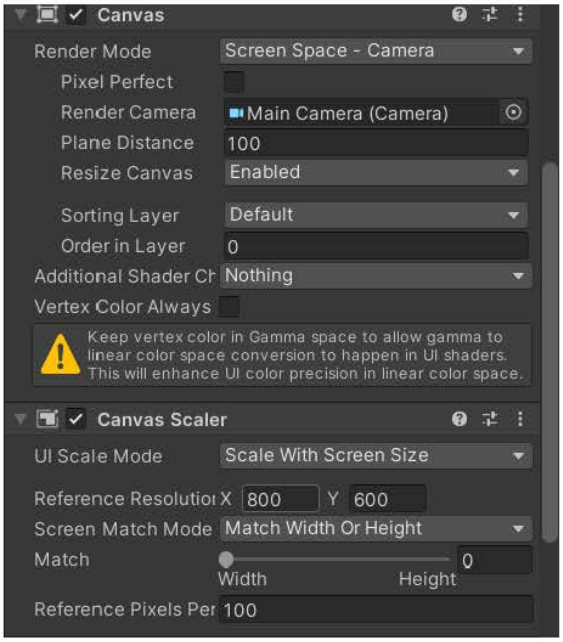

Рисунок 3.28 – Об’єкт Canvas та його компоненти

Налаштування Screen Space – Camera, дозволяють прикріпити даний об’єкт за камерою, і кожен раз коли камера буде переміщуватися, задній фон буде рухатися разом з ним.

Canvas дозволяє додавати різні елементи інтерфейсу починаючи від тексту та кнопок до різноманітних слайдерів які можна налаштовувати для властних потреб. В даному проекті є лічильник, який відображає кількість зібраної їжі за допомогою об’єкту Text (рис. 3.29).

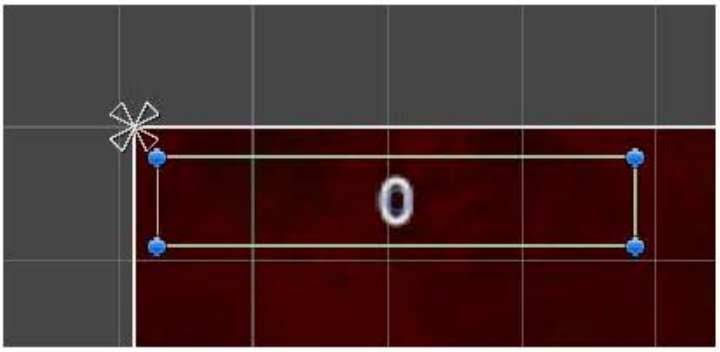

Рисунок 3.29 – об’єкт Text

Крім лічильника також є міні-карта, яка зображає положення гравця та інших об’єктів, що рухаються (рис. 3.30):

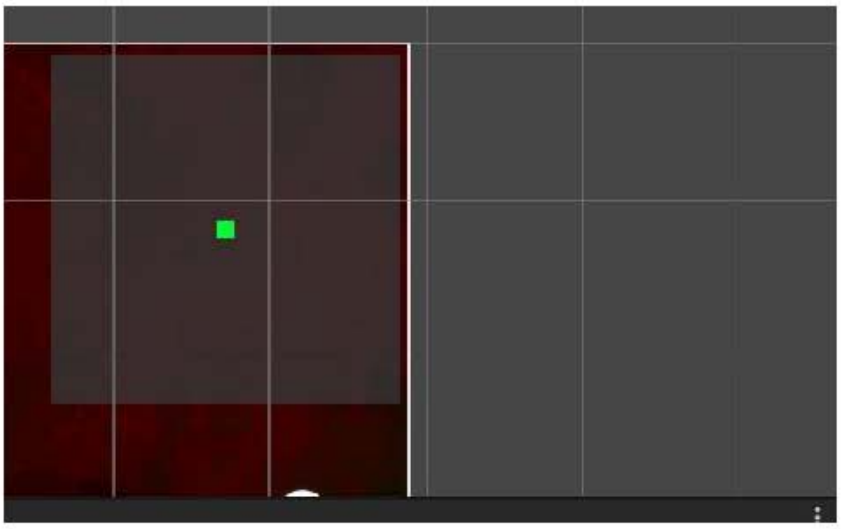

Рисунок 3.30 – Міні-карат

Далі об’єкт, яким керує сам гравець (рис. 3.31).

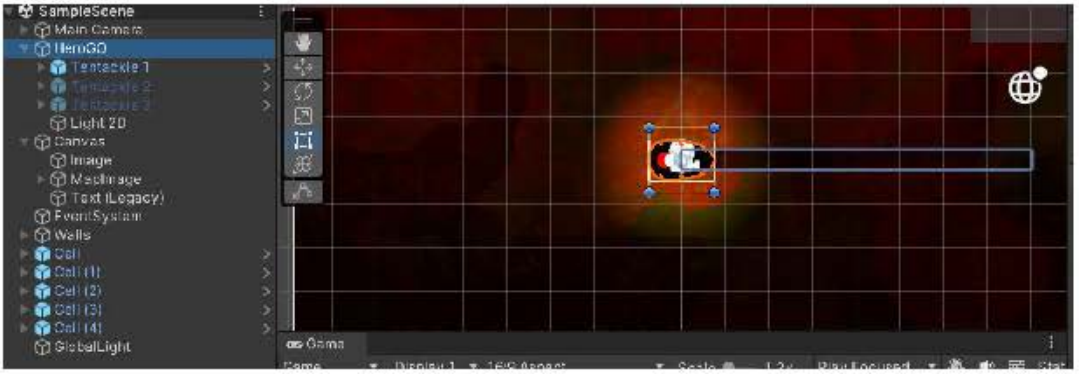

Рисунок 3.31 – Об’єкт HeroGO

Даний об’єкт переміщується за допомогою прикріпленого скрипта RotateTarget, за курсором миші. Даний об’єкт має декілька дочірніх об’єктів, що будуть активуватися при зібраній певній кількості їжі:

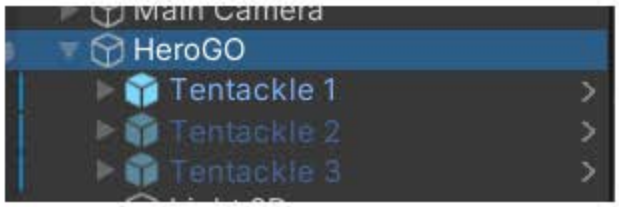

Рисунок 3.32 – Дочірні об’єкти HeroGO

Об’єктом анімації виступає Tentackle, цей об’єкт представляє собою LineRenderer, яким керується також відповідний прикріплений скрипт, як і назва об’єкту (рис. 3.33, 3.34):

Рисунок 3.33 – Об’єкт Tentackle.

Рисунок 3.34 - Об’єкт Tentackle (зовнішній вигляд)

Наступний об’єкт Cell, на сцені їхня кількість становить 5 таких об’єктів. Використовуються як цільові точки для переміщення об’єктів friends (рис. 3.35).

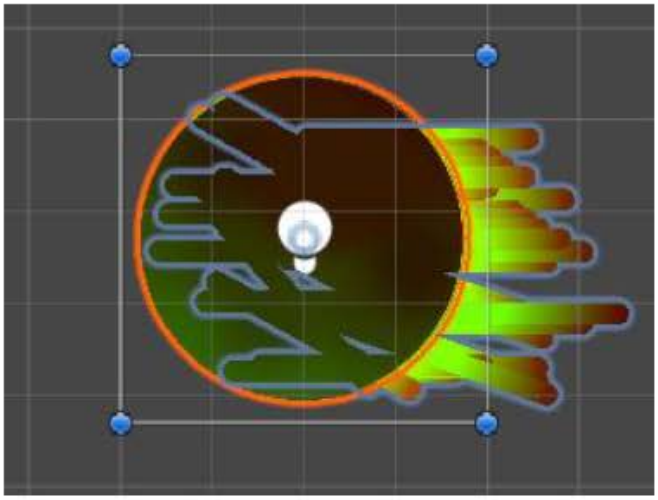

Рисунок 3.35 – Об’єкт Cell

Останніми об’єктами є FriendPrefab, що підчас руху створюють їжу для гравця (рис. 3.36):

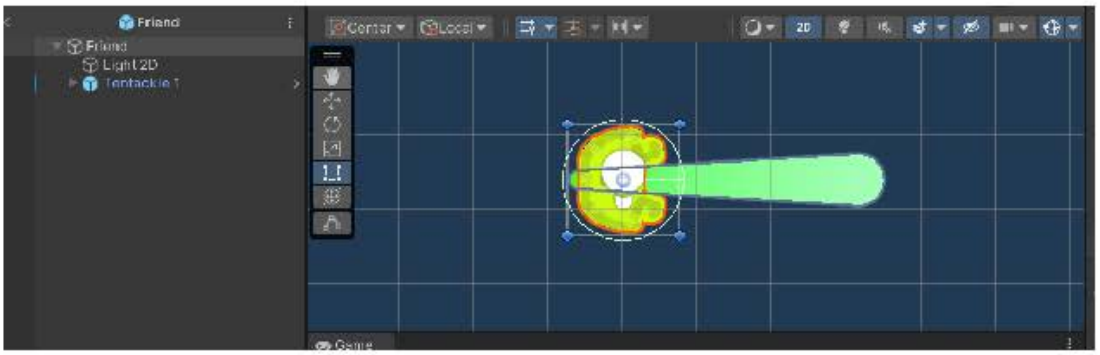

Рисунок 3.36 – Об’єкт Friend

3.5 Тестування проекту

Спочатку потрібно зібрати проект, щоб його можна було відкрити на будь-якому комп'ютері, для цього потрібно відкрити вікно File/Build Settings (рис. 3.37):

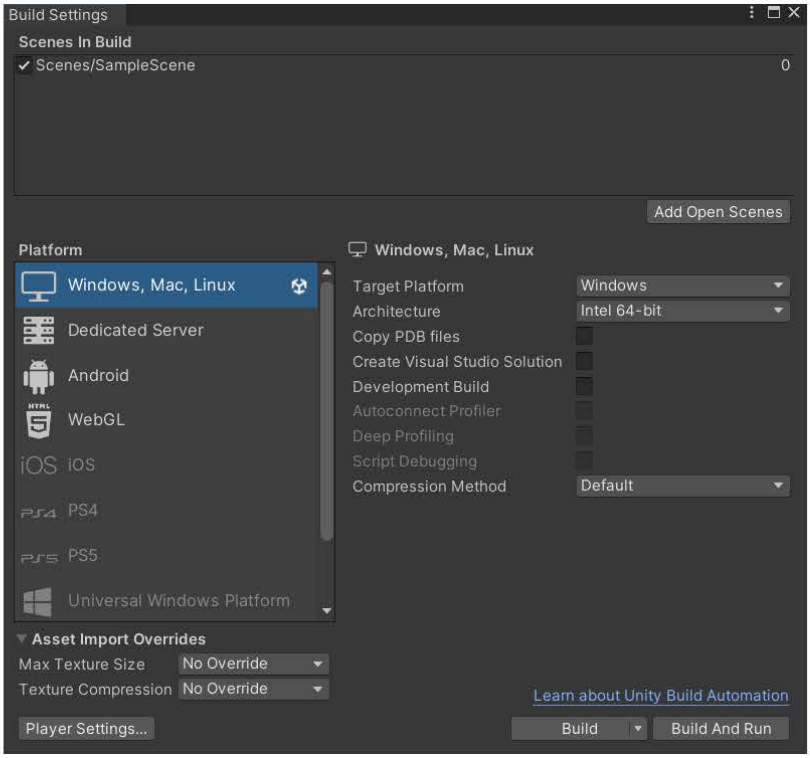

Рисунок 3.37 – Вікно для створення збірки проекту

Після збірки проекту потрібно відкрити проект у папці, та знайти .exe файл, та відкрити його:

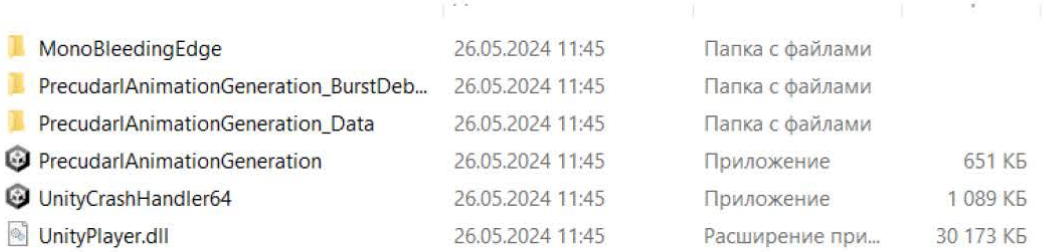

Рисунок 3.38 – Папка з проектом

Після відкриття файлу, користувач повинен бачити наступне вікно з грою (рис 3.39):

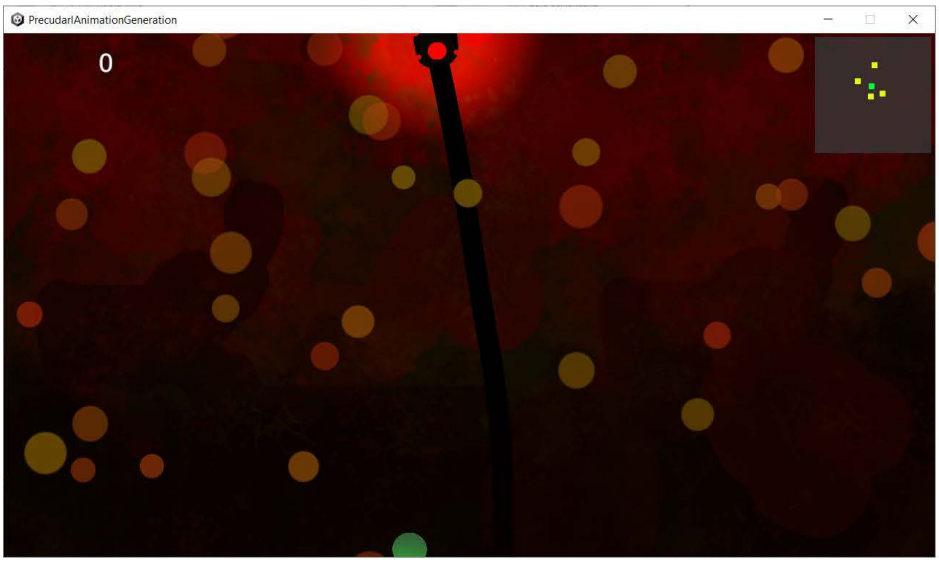

Рисунок 3.39 – Запуск проекту

Після запуску проекту користувач може за допомогою курсору миші керувати напрямком руху ігрового персонажу. Далі в процесі гри користувач може відслідковувати переміщення інших об’єктів та переміщатися разом з ними (рис. 3.40):

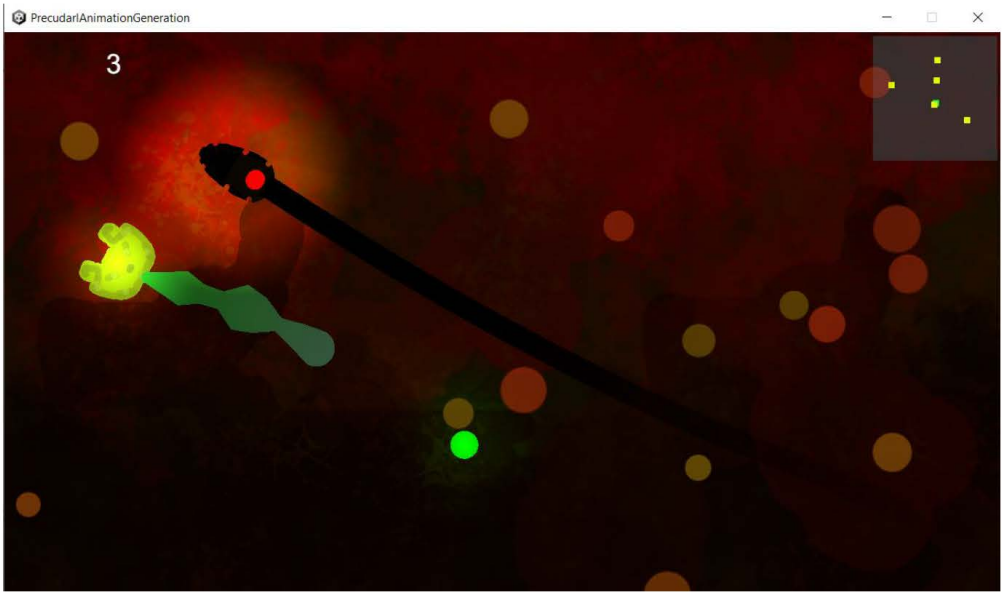

Рисунок 3.40 – Зміни положень об’єктів на міні-карті

Також підчас збору їжі користувач, повинен бачити зміни ігрового персонажу. Спочатку об’єкт буде збільшуватися в розмірах, як тільки досягне позначки більше 10 (рис. 3.41):

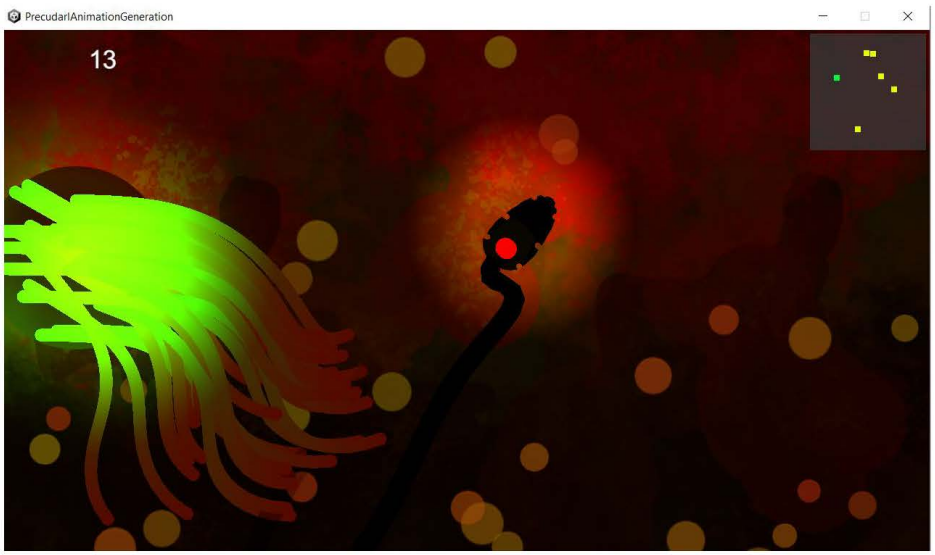

Рисунок 3.41 – Зміни зовнішнього вигляду ігрового персонажу

Далі, якщо користувач отримав більше 50, то з’явиться нова частина тіла в ігрового персонажа (рис. 3.42):

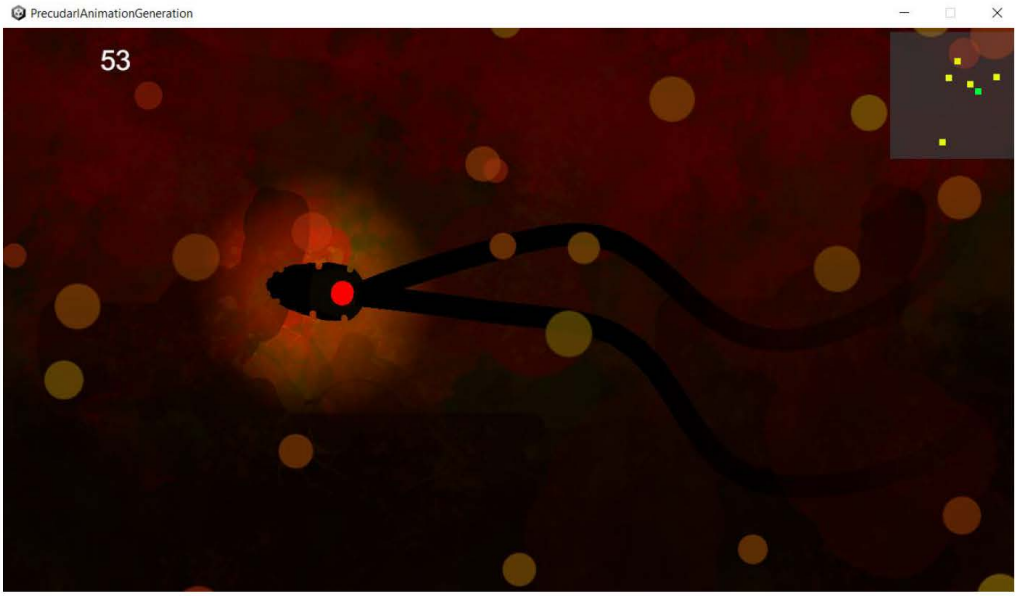

Рисунок 3.42 – Новий зовнішній вигляд ігрового персонажу

Останні зміни користувач спостерігатиме коли досягне відмітки більше 100 (рис. 3.43):

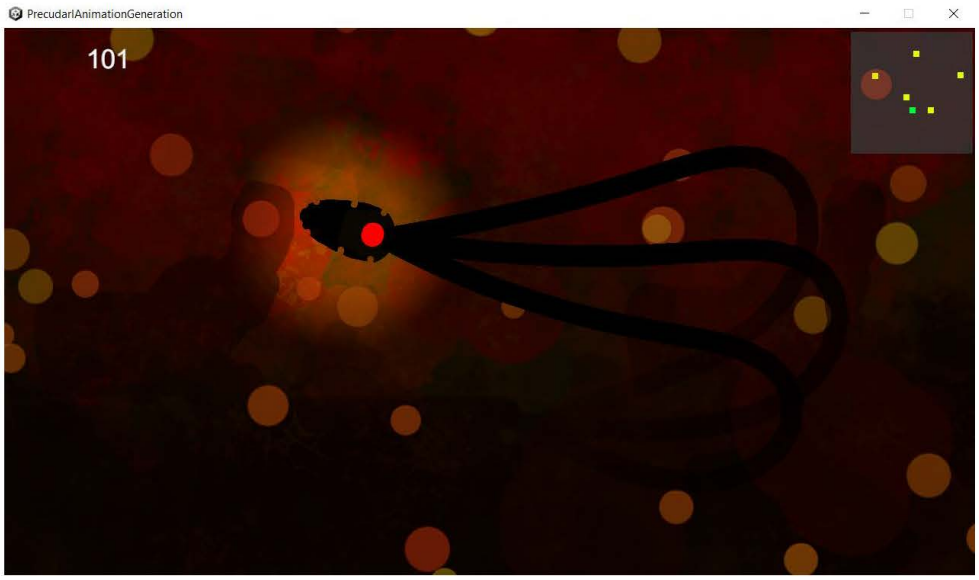

Рисунок 3.43 – Останні зовнішні зміни ігрового об’єкту

З попередніх ілюстрацій можна визначити також, що окрім об’єктів, що створюють їжу, є статичні об’єкти які не рухаються але теж реагують на гравцю обертаючись до нього. Біля цих об’єктів, створюється найбільша кількість їжі, яку гравець може зібрати (рис. 3.44):

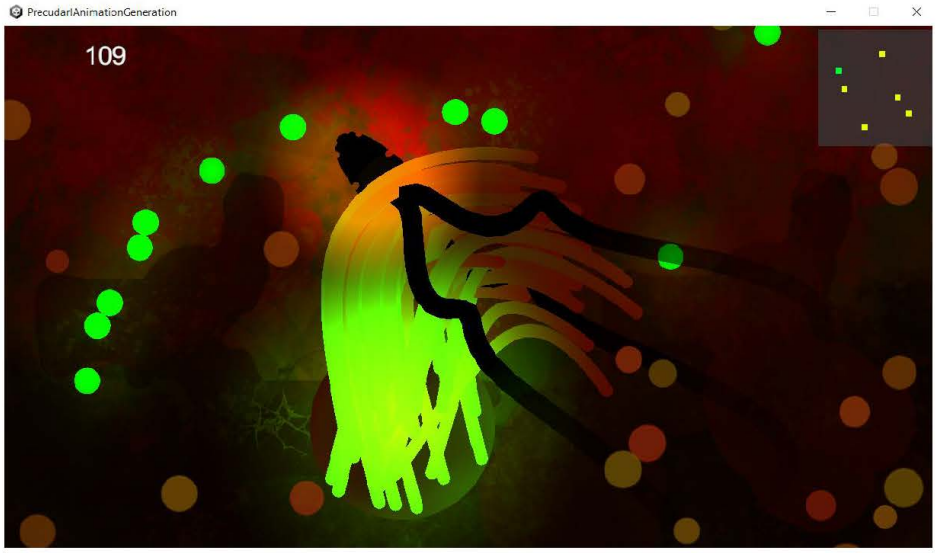

Рисунок 3.44 – Статичні об’єкти, що обертаються до гравця

3.6 Висновок до третього розділу

Проект з розробки процедурної анімації в Unity успішно демонструє ефективне використання можливостей Unity для створення динамічних і інтерактивних світів.

У проекті було реалізовано процедурну анімацію об'єкту, яка включає плавний рух сегментів, створення ефекту коливання та динамічне змінювання розміру та вигляду залежно від кількості зібраної їжі. Анімація забезпечує реалістичний та привабливий вигляд, що підвищує загальний візуальний досвід гри. Реалізовані функції забезпечують інтерактивність, динамічність та гнучкість, що дозволяє створювати реалістичні анімації, які реагують на дії користувача та зміни у ігровому середовищі. Завдяки цьому, проект може служити хорошою основою для подальшого розвитку ігрових механік та анімаційних систем.

ВИСНОВОК

Кваліфікаційна робота присвячена розробці 2D гри з процедурною анімацією, що є актуальним завданням у контексті сучасного розвитку індустрії відеоігор. Метою роботи було створення ефективного інструменту для реалізації динамічних та інтерактивних ігрових середовищ, що сприятиме підвищенню якості ігрового досвіду.

У роботі було проведено детальний аналіз предметної області, зокрема, дослідження сучасних методів та підходів до розробки ігрових додатків. Особливу увагу було приділено аналізу існуючих рішень, що дозволило визначити ключові тенденції та технології, які використовуються для створення 2D ігор. Було встановлено, що вибір правильних інструментів та технологій є критичним для успішної реалізації проекту, оскільки це забезпечує гнучкість та продуктивність розробки.

Процедурна анімація забезпечує високу гнучкість і адаптивність. Алгоритмічний підхід дозволяє легко змінювати параметри анімації, що важливо для створення інтерактивних середовищ у реальному часі, що використовуються у відеоіграх і віртуальній реальності. Це відкриває нові можливості для креативності, дозволяючи художникам і розробникам експериментувати з різними варіантами анімації без необхідності створювати нові кадри з нуля.

Процедурна анімація також сприяє створенню більш реалістичних і фізично правдоподібних ефектів. Використання фізичних симуляцій та математичних моделей дозволяє створювати природні явища, такі як вода, вогонь, дим та інші. Це не тільки підвищує загальну якість візуальних ефектів, але й допомагає створювати більш захоплюючі та реалістичні віртуальні світи.

У процесі дослідження було розглянуто різні методи розробки ігрових додатків, включаючи водоспадний метод, гнучкі методи (Agile), спіральний метод та прототипування. Водоспадний метод забезпечує чітку послідовність етапів розробки, але є менш гнучким у порівнянні з Agile, який дозволяє

швидко реагувати на зміни у вимогах. Спіральний метод поєднує переваги обох підходів, акцентуючи увагу на управлінні ризиками. Прототипування дозволяє швидко оцінити життєздатність ігрових концепцій на ранніх етапах розробки, що допомагає уникнути великих витрат на неперспективні ідеї.

Важливим аспектом розробки ігрових додатків є вибір ігрового рушія. Сучасні рушії, такі як Unity, Unreal Engine, Godot, Phaser та GameMaker Studio 2, надають розробникам широкий спектр інструментів для створення високоякісних ігор. Кожен з них має свої переваги та недоліки, що дозволяє обирати оптимальний рушій залежно від специфіки проекту.

Було проведено аналіз засобів реалізації 2D гри, що включає вибір програмних інструментів, які найкраще відповідають вимогам проекту. Основним акцентом аналізу став вибір ігрового рушія та графічних інструментів, які забезпечать високий рівень продуктивності та якісної графіки. Було обрано оптимальні програмні засоби для реалізації проекту. Для розробки гри було обрано ігровий рушій Unity та програму для роботи з графікою Krita. Використання Unity забезпечило потужні можливості для створення складних анімацій та інтерактивних елементів. Його переваги включають широку функціональність, підтримку крос-платформенності, а також активну спільноту користувачів, що забезпечує доступ до великої кількості ресурсів та додаткових інструментів. Krita дозволила створювати високоякісні графічні ресурси. Krita було вибрано завдяки її безкоштовності та потужним можливостям для цифрового малювання та анімації, що дозволяє створювати високоякісні графічні ресурси для гри. Photoshop залишився у резерві як додатковий інструмент для редагування та доопрацювання графічних матеріалів.

Описано процес розробки гри з процедурною анімацією. Реалізовано основні компоненти, такі як алгоритми процедурної анімації, створення ігрових об'єктів та їх взаємодія. Було розроблено та протестовано ключові елементи гри, включаючи рух ігрового персонажу, генерацію об'єктів та

інтерактивні анімації, що забезпечило плавність та реалістичність ігрового процесу.

Проведене тестування проекту підтвердило його функціональність та відповідність вимогам. Результати тестування показали, що гра ефективно обробляє взаємодію між об'єктами, забезпечує надійне зберігання даних та пропонує користувачам захоплюючий ігровий досвід.

Розроблений проект демонструє високу ефективність та відповідає сучасним вимогам індустрії відеоігор. Використання процедурної анімації дозволило досягти високої гнучкості та адаптивності коду, що полегшує його подальшу підтримку та розвиток. Результати роботи можуть бути корисними для інших розробників та дослідників у сфері ігрової індустрії, які прагнуть створювати інноваційні продукти, що відповідають сучасним тенденціям та очікуванням гравців.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Unity 3D [Електронний ресурс] – Режим доступу: https://docs.unity.com

2. Unity for Games. Unity. [Електронний ресурс] – Режим доступу: https://unity.com

3. Scripting in Unity. Unity Learn. [Електронний ресурс] – Режим доступу: https://learn.unity.com/course/create-with-code

4. Майк Гейг. Розробка ігор на Unity за 24 години, 2020. 464 с.

5. Джо Хокінг. Unity в дії: Розробка мультиплатформних ігор на C#, 2015. 352 с.

6. Алан Торн. Опанування Unity Scripting, 2015. 380 с.

7. Джеремі Гібсон Бонд. Unity і C#. Геймдев від ідеї до реалізації, третє видання, 2022. 944 с.

8. Алекс Окіта. Вивчаємо програмування на C# з Unity 3D, друге видання, 2019. 690 с.

9. C# і .NET | Властивості. METANIT.COM [Електронний ресурс] – Режим доступу: https://metanit.com/sharp/tutorial/3.4.

10. Phaser API Documentation [Електронний ресурс] – Режим доступу: https://newdocs.phaser.io/docs/3.80.0

11. Що таке UML-діаграми? [Електронний ресурс] – Режим доступу: https://evergreens.com.ua/ua/articles/uml-diagrams.html

12. License - Godot Engine. [Електронний ресурс] – Режим доступу: https://godotengine.org/license/.

13. Photon Engine: Multiplayer Game Development Made Easy [Електронний ресурс] – Режим доступу: https://www.photonengine.com/pun.

14. Short T., Adams T. Procedural Generation in Game Design. 2017. 313p.

15. Unreal Engine 5.4 Documentation [Електронний ресурс] – Режим доступу: https://dev.epicgames.com/documentation/en-us/unreal-engine/unreal-engine-5-4-documentation