ОДЕСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ І. І. МЕЧНИКОВА

(повне найменування закладу вищої освіти)

Факультет математики, фізики та інформаційних технологій

(повне найменування факультету)

Кафедра інформаційних технологій

(повна назва кафедри)

Кваліфікаційна робота

на здобуття ступеня вищої освіти «Бакалавр»

«Розробка 2Д-гри на рушії Unity»

(тема кваліфікаційної роботи українською мовою)

«Development of a 2D game on the Unity engine»

(тема кваліфікаційної роботи англійською мовою)

Виконав: здобувач заочної форми навчання

спеціальності 122 Комп'ютерні науки

(код, назва спеціальності)

Освітня програма Комп'ютерні науки

(назва)

Кузнецов Роберт Германович

(прізвище, ім'я, по-батькові здобувача)

Керівник ст. викладач Штефан Н.З. _____

(науковий ступінь, вчене звання, прізвище, ініціали) (підпис)

Рецензент к.т.н., доцент Перелигін Б.В.

(науковий ступінь, вчене звання, прізвище, ініціали)

Рекомендовано до захисту:
Протокол засідання кафедри
Інформаційних технологій .
№ 1 від 09 червня 2024 р.

Завідувачка кафедри

_____ КАЗАКОВА Надія .

(підпис) (прізвище,ім'я)

Захищено на засіданні ЕК № 13 .
протокол № 3 від 19 червня 2024 р.

Оцінка добре / С / 75 .

(за національною шкалою/шкалою ECTS/ бали)

Голова ЕК

_____ КОПИЧЕНКО Іван .

(підпис) (прізвище,ім'я)

Одеса 2024

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ ТА ТЕРМІНІВ

HUD – head-up display

WBS – Work Breakdown Structure

Ассети – ігровий ресурс, цифровий об'єкт, який переважно складається з однотипних даних

Спрайти – двомірне зображення, що застосовується в комп'ютерній графіці

Платформери – жанр комп'ютерних ігор

C# – мова програмування

State Machine Diagram – діаграма станів

Unity – кроссплатформне середовище розробки комп'ютерних ігор

Unreal Engine – ігровий рущій

Use Case Diagram – діаграма варіантів використання

ВСТУП

Історія двовимірних ігор починається в 1970-х роках з появою перших аркад і домашньої консолі. Приблизно наприкінці 90-х і на початку 2000-х 2D сприймався як сходинка до 3D і майже вимер. Усі найцінніші франшизи відеоігор перейшли на 3D. 3D домінував на домашніх консолях, замінюючи 2D на кишенькових комп'ютерах. Так, 2D-ігри не зникли повністю. Але коли їх перевели на кишенькові комп'ютери, вони відчули себе менш важливими.

Їхнє відродження почалося з запуску Xbox Live Arcade на оригінальній Xbox у 2001 році. Тут розміщувалися класичні аркадні та деякі прості ігри. Xbox Live Arcade став живильним середовищем для невеликих ігор і створив місце для 2D-ігор у стилі ретро на домашній консолі. Тепер двовимірні ігри можуть співіснувати з високобюджетними 3D.

Інді-гра – це не жанр, клас чи навіть тип гри. Інді-гра означає гру, створену незалежною командою або навіть одним розробником. Це гра без великої фінансової підтримки, але зі свободою та сміливістю поставити ідею гри вище прибутку (ті ж мобільні ігри набагато прибутковіші).

Деякі з найбільших успіхів на ринку 2D-відеоігор досягли незалежних розробників ігор. Через нижчий рівень складності розробники ігор витрачають менше часу та грошей на створення 2D-ігор. Розробники можуть більше зосередитися на експериментуванні з різними сюжетними лініями та художніми стилями, а також легше виконувати оновлення. Для ефективнішого виробничого процесу незалежні розробники залучають 2D концепт-арт персонажів та інші виробничі процеси [1].

Мета – розробка захоплюючого, цікавого та інтуїтивно зрозумілого ігрового процесу з викликами та головоломками, які сприятимуть підтримки зацікавленості гравців.

Для реалізації поставленої мети слід виконати наступні завдання:

1. Проаналізувати предметну область та зробити аналітичний огляд існуючих аналогів.

2. Визначити вимоги до ігрового застосунку.
3. Створити ігровий світ та персонажів.
4. Розробити ігрові механіки.
5. Реалізувати ігровий застосунок обраними засобами розробки.
6. Провести тестування та виправити помилки.

1 СПЕЦІФІКАЦІЯ ВИМОГ ДО ОБ'ЄКТУ РОЗРОБКИ

1.1 Характеристика предметної області

Двовимірні ігри пройшли довгий шлях, і тепер вони популярні, ніж будь-коли з моменту їх появи. Вони трохи зрозуміліші та коротші, ніж 3D-ігри, тому завжди знайдуть свою аудиторію. 2D-ігри довели, що деякі жанри ніколи не вмирають. Їх можна переродити, впровадивши нові технології та графіку.

Space Invaders у 1978 році зіграли важливу роль у перетворенні відеоігор на глобальну індустрію. Це змусило десятки виробників помітити і вийти на ринок відеоігор. Понг, настільний теніс і Atari Football були одними з перших спортивних ігор. Тетріс популяризував жанр ігор-головоломок, особливо на кишенькових комп'ютерах. Super Mario Bros. 1985 популяризував ігри на платформі зі скроллінгом. Це, безперечно, одна з найкращих 2D-ігор усіх часів. Легенда про Зельду 1986 представила пригодницький бойовик. Це був початок серії ігор, яка отримала найбільше визнання критиків. SimCity 1989 став піонером у жанрі симуляторів і відкритих типів ігор.

Двомірні ігри можуть відрізнятися художнім стилем, жанром, перспективним виглядом і платформою. Жанр відеоігор – це категорія ігор, яка має схожі ігрові характеристики. Жанри відеоігор майже однакові як для 2D, так і для 3D ігор. Це різні види платформних ігор, рольові ігри, екшн-ігри, файтинги, симулятори, стрілялки, головоломки тощо.

Але ключовою відмінністю 2D-ігор є вибір перспективи. Ось найпоширеніші типи 2D-ігор на основі їх перспективи:

1. Вид збоку. Це один із найпопулярніших видів 2D-ігор. Вигляд збоку є класичним вибором для платформерів, коли персонажі рухаються переважно зліва направо, вгору та вниз.
2. Вигляд зверху вниз. Гра виглядає так, ніби камера знаходиться над головою. Ігрове поле або видно з висоти пташиного польоту, або має трохи нахилену камеру.

3. Ізометрична перспектива. Ігрове поле показано під певним кутом камери. Він ідеально підходить для створення ілюзії тривимірного простору, що показує три сторони об’єкта.
4. Один екран. Кожен рівень розташований у новій кімнаті, яка заповнює весь простір екрана. Коли рівень пройдено, ви переходите до наступної кімнати [2].

Основні механіки та характеристики для платформерів:

1) Moveset.

Набір ходів – це туманний термін, який охоплює всі здібності та властивості персонажа гравця. До них входять стандартний біг і стрибок, а також інші механізми, такі як ковзання, і правила різноманітної поведінки, наприклад, скільки часу потрібно, щоб розігнатися до максимальної швидкості, як персонаж реагує на удар тощо. Набір ходів має бути дуже чітким і точним. Немає нічого гіршого, ніж наступити на колекційний предмет і не підібрати його.

2) Рівні.

Здебільшого на естетичному рівні дуже допомагає, якщо гра складається з різних зон, кожна з яких має свій унікальний вигляд. Звичайно, ця унікальність часто супроводжується численними інтерактивними об’єктами, які додають різноманітності та допомагають у темпі, але є одна маленька деталь, яка час від часу пропадає в очі: поділ переднього та заднього планів [3].

3) Стани: якщо розглянути приклад платформера, де гемплей розгортається у підземеллі, то зазвичай частини стель, які висять у повітрі, можуть руйнуватися при попаданні снаряду або коли ігровой юніт торкається їх поверхні під час стрибку. Елементарна анімація руйнування таких об’єктів дає гравцю у відчуття живої картинки ігрового процесу. Також, до станів можна віднестти маленьке колихання ліан чи рослин, що прикрашають ігровий світ.

1.2 Огляд аналогів

Огляд існуючих аналогів 2D платформерів перед початком розробки власного проекту може значно покращити якість та успіх нового проекту. Нижче наведено основні аспекти, в яких цей аналіз може бути корисним:

1. Розуміння ринку:
   - визначення популярних жанрів,и а саме аналіз успішних ігор допоможе зрозуміти, які жанри та стилі користуються популярністю серед гравців;
   - цільова аудиторія: хто вона та які її уподобання.
2. Вивчення механік гри:
   - ознайомлення з різноманітними ігровими механіками, що вже існують, і визначення, які з них можна використати у своїй грі або які можна покращити.
   - виявлення унікальних елементів, які виділяють інші ігри, та використання їх як натхнення для створення власних інноваційних рішень.
3. Аналіз графіки та дизайну:
   - вивчення різних візуальних стилів та їх впливу на гравців допоможе створити привабливий дизайн свого проекту;
   - інтерфейс користувача: аналіз ui/ux інших ігор для покращення зручності використання вашої гри.
4. Ігровий процес та баланс:
   - складність рівнів: вивчення рівнів складності та структури рівнів, що використовуються в інших іграх, допоможе створити збалансований та захоплюючий геймплей;
   - прогресія гри: аналіз того, як інші ігри керують прогресією гравця через рівні та виклики.
5. Монетизація та бізнес-моделі:

- методи монетизації: ознайомлення з різними моделями монетизації (внутрішньоігрові покупки, реклами, платні версії) для вибору оптимального підходу;
- цінова політика: аналіз цінових стратегій, що використовуються в успішних іграх, допоможе визначити найкращу стратегію для вашого проекту.

6. Технічні аспекти:
   - технологічні рішення: вивчення технічних рішень, використаних в інших іграх, таких як рушії, бібліотеки та фреймворки;
   - оптимізація: аналіз методів оптимізації для забезпечення плавної роботи гри на різних платформах.
7. Маркетинг та просування:
   - стратегії просування: вивчення маркетингових стратегій, що використовуються іншими розробниками, допоможе в розробці власного плану просування гри;
   - аналіз відгуків гравців на інші ігри дозволить зрозуміти, що саме їм подобається або не подобається, та уникнути аналогічних помилок.
8. Вдосконалення та інновації:
   - виявлення недоліків: вивчення негативних аспектів інших ігор для уникнення подібних помилок у вашому проекті;
   - інновації: використання отриманих знань для впровадження нових та інноваційних рішень, які зроблять вашу гру унікальною та привабливою.

Першим аналогом для аналізу було обрано «Sonic Mania» (2017).

«Sonic Mania» була створена як данина класичним 2D платформерам серії Sonic the Hedgehog з 1990-х років, зокрема іграм на консолі «Sega Genesis». Гра поєднує елементи з попередніх ігор з новим контентом та рівнями. Гравці керують Соніком та його друзями (Тейлзом і Наклзом), досліджуючи ретельно розроблені рівні, борючись з ворогами та збираючи

кільця (рис. 1.1). Гра зберігає високі швидкості та динамічний геймплей, характерні для серії.

Рисунок 1.1 – Скрін гри «Sonic Mania»

«Sonic Mania» містить ідеальне поєднання всього, що робить Соніка особливим. Ця гра використовує піксельну графіку, яка нагадує оригінальні ігри, але з покращеннями, які забезпечують більш гладкі анімації та візуальні ефекти. У грі є етапи, які заохочують до впевненої швидкості, творчі боси, які вшановують історію Соніка.

Sonic Mania прибув якраз до 25-річчя Sonic, і його очолив провідний програміст і друг пікселів і полісів, Крістіан «Податківець» Вайтхед. Найпомітнішою рисою Соніка, його швидкістю, часто було скасування ігор із дизайном рівнів, який не міг повністю вмістити персонажа, який рухається так само швидко, як Соник.

Можливо, більше ніж у будь-якій іншій грі Соника, Sonic Mania демонструє пересування свого головного героя та, у свою чергу, викликає

усмішку на обличчі гравців, коли вони проносяться зонами, перекочуються через петлі та стукають між бамперами.

Рівні та фізика Sonic Mania – це не єдине, що варто відзначити. Музика Ті Лопеса забезпечує саундтрек, який по суті є класичним Соником і водночас сучасним. Ціла низка ігрових персонажів, нескінченну кількість повторів, приголомшливі битви з босами та безліч ігрових режимів, і легко зрозуміти, що робить Sonic Mania однією з найкращих 2D-платформерів і справді найкращих ігор усіх часів.

Sonic Mania є чудовим прикладом того, як можна взяти класичний геймплей і внести в нього нові елементи, зберігаючи при цьому дух оригіналу. Це робить її важливою грою для розгляду при аналізі найкращих 2D платформерів.

Наступний аналог – «Celeste» (2018) (рис. 1.2). Це одна з найвідоміших і найбільш захоплюючих 2D платформерів останніх років. Гра отримала численні нагороди та високу оцінку критиків і гравців за свій геймплей, сюжет та художній стиль. «Celeste» – це важкий 2D платформер, в якому гравець керує дівчиною на ім'я Маделін, яка намагається підкорити гору Селеста. Гра пропонує безліч викликів і головоломок, які вимагають точних і добре обміркованих дій.

Головна механіка включає стрибки, даші та лазіння по стінах. Гравці повинні використовувати ці можливості для подолання складних перешкод на кожному рівні. Сюжет «Celeste» розповідає про внутрішні боротьби та самопізнання Маделін. Це історія про подолання власних страхів і труднощів, яка надає грі емоційної глибини. ра використовує ретро-піксельну графіку, яка додає їй чарівності. Візуальний стиль відзначається деталізацією та привабливими пейзажами [4].

Саундтрек до гри, створений Лєною Рейн, отримав багато похвал за свої мелодії, що доповнюють атмосферу гри та емоційну складову сюжету. «Celeste» отримала численні нагороди, включаючи "Найкраща незалежна гра" на The Game Awards 2018 та "Найкраща гра, що впливає" (Games for Impact).

Рисунок 1.2 – Скрін гри «Celeste» (2018)

Основні елементи успіху гри:

1. Геймплей: точний контроль і складні головоломки роблять кожен рівень викликом, але й винагороджують гравця за успіх.
2. Сюжет: емоційно насичена історія про подолання труднощів та внутрішніх страхів додає грі глибини і резонує з багатьма гравцями.
3. Атмосфера: поєднання ретро-графіки та чудової музики створює унікальну атмосферу, яка залучає гравців у світ гри.

«Celeste» є відмінним прикладом того, як незалежна гра може досягти великих висот завдяки продуманому геймплею, сильному сюжету та високоякісному виконанню.

«Hollow Knight» (2017) – це глибокий і складний платформер з відкритим світом, який пропонує гравцям досліджувати підземне королівство Халлоунест. Гра поєднує елементи платформера з екшеном та дослідженням.це один з найвідоміших та найуспішніших 2D платформерів з

елементами метродванії, який здобув визнання як серед критиків, так і гравців. Дія гри відбувається в підпільному суспільстві жуків і моторошних повзаків, «Hollow Knight» (рис. 1.3) протистоїть тенденціям багатьох 2D-платформерів і використовує приглушену та темнішу палітру кольорів, ніж багато хто з його сучасників [5].

Граючи за скромного лицаря, гравець повинен подорожувати до покинутого королівства та розкрити правду про інфекцію, яка спустошила колись велику цивілізацію. Шанувальники серіалу «Souls» почуватимуться як вдома з великою кількістю оповідань про навколишнє середовище та історії.

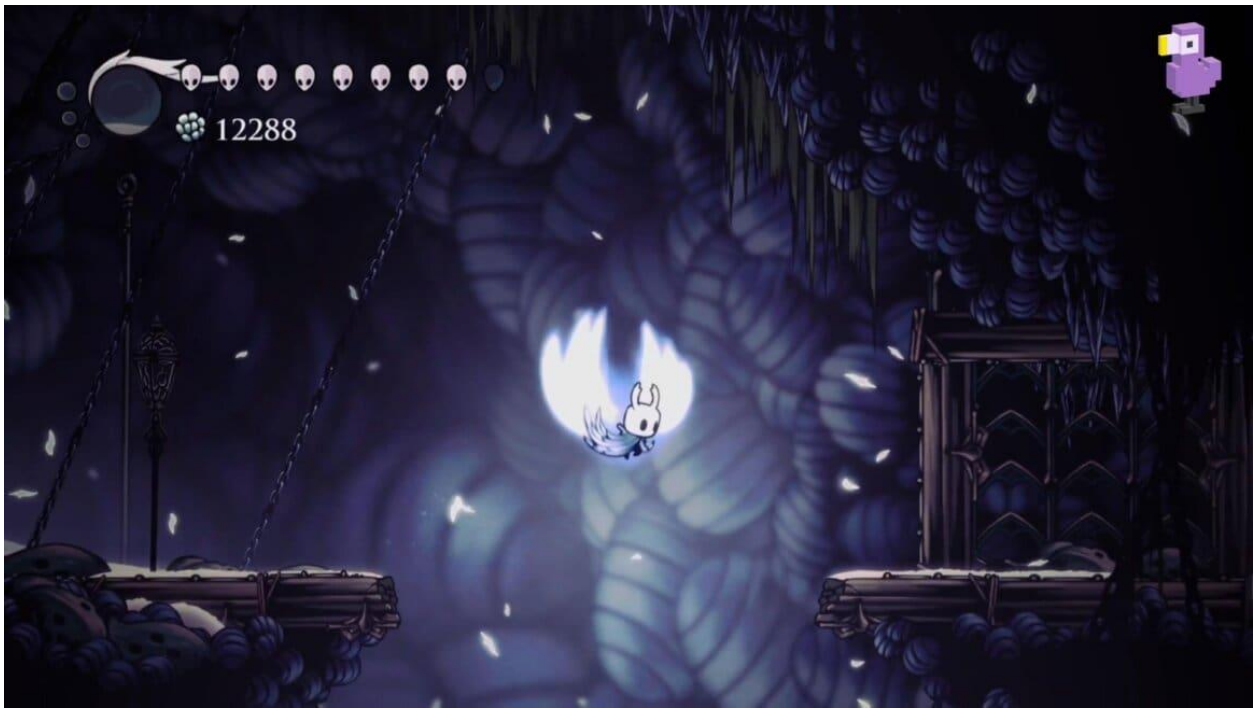

Рисунок 1.3 – Скрін гри «Hollow Knight» (2017)

Hallownest – це лабіринт цікавих проходів і прихованих алей, які складно пов’язані між собою. Завдяки вдосконаленню Лицаря відкриваються нові здібності, які дозволяють гравцям глибше досліджувати невідоме та відкривати всі види жахів і загроз. Не для слабкодухих, «Hollow Knight» – складна гра, у яку легко закохатися.

Основні елементи успіху «Hollow Knight»:

1. Геймплей: поєднання платформінгу та бойових елементів з відкритим світом і механіками метродванії створює захоплюючий ігровий процес, який постійно викликає інтерес.
2. Світ та атмосфера: детально розроблений світ Халлоунест з багатьма зонами та персонажами, кожен з яких має свою історію та атмосферу.
3. Графіка та музика: візуальний стиль з ручним малюванням та чудовий саундтрек створюють незабутню атмосферу, яка поглинає гравця.

1.4 Функціональні вимоги до ігрового застосунку

При розробці гри на рушії Unity важливо визначити функціональні вимоги, які будуть описувати, що саме гра має робити і як вона має працювати. Розробка детального списку функціональних вимог допоможе уникнути непорозумінь під час розробки, а також забезпечити, що кінцевий продукт відповідатиме очікуванням як розробника, так і гравців.

Нижче перелічені основні категорії функціональних вимог, які можуть бути корисні:

1. Основний ігровий процес:
   - ігрові механіки: опис основних дій, які можуть виконувати гравці (рух, стрибки, атаки, взаємодія з об'єктами);
   - рівні: структура рівнів, включаючи кількість, дизайн, складність та прогресію;
   - цілі гри: що гравець має досягти для завершення рівня або гри в цілому.
2. Користувацький інтерфейс (UI):
   - меню: головне меню, меню налаштувань, паузи, екран завершення гри;

- HUD (head-up display): інформація, що відображається на екрані під час гри (життя, очки, карта);
- інтерактивні елементи: кнопки, слайдери, діалогові вікна.

3. Графіка та анімація:
   - візуальний стиль: загальний стиль гри (2d, 3d, піксель-арт, реалістичний тощо);
   - моделі та текстури: вимоги до якості та деталізації моделей та текстур;
   - анімація: вимоги до анімацій персонажів та об'єктів (рухи, атаки, взаємодії) [6].
4. Аудіо:
   - фоновий звук: музика, що відтворюється під час гри;
   - ефекти: звукові ефекти для дій (стрибок, постріл, удар);
   - діалоги: наявність озвучених діалогів або текстових повідомлень.

1.4.1 Моделювання варіантів використання ігрового застосунку

Діаграма варіантів використання (Use Case Diagram) є корисним інструментом для проектування гри, оскільки допомагає візуалізувати і структурувати функціональні вимоги, а також зрозуміти взаємодію користувачів (гравців) з системою (грою).

Діаграма використання відображає всі можливості і механіки гри, які може надавати вона для акторів, тобто її користувачів. Для даного проекту актор один – це гравець, так як в грі не передбачено використання ШІ як віртуального супротивника або мережне використання проекту з можливістю грати декільком гравців одночасно.

Для варіантів використання є три межі, які характерні проекту: це головне меню, HUD та геймплей. Буде зручно відображати їх на діаграмі окремими блоками, це допоможе наочність під час реалізації відповідних механік і спростить читання діаграми [7].

Перше, з чим буде працювати користувач-гравець – це головне меню гри і ігрове меню, яке буде завжди відображатись на екрані. Діаграму варіантів використання для цих об’єктів гри представлено на рисунку 1.4.

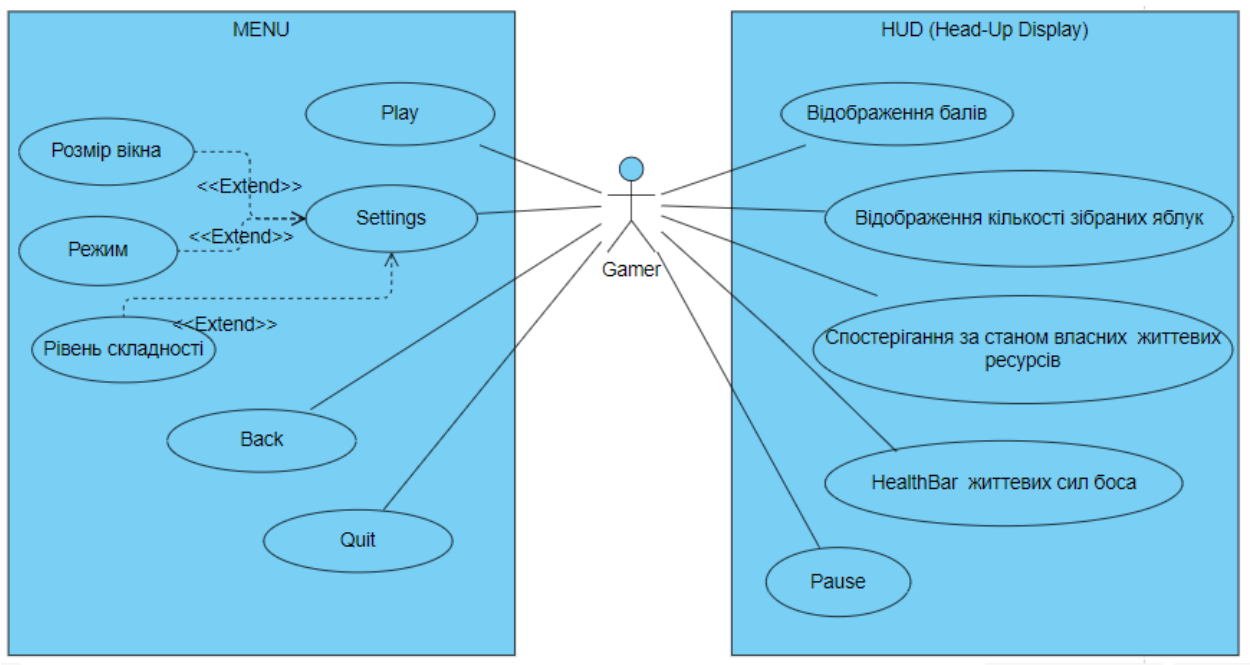

Рисунок 1.4 – Діаграма варіантів використання для меню ігрового додатку

Головне меню повинно включати в себе такі кнопки по роботі з ігровим додатком, як:

- запустити гру;
- налаштування ігрового процесу (зміна розміру вікна, вибір режиму на весь екран чи в мінімальному розмірі, вибір рівня складності гри);
- повернення до гри;
- закрити ігровий додаток.

Для хеадап-дісплею необхідно передбачити такі варіанти використання:

- відображення загальної кількості балів;
- відображення кількості зібраних яблук;
- Healthbar для спостерігання за станом власних життєвих ресурсів;
- Healthbar для відображення сил боса;

– функція паузи ігрового процесу.

Наступна діаграма демонструє варіанти використання щодо самого гемплею. Актор – гравець (рис. 1.5).

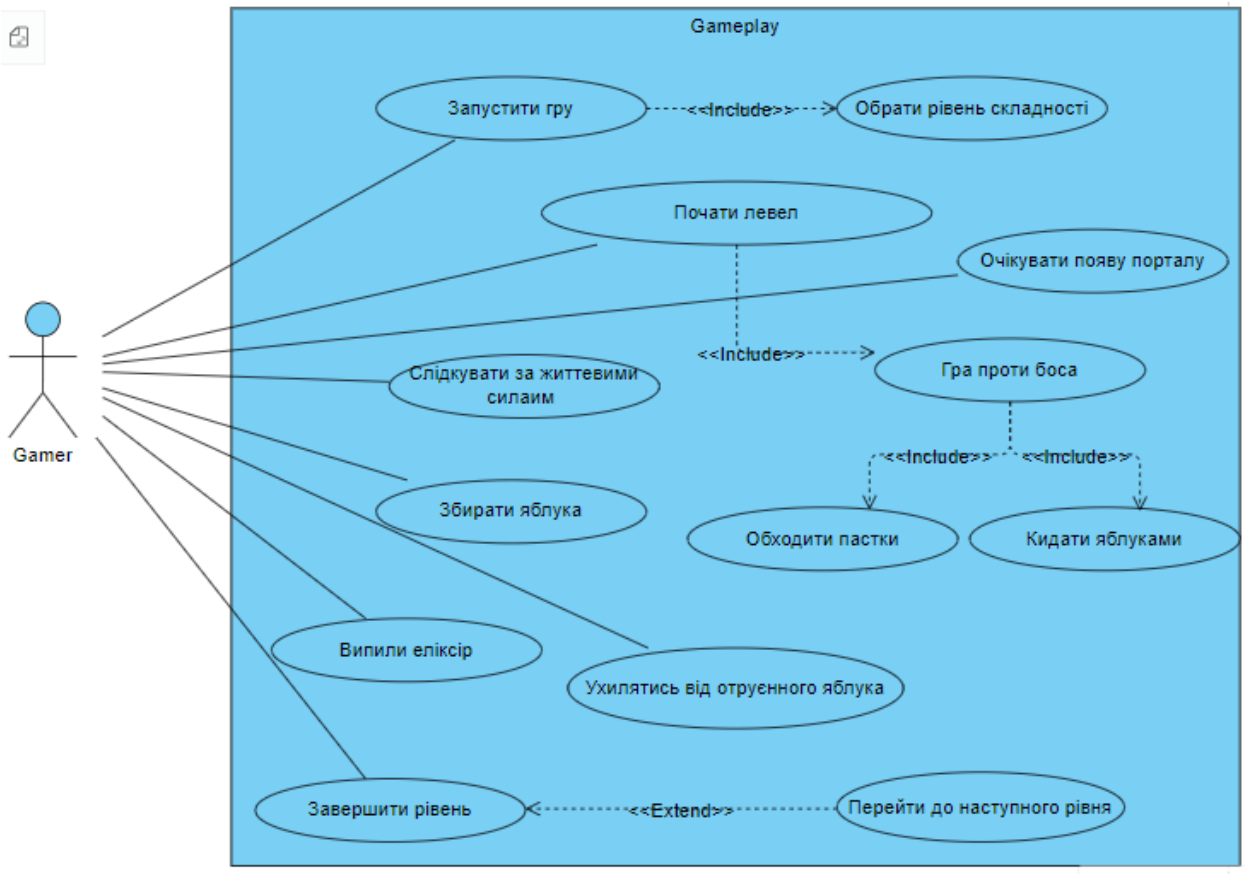

Рисунок 1.5 – Діаграма варіантів використання для гемплею

Гравцю доступні такі функції:

– запустити гру, включаючи вибір рівня складності левела;
– почати проходження рівня, який обов’язково у кінці передбачатиме появу боса та битву з ним;
– гра проти боса, з використанням зібраних яблук як знаряддя проти ворога та слідкування за його життєвими силами.
– очікування гравця на появу порталу, який відкриє прохід до наступного рівня гри;
– збирати яблука для поповнення власних сил;

- ухилятись від отруєного яблука;
- завершення рівня, з можливістю переходу до наступного випробування.

1.4.2 Опис сценаріїв варіантів використання

Наступним етапом є детальний опис кожного з зазначених вище варіантів використання. Для кожного з прецедентів буде наведена його назва, передумова, основний та, за наявності, альтернативний сценарій:

- назва має відображати основну діяльність, яка виконується у цьому варіанті використання, а також має відповідати назві, зазначеній на діаграмі;
- передумова демонструє стан системи, під час якого можливо задіяти цей варіант використання, можуть входити умови, в яких межах знаходиться гравець (в меню або в грі) або чи виконав він вже певний елемент сюжету;
- основний сценарій демонструє кроки, які необхідно виконати для успішного завершення прецеденту;
- альтернативний сценарій показує можливі відхилення або виключення з пунктів основного сценарію, якщо такі наявні.

Почнемо з діаграми для головного меню (рис. 1.4).

ВВ №1 «Розпочати гру».

Актор: Гравець.

Передумови: Гравець запустив ігровий застосунок і знаходиться у головному меню гри.

Основний сценарій:

1. Гравець натискає кнопку "Play".
2. Система переводить застосунок до другої сцени (ігрової сцени).

Очікуваний результат: гра починається.

ВВ №2 «Налаштування» (Settings).

Актор: Гравець.

Передумова: Гравець знаходиться у головному меню гри.

Основний сценарій:

1. Гравець натискає кнопку "Settings".
2. Система відкриває вікно із налаштуваннями.
3. Гравець змінює розмір вікна гри під роздільну здатність свого екрану.
4. Гравець обирає режим відображення (у вікні або на весь екран).
5. Гравець змінює рівень складності гри.
6. Гравець натискає кнопку "Back".
7. Система повертає гравця до головного меню.

Очікуваний результат: налаштування збережено, Гравець повертається до головного меню.

ВВ№3 «Вийти з гри» (Quit).

Актор: Гравець.

Передумова: Гравець знаходиться у головному меню гри.

Основний сценарій:

1. Гравець натискає кнопку "Quit".
2. Система завершує роботу застосунку.

Очікуваний результат: Гравець виходить з гри.

ВВ№4 «Повернутися» (Back).

Актор: Гравець.

Передумова: Гравець знаходиться у головному меню гри в налаштуваннях.

Основний сценарій:

1. Гравець натискає кнопку " Back".
2. Система реагує на запит Гравця.

Очікуваний результат: Гравець повертається у головне меню гри.

Далі надано опис варіантам використання стосовно гемплею (рис. 1.5). Нумерацію варіантів використання буде продовжено.

ВВ№5 «Почати левел».

Актор: Гравець.

Передумова: Гравець знаходиться у головному меню гри.

Основний сценарій:

1. Гравець натискає кнопку " Play".
2. Система реагує на запит Гравця та завантажує левел на стартовій сцені гри.

Очікуваний результат: Гравець бачить розгорнутий левел з активними функціями управління.

ВВ№6 «Очікувати появу порталу».

Актор: Гравець.

Передумова: Гравець запустив рівень гри для проходження завдань.

Основний сценарій:

1. Гравець починає проходити рівень гри.
2. Гравець слідкує за відліком часу.
3. При появі на сцені порталу Гравець швидко направляє юніт головного героя у напрямку порталу щоб встигнути пройти через нього.

Очікуваний результат: Гравець проходить портал та Система генерує наступний рівень і відкриває його стартову сцену.

ВВ№7 «Слідкувати за життєвими силами».

Актор: Гравець.

Передумова: Гравець запустив рівень гри для проходження завдань.

Основний сценарій:

1. Гравець починає проходити рівень гри.
2. Гравець намагається як можна більше зловити яблук, кожне з яких додає його життєві сили.

Очікуваний результат: Гравець бачить в меню гри рівень Healthbar, який зростає по мірі отримання яблук.

ВВ№8 «Гра проти боса».

Актор: Гравець.

Передумова:

1. Гравець запустив рівень гри для проходження завдань.
2. Час рівня гри спливає і з’являється на ігровій сцені юніт боса.

Основний сценарій:

1. Гравець намагається як можна більше разів влучити в боса яблуками.
2. Гравець обходить пастки з отруєними яблуками.
3. Гравець слідкує на рівнем життєвий сил боса.
4. Гравець слідкує на власним рівнем життєвий сил.

Очікуваний результат: Гравець бачить в меню гри рівень Healthbar боса, який зменшується до мінімальної відмітки і бос зникає.

ВВ№9 «Збирати яблука».

Актор: Гравець.

Передумова:

1. Гравець запустив рівень гри для проходження завдань.
2. Гравець обходить пастки отруєними яблуками.

Основний сценарій:

1. Гравець слідкує на власним рівнем життєвий сил.
2. На сцені з’являються яблука і Гравець намагається їх спіймати.
3. Після схопленого яблука додається одиниця життєвих сил Гравця.

Очікуваний результат:

1. Гравець бачить в меню гри кількість яблук, які він спіймав.
2. Кількість бонусів зростає.

Альтернативний сценарій:

1. Гравець слідкує на власним рівнем життєвий сил.
2. На сцені з’являються яблука і Гравець намагається їх спіймати.

3. З кожним періодом часу у 10 секунд рівень життя головного героя знижується.
4. У Гравця не виходить зловити яблука і рівень життєвих сил падає у мінімальну позицію.
5. Рівень закінчується поразкою для Гравця.

ВВ№10 «Ухилятись від отруєного яблука».

Актор: Гравець.

Передумова:

1. Гравець запустив рівень гри для проходження завдань.

Основний сценарій:

1. Гравець слідкує за появою нових ігрових елементів на сцені.
2. На сцені з’являються яблука і Гравець намагається їх спіймати.
3. Коли швидкість появи яблук зростає, на сцені з’являються отруєні яблука.
4. Гравець керує напрямком руху свого ігрового юніта.

Очікуваний результат:

1. Гравець ухиляється від отруєного яблука.
2. Отруєне яблуко зникає при досягнені нижньої платформи сцени гри.

Альтернативний сценарій:

1. Гравець слідкує за появою нових ігрових елементів на сцені.
2. На сцені з’являються яблука і Гравець намагається їх спіймати.
3. Коли швидкість появи яблук зростає, на сцені з’являються отруєні яблука.
4. Гравець керує напрямком руху свого ігрового юніта.
5. Гравець перетинається з отруєним яблуком.
6. Рівень життєвих сил юніта Гравця зменшується на одиницю.

ВВ№11 «Випити еліксир».

Передумова:

1. Гравець запустив рівень гри для проходження завдань.

2. Гравець набрав необхідну кількість бонусів для покупки еліксиру.

Основний сценарій:

1. Гравець слідкує за кількістю набраних бонусів.
2. На сцені з’являється юніт еліксиру.
3. Гравець керує напрямком папуги щоб зловити пляшку.

Очікуваний результат:

1. Гравець ловить пляшку еліксиру.
2. На панелі меню з’являється новий спрайт пляшки еліксиру.

Альтернативний сценарій:

1. Гравець слідкує за рівнем життєвих сил.
2. На сцені з’являється еліксир.
3. Гравець керує напрямком руху свого ігрового юніта щоб схопити пляшку але не встигає.
4. Пляшка зникає на нижній межі платформи.
5. У випадку закінчення життєвих сил головного герою рівень завершується поразкою.

ВВ№12 «Завершення рівня».

Передумова:

1. Гравець запустив рівень гри для проходження завдань.

Основний сценарій:

1. Гравець вирішує вийти з гри.
2. Гравець натискає на кнопку «Pause».
3. Стан гри зберігається.

Очікуваний результат:

1. Гравець вийшов з гри.
2. Гравець повертається у гру на продовжує проходити рівень з точки попереднього виходу.

1.5 Опис нефункціональних вимог

Нефункціональні вимоги описують атрибути, які визначають якість системи і способи її реалізації, такі як продуктивність, безпека, зручність користування тощо. Для гри, яка включає меню з інтерактивними кнопками «Play», «Settings» та «Quit», можна визначити наступні нефункціональні вимоги:

1. Продуктивність.
   - час завантаження: гра повинна завантажуватися протягом 5 секунд після натискання кнопки «Play»;
   - реакція інтерфейсу, а саме кнопки в меню повинні реагувати на натискання протягом 0,1 секунди.
2. Масштабованість.

Гра повинна коректно працювати на роздільних здатностях від 800x600 до 3840x2160.

3. Надійність.
   - стійкість до збоїв: гра не повинна аварійно завершуватися більше ніж один раз на 10 годин гри;
   - збереження даних: налаштування гри повинні зберігатися між сесіями.
4. Зручність користування.

Інтерфейс головного меню повинен бути інтуїтивно зрозумілим для користувача без необхідності додаткових інструкцій. Користувач повинен мати можливість змінювати налаштування гри за допомогою не більше ніж трьох натискань миші.

5. Конфіденційність даних.

Гра не повинна збирати або зберігати особисті дані користувачів без їхньої згоди.

6. Документування коду.

Вихідний код гри повинен бути задокументований для полегшення підтримки і оновлення.

7. Дизайн інтерфейсу та інтерактивні підказки.

Інтерфейс повинен бути естетично приємним і відповідати стилю гри. Гра повинна надавати інтерактивні підказки, щоб допомогти новим користувачам зрозуміти, як використовувати інтерфейс.

Ці нефункціональні вимоги забезпечать, що гра буде не тільки функціональною, але і приємною у використанні, надійною і сумісною з різними пристроями та операційними системами [6].

1.6 Концепція гри

Назва ігрового додатку: "Пригоди папуги Pio".

Жанр: пригодницька гра з елементами платформера.

Платформи: PC Windows.

Цільова аудиторія: діти та підлітки (6-15 років), любителі пригодницьких ігор.

Опис гри: "Пригоди Папуги Pio" – це яскрава і захоплююча пригодницька гра, де гравці беруть на себе роль хороброго папуги на ім'я Pio. Гравцеві належить пройти через різні рівні, розгадуючи головоломки, уникаючи пасток та збираючи бонуси, щоб врятувати своїх друзів і знайти логово босів та подолати їх. Гравці зануряться у захоплюючий світ, де їх чекають незабутні пригоди та неймовірні відкриття.

Головний герой:

Ім'я: Pio.

Характеристика героя: Pio – це веселий і відважний папуга, завжди готовий до нових пригод. Він вирушає у подорож по Виміру, щоб зупинити зловісного ворога і врятувати світ від знищення. Він швидкий, кмітливий і дуже дружелюбний.

Основні можливості гри:

1. Інтерактивні елементи: Гравець може стрибати, літати на короткі дистанції, збирати бонуси та вирішувати головоломки.
2. Рівні: різноманітні рівні, від густих джунглів до стародавніх храмів і скельних ущелин.
3. Бонуси: збирання фруктів, цінних предметів, еліксирів.
4. Перешкоди: уникання пасток, ворогів та природних небезпек, таких як колючки та обвали.
5. Налаштування: можливість змінювати розмір вікна гри, обирати режим відображення (у вікні або на весь екран) та налаштовувати рівень складності.

Історія:

Гра розпочинається у мирному зоряному королівстві, де живе папуга Ріо. Одного дня таємничий портал відкривається, і з нього виходять чудернацькі істоти, що загрожують мирному життю населення. Папуга Ріо вирішує взяти справи у свої руки і відправитися на пошуки джерела цих таємничих істот, щоб зупинити загрозу та врятувати світ.

Геймплей:

1. Головне меню:
   - Play: початок гри, переведення до першого рівня;
   - Settings: вікно налаштувань, де гравець може змінити розмір вікна гри, обрати режим відображення та налаштувати складність. Кнопка "Back" повертає до головного меню;
   - Quit: вихід з гри.
2. Рівні:
   - Джунглі: перший рівень, де гравець знайомиться з основними механіками гри;
   - Печери: більш складний рівень з головоломками;
   - Стародавні храми: рівень з багатьма пастками і прихованими скарбами;

- Скельні ущелини: кінцевий рівень з головним босом.

3. Механіки:
   - Стрибки: полі може стрибати, щоб уникати перешкоди;
   - Політ: короткий політ для подолання великих відстаней;
   - Збирання бонусів: гравець збирає фрукти, монети та інші цінні предмети для підвищення бонусів;
   - Головоломки: вирішення простих головоломок для просування вперед.
4. Графіка та звук:
   - Графіка: кольорові та яскраві 2D спрайти, мультяшний стиль, що приваблює дітей;
   - Звукові ефекти: веселі звукові ефекти для стрибків, збирання бонусів та інших дій;
   - Музика: легка та мелодійна фонові музика, що створює атмосферу пригод.

2 ПЛАНУВАНЯ ІГРОВОГО ЗАСТОСУНКУ

2.1 Визначення плану виконання робіт

Структура декомпозиції робіт (WBS – Work Breakdown Structure) – це методологічний інструмент управління проектами, що використовується для ієрархічного розподілу проекту на менші, керовані компоненти. WBS допомагає проектним командам організувати роботу, визначити всі необхідні завдання та підвищити ефективність управління проектом.

WBS для об’єкту розробки включає такі етапи:

1. Ініціація проекту:
   - визначення мети та завдань;
   - розробка концепт-документу;
   - затвердження графіку проекту;
2. Дослідження та планування:
   - збір вимог;
   - аналіз ринку та конкурентів;
   - розробка попереднього плану проекту;
   - визначення унікальних особливостей гри.
3. Розробка концепції та прототипування:
   - створення концепт-документу;
   - прототипування ігрових механік.
4. Дизайн:
   - дизайн рівнів;
   - створення персонажів;
   - розробка анімацій;
   - дизайн фонів та елементів оточення;
   - інтерфейс користувача (UI) дизайн.
5. Програмування:
   - розробка основної ігрової логіки;
   - реалізація управління персонажем;

- програмування фізики гри;
- створення системи збережень та завантажень;
- розробка меню та налаштувань;
- інтеграція звукових ефектів та музики.

6. Графіка:
   - створення 2d спрайтів персонажів;
   - розробка анімацій для персонажів;
   - дизайн фонів та декорацій;
   - створення анімаційних ефектів;
   - інтеграція графічних елементів в гру.
7. Звук: створення звукових ефектів.
8. Тестування
   - розробка тестових сценаріїв;
   - тестування;
   - виправлення багів після тестування [8].

2.2 Побудова діаграми Ганта

Діаграма Ганта – це графічний інструмент управління проектами, що відображає план проекту на шкалі часу. Вона показує завдання проекту, їх тривалість та взаємозв’язки між ними у вигляді горизонтальних смуг.

Основні елементи діаграми Ганта:

1. Шкала часу: горизонтальна вісь, що показує дати та терміни виконання завдань.
2. Завдання: вертикальна вісь, де перераховані всі завдання проекту.
3. Горизонтальні смуги: представляють тривалість кожного завдання, їх початок і кінець.
4. Віхи (milestones): ключові події або дати в проекті, які зазвичай відзначаються окремими символами [9].

5. Зв’язки між завданнями: стрілки або лінії, що показують залежності між завданнями.

Діаграма Ганта допомагає у плануванні проекту, а саме дозволяє розбити проект на окремі завдання, визначити їх тривалість і побудувати загальний графік виконання проекту. Крім цього, вона забезпечує наочне відображення плану проекту, що допомагає всім учасникам проекту легко зрозуміти, які завдання потрібно виконати та в які терміни.

Діаграма Ганта допомагає відстежувати прогрес проекту, визначати, які завдання вже виконані, які ще в процесі, а які відстають від графіку. Вона показує взаємозв’язки між завданнями, що допомагає команді краще координувати свої дії, розуміючи, які завдання залежать від виконання інших.

Головна задача діаграми Ганта – допомога при виявленні критичного шляху – послідовність завдань, що визначає мінімальний час завершення проекту. Це важливо для розуміння, які завдання не можуть бути затримані без впливу на загальний термін проекту.

Також вона сприяє плануванню та розподілу ресурсів, оскільки дозволяє бачити, коли і які ресурси будуть задіяні на різних етапах проекту та служить ефективним інструментом для спілкування з клієнтами, стейкхолдерами та командою, надаючи чітке уявлення про стан проекту та його подальший розвиток [9].

Для побудови діаграми Ганта спершу слід розробити структуру послідовності робіт (табл. 2.1).

Таблиця 2.1 – Структура декомпозиції робіт

№	Назва задачі	Початок	Завершення
1	Специфікація вимог	25.01.2024	06.02.2024
1.1	Опис предметної області	25.01.2024	29.01.2024
1.2	Функціональні вимоги	30.01.2024	01.02.2024
1.3	Нефункціональні вимоги	05.02.2024	05.02.2024

Продовження таблиці 2.1

2	Розробка концепції гри	06.02.2024	16.02.2024
2.1	Створення концепт-документу	06.02.2024	09.02.2024
2.2	Опис ігрових механік та рівнів	12.02.2024	16.02.2024
3	Проектування	19.02.2024	29.02.2024
3.1	Декомпозиція робіт	19.02.2024	20.02.2024
3.2	Опис модулів ігрового застосунку	21.02.2024	26.20.2024
3.3	Проектування UI	27.02.2024	29.02.2024
4	Прототипування ігрових механік	01.03.2024	15.03.2024
4.1	Прототипування базового руху персонажа	01.03.2024	06.03.2024
4.2	Прототипування основних перешкод та пасток	07.03.2024	13.03.2024
4.3	Прототипування збирання бонусів	14.03.2024	15.03.2024
5	Дизайн	18.03.2024	02.04.2024
5.1	Дизайн і деталізація рівнів	18.03.2024	20.03.2024
5.2	Балансування складності рівнів	21.03.2024	25.03.2024
5.3	Створення персонажів	26.03.2024	28.03.2024
5.4	Розробка анімацій	29.03.2024	02.04.2024
6	Програмування	03.04.2024	19.04.2024
6.1	Розробка основної ігрової логіки	03.04.2024	09.04.2024
6.2	Програмування руху персонажа	10.04.2024	15.04.2024
6.3	Програмування взаємодії з об'єктами	16.04.2024	19.04.2024
7	Тестування	22.04.2024	17.05.2024
7.1	Створення тест-кейсів	22.04.2024	24.04.2024
7.2	Тестування варіантів використання	25.04.2024	6.05.2024
7.3	Виправлення помилок	07.05.2024	17.05.2024
8	Оформлення документації (пояснювальної записки)	20.05.2024	31.05.2024

Діаграму Ганта представлено на рисунку 2.1.

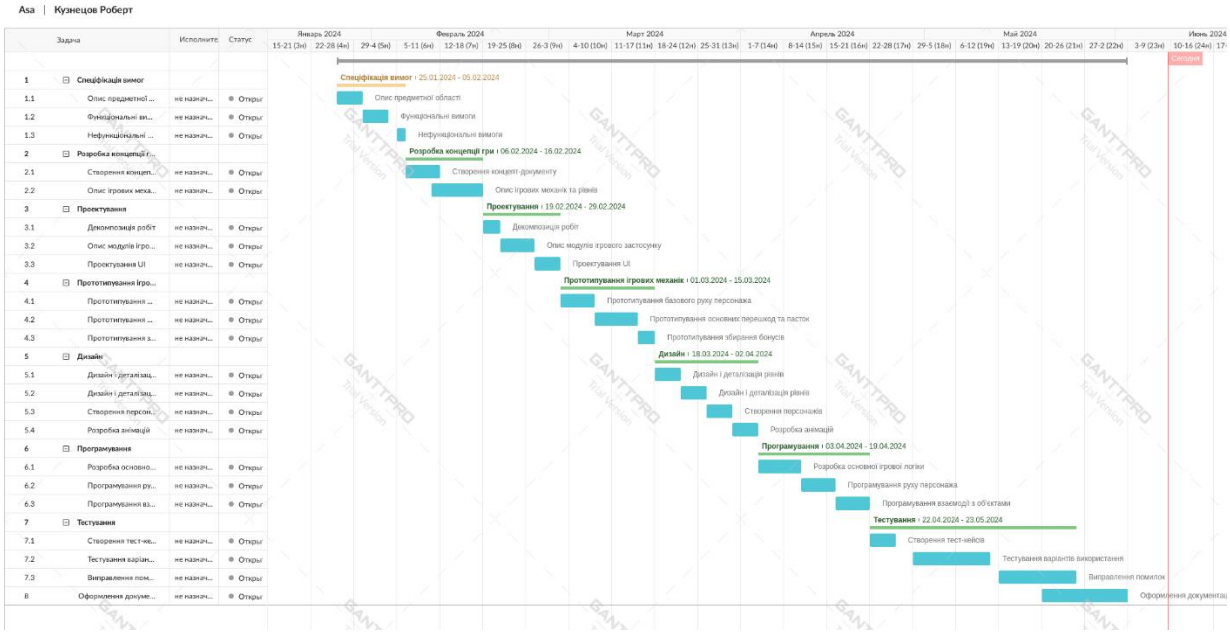

Рисунок 2.1 – Діаграма Ганта

2.3 Визначення ризиків проекту

Першим кроком у роботі з ризиками є визначення всіх можливих ризиків, які можуть виникнути під час виконання проекту. Для цього ідеально підходить діаграма Ішикаві, також відома як «діаграма причинно-наслідкових зв’язків». Використання діаграми Ішикаві для аналізу ризиків:

1. Визначення основних проблем.

На початку визначаються основні проблеми, які можуть завадити успішному виконанню завдання. Це можуть бути такі категорії, як технічні проблеми, проблеми з ресурсами, людські фактори, зовнішні впливи тощо.

2. Виявлення факторів.

Для кожної з основних проблем визначаються фактори, що їх спричиняють. Наприклад, для технічних проблем це можуть бути недостатні технічні знання, збої в обладнанні або програмному забезпеченні [10].

3. Аналіз складових.

Далі, для кожного фактору визначаються найменші складові, які створюють ці проблеми. Це ті конкретні ситуації або події, які можуть виникнути та спричинити проблеми. Наприклад, для фактора "збої в обладнанні" це можуть бути поломки, перебої в електропостачанні або програмні помилки.

4. Візуалізація.

Усі ці елементи зображуються на діаграмі Ішикаві, що дозволяє візуально побачити всі можливі причини проблем і їх взаємозв'язки. Центральною лінією діаграми є проблема, а від неї відходять основні фактори, які в свою чергу розбиваються на менші складові.

5. Ідентифікація ризиків.

Використовуючи діаграму, можна систематично ідентифікувати всі потенційні ризики, які можуть виникнути під час реалізації проекту. Це дозволяє краще підготуватися до їх можливого впливу і розробити плани щодо їхнього уникнення або мінімізації [10].

На рисунках 2.2 та 2.3 зображені діаграми Ішикаві, що демонструють критичні ситуації, пов'язані з нестачею часу та якістю.

Рисунок 2.1 – Діаграма Ішикаві для дедлайну

Рисунок 2.2 – Діаграма Ішикаві для якості проекту

Такий вибір критичних ситуацій дозволяє розглянути найбільш ймовірні та значущі ризики, які можуть суттєво вплинути не лише на створення програмного забезпечення, а й на виконання кваліфікаційної роботи в цілому.

3 ПРОЕКТУВАННЯ

3.1 Обґрунтування вибору засобів розробки

У сфері розробки ігор два провідні движки індустрії стали основними виборами для новачків: Unreal Engine і Unity. Обидва механізми пропонують потужні інструменти та можливості, які задовольняють потреби початківців розробників ігор. Однак вибір правильного двигуна може бути складним завданням, особливо для початківців [11].

3.1.1 Графіка та можливості візуалізації

Unreal Engine (рис. 3.1) відомий своєю фотореалістичною графікою. Розширені можливості візуалізації движка дозволяють розробникам створювати приголомшливі візуальні ефекти, які можуть конкурувати з іграми AAA. За допомогою Unreal Engine ви можете створити реалістичне освітлення, тіні та відображення, оживляючи свій ігровий світ.

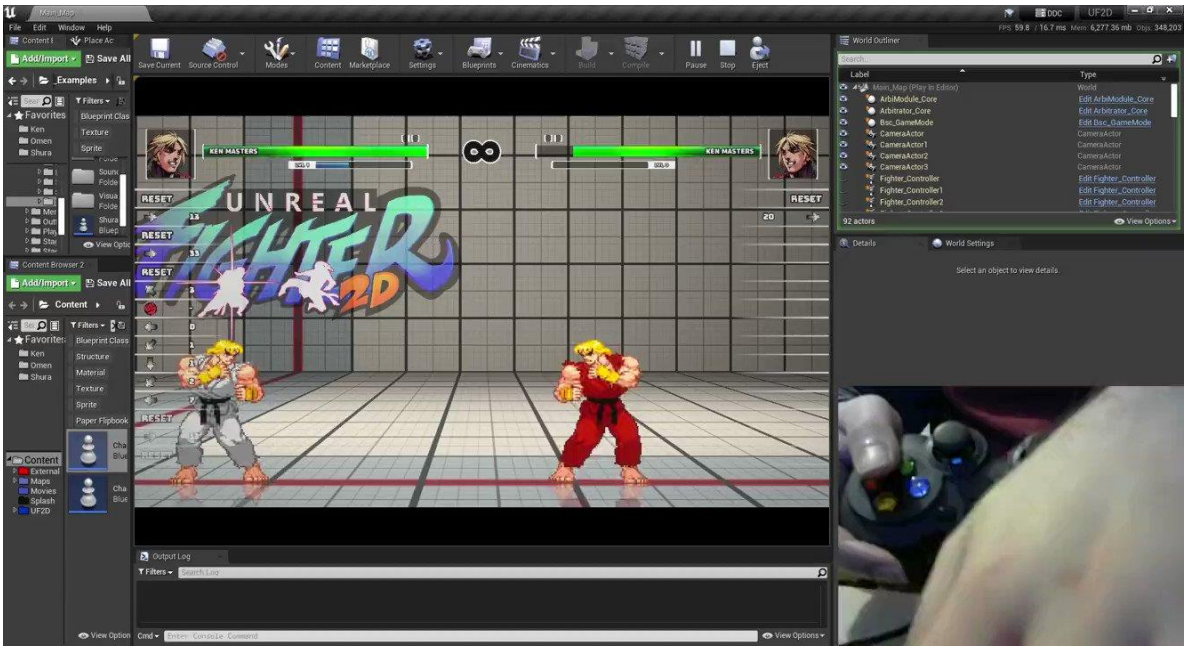

Рисунок 3.1– Робоче вікно Unreal Engine

З іншого боку, Unity (рис. 1.5) пропонує гнучкі варіанти візуалізації, які задовольняють широкий спектр проектів.

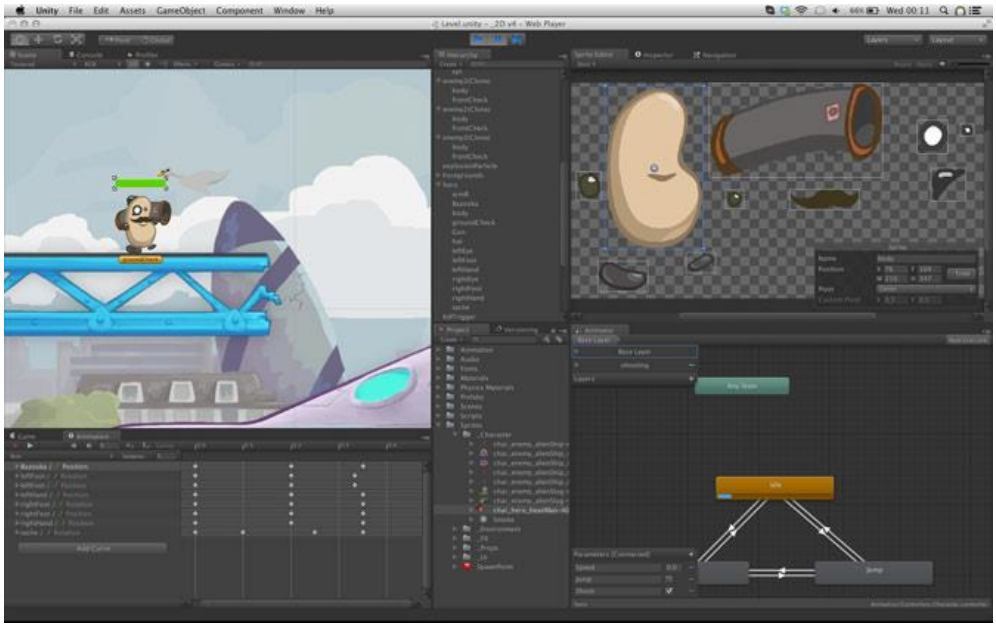

Рисунок 3.2 – Робоче вікно Unity

Хоча Unity може не відповідати фотореалістичній графіці Unreal Engine з коробки, він надає розробникам інструменти та гнучкість для створення візуально привабливих ігор. Конвеєр рендеринга Unity можна налаштувати та оптимізувати відповідно до конкретних потреб вашої гри, забезпечуючи візуально приємний досвід для гравців.

Обидва двигуни мають свої сильні сторони в графіці та візуалізації, причому Unreal Engine перевершує фотореалізм, а Unity пропонує гнучкість і налаштування. Вибір зрештою залежить від візуального стилю та вимог вашої гри [11].

3.1.2 Фізика та моделювання

Unreal Engine може похвалитися вдосконаленим фізичним движком, який дозволяє реалістичне моделювання та взаємодію. Від складної фізики

ragdoll до симуляції транспортних засобів, Unreal Engine надає розробникам широкий спектр інструментів на основі фізики для створення захоплюючого ігрового процесу. Завдяки таким функціям, як руйнування на основі фізики та симуляція рідини, Unreal Engine розширює межі можливого з точки зору фізики.

Unity також пропонує фізичний механізм і можливості моделювання, які підходять для більшості ігрових проектів. Хоча фізичний движок Unity може бути не таким надійним, як Unreal Engine, він все ж надає розробникам необхідні інструменти для створення захоплюючої механіки ігрового процесу. Фізичний движок Unity простий у використанні та добре інтегрується з іншими аспектами движка, що дозволяє бездоганно інтегрувати фізику у вашу гру.

Обидва механізми чудово підходять для фізики та симуляції, причому Unreal Engine пропонує розширені функції, а Unity – зручний і універсальний фізичний механізм. Приймаючи рішення, враховуйте складність і вимоги фізичної взаємодії вашої гри [12].

3.1.3 Мови сценаріїв і програмування

Unreal Engine пропонує унікальну систему візуальних сценаріїв під назвою Blueprint. Blueprint дозволяє розробникам створювати логіку і функціональність ігрового процесу без написання коду. Це робить Unreal Engine чудовим вибором для початківців або тих, хто віддає перевагу візуальному підходу до програмування. Базована на вузлах система Blueprint є інтуїтивно зрозумілою та легкою для вивчення, що дозволяє розробникам швидко створювати прототипи та повторювати свої ігрові ідеї [11].

Unity, з іншого боку, використовує C# як основну мову сценаріїв. C# – потужна та широко використовувана мова програмування в індустрії розробки ігор. З C# розробники мають більше контролю та гнучкості над кодом своєї гри, дозволяючи створювати складніші та налаштовані механізми ігрового

процесу. Хоча C# може мати крутішу криву навчання порівняно з Blueprint, він пропонує більшу універсальність і є цінним навиком, яким потрібно володіти в галузі.

Обидва механізми надають варіанти для сценаріїв і програмування, Unreal Engine пропонує візуальну систему сценаріїв, а Unity використовує широко поширену мову C#. Зважайте на свій досвід програмування та вподобання, коли вибираєте між ними [12].

3.1.4 Asset Store and Marketplace

Unreal Engine's Marketplace – це скарбниця ресурсів, інструментів і плагінів, створених спільнотою Unreal Engine. Від 3D-моделей і анімації до звукових ефектів і систем частинок, Marketplace пропонує широкий вибір ресурсів, які можуть заощадити час і зусилля розробників при створенні своїх ігор. Якість і кількість ресурсів, доступних на Unreal Engine Marketplace, вражають, що робить його цінним ресурсом для розробників.

Магазин асетів Unity також надає велику колекцію активів, починаючи від 2D-спрайтів і закінчуючи повними шаблонами ігор. Asset Store пропонує розробникам зручний спосіб покращити свої ігри за допомогою готових ресурсів, заощаджуючи час і зусилля в процесі розробки. Завдяки великій та активній спільноті Asset Store постійно оновлюється новими та високоякісними активами [12].

І Unreal Engine Marketplace, і Unity Asset Store пропонують цінні ресурси для розробників, надаючи велику кількість активів та інструментів для покращення процесу розробки ігор.

3.1.5 Інтерфейс користувача та редактор

Редактор Unreal Engine відомий своєю надійністю та потужними функціями. Після запуску двигуна користувачі вітаються з візуально

приголомшливим інтерфейсом користувача, який може бути приголомшливим на перший погляд. Однак, як тільки ви звикнете до макета, ви побачите, що він пропонує безліч інструментів і опцій для створення приголомшливих ігор.

Редактор розділений на кілька панелей, кожна з яких виконує певну мету. Головне вікно перегляду дозволяє переглядати ваш ігровий світ і взаємодіяти з ним у режимі реального часу, забезпечуючи безперебійний досвід. Панель браузера вмісту дозволяє впорядковувати та імпортувати активи, а редактор схем дозволяє створювати складну логіку гри за допомогою системи на основі вузлів. З іншого боку, редактор рівнів дає змогу легко проектувати та створювати рівні гри [11].

Хоча спочатку редактор Unreal Engine може здатися приголомшливим, він пропонує неперевершену гнучкість і потужність. Опанувавши це, ви зможете розкрити свій творчий потенціал і втілити свої ігрові ідеї в життя.

Редактор Unity має більш спрощений підхід, що робить його ідеальним для початківців. Після запуску Unity вас зустріне чистий та інтуїтивно зрозумілий інтерфейс користувача, який зосереджується на простоті та легкості використання. Редактор поділено на кілька вкладок, кожна з яких призначена для певного аспекту розробки гри.

Перегляд сцени дозволяє вам переглядати свій ігровий світ і керувати ним, а панель ієрархії забезпечує організований перегляд усіх ігрових об’єктів у вашій сцені. З іншого боку, панель інспектора дозволяє змінювати властивості вибраного об’єкта гри. Unity також пропонує потужне сховище активів, де ви можете переглядати та завантажувати широкий спектр ресурсів, щоб покращити свою гру.

Редактор Unity розроблений для початківців, що дозволяє швидко осягнути основи розробки ігор. Він забезпечує плавну криву навчання, дозволяючи початківцям розпочати роботу, не відчуваючи себе приголомшеними. Однак варто зазначити, що простота Unity може коштувати деяких розширених функцій, які пропонує Unreal Engine [12].

3.1.6 Навчання для початківців

Система blueprint Unreal Engine – це візуальний інструмент сценаріїв, який дозволяє початківцям створювати складну логіку гри без написання жодного рядка коду. Він використовує інтерфейс на основі вузлів, де ви можете підключати вузли, щоб створити представлення логіки гри, схоже на блок-схему. Завдяки цьому новачкам надзвичайно легко зрозуміти та змінити поведінку гри.

Система blueprint пропонує широкий спектр готових вузлів, які охоплюють різні аспекти розробки ігор, такі як фізика, анімація та штучний інтелект. Це дозволяє початківцям швидко створювати прототипи та повторювати свої ідеї без потреби у глибоких знаннях програмування. Однак варто зазначити, що у міру того, як ваш проект стає складнішим, ви можете виявити, що система планів обмежена, і вам може знадобитися зануритися в код двигуна C++ для більш розширеної функціональності.

Загалом система креслень Unreal Engine є потужним інструментом для початківців, що дозволяє їм створювати складну логіку гри без необхідності програмування. Він пропонує легку криву навчання та забезпечує міцну основу для тих, хто хоче глибше зануритися в розробку ігор [11].

Зручний для початківців інтерфейс Unity розроблено з урахуванням початківців, що забезпечує плавне навчання. Двигун надає широкий спектр зручних для початківців посібників і документації, що полегшує новачкам почати роботу. Unity також має активну спільноту, де початківці можуть звернутися за допомогою та порадами до досвідчених розробників.

Однією з видатних особливостей Unity є інструмент створення візуальних сценаріїв під назвою Playmaker. Playmaker дозволяє початківцям створювати ігрову логіку за допомогою системи на основі вузлів, схожої на систему планів Unreal Engine. Завдяки цьому новачкам надзвичайно легко зрозуміти та змінити поведінку гри без необхідності програмування.

Хоча зручні для початківців інтерфейс та інструменти Unity роблять його чудовим вибором для початківців, варто зазначити, що у міру того, як ваш проект стає складнішим, вам може знадобитися зануритися в кодування та сценарії, щоб отримати розширену функціональність. Однак добре задокументований API Unity та широка підтримка спільноти роблять перехід від візуальних сценаріїв до кодування безпроблемним [12].

3.1.7 Продуктивність і оптимізація

Unreal Engine, розроблений Epic Games, відомий своїми передовими можливостями рендерингу, тоді як Unity, створений Unity Technologies, зосереджений на наданні широкого спектру інструментів для оптимізації продуктивності.

Отже, після аналізу можливостей і характеристик двох потужних рушіїв для розробки ігрових застосувань було прийнято рішення використовувати як платформу для реалізації проекту Unity 2Д [12].

3.2 Опис основних елементів гри

Здоров'я головного герою: здоров'я, яке зменшується, коли вороги наносять урон папуги Pio. Якщо воно падає до нуля, віднімається одне життя, а гра починається з контрольної точки.

Життя головного герою: персонаж має кілька життів. Якщо втратити усі, гру доведеться розпочинати з самого початку.

Вороги: рядові вороги – моби, які поводяться агресивно, коли гравець наближається занадто близько. Але й без цього вони переміщуються певним напрямком, тому гравець обирає, як минути супротивника: обійти, знищити або поглинути.

Міні боси: різновід ворогів між рядовими мобами та босами. Деякі з них будуть опціонними, тобто не обов'язковими задля проходження. Матимуть

власні здібності, які можуть іноді повторюватись, нагадуватимуть когось з рядових ворогів. За їхнє знищення гравець отримуватиме певні бонуси. На відміну від рядових ворогів агресивні до папуги Ріо на будь-якій дистанції.

Боси: головні вороги у гровому процесі. Мають унікальні здібності, проходяться індивідуальними методиками. На відміну він міні-босів та рядових ворогів відповідно є обов'язковими задля проходження, аби гравець просувався у грі, й поводиться агресивно до гравця на будь-якій дистанції.

Світи (біоми): різновид локацій. Кожна з локацій належить ворогам певного типу та своєму босові.

Секрети: різноманітні бонуси, які може знайти гравець, застосувавши кмітливість та уважність. Наприклад, альтернативний метод проходження рівня, опціонний міні-бос, кімната з додатковим життям або унікальною здібністю. Можливо комбінувати ці бонуси.

Глобальний секрет: спосіб, який дозволяє відкриту таємний щасливий фінал історії. Задля отримання такого кінця, потрібно уважно досліджувати рівні та проходити різноманітні головоломки й додаткових босів, щоб отримати особливу здібність, яка дозволить пройти боса світу, не знищуючи його.

3.3 Опис елементів, які впливають на досвід гравця

1. Здоров'я та життя головного героя:

- здоров'я папуги Ріо впливає на можливість гравця вижити в поєдинках з ворогами. Низький рівень здоров'я може стимулювати гравця більше уникати боїв або використовувати стратегічні прийоми;
- життя папуги Ріо впливає на рівень толерантності гравця до помилок. Втрата всіх життів може посилити відчуття напруги та спонукати гравця до більш обережних дій.

2. Вороги розширюють можливості для стратегічного мислення гравця. Гравець має розглядати різні методи взаємодії з ними, від обходу до боротьби, в залежності від умов та власних навичок. Різноманітність ворогів збагачує геймплей та стимулює гравця до адаптації та вдосконалення власної стратегії.

3. Міні-боси та боси:

- поява міні-босів розширює геймплей та створює додаткові виклики для гравця. Перемога над ними може надати гравцеві бонуси, що стимулює його до дослідження та вдосконалення навичок;
- головні боси представляють собою кульмінаційні битви, які вимагають від гравця повного використання його знань та навичок. Перемога над ними відзначається відчуттям досягнення та задоволення.

4. Світи (біоми) та секрети:

- різноманітність світів та секретів робить гру більш цікавою та неочікуваною. Гравець має можливість досліджувати різноманітні локації та виявляти секрети, що стимулює його активність та увагу;
- відкриття глобального секрету може вплинути на фінальний результат гри, надаючи гравцеві особливе задоволення та відчуття досягнення.

Таким чином, елементи гри взаємодіють між собою, створюючи насичений та захоплюючий досвід для гравця.

3.4 Діаграма станів головного героя

Діаграма станів (State Machine Diagram) для гри-платформера є корисним інструментом для моделювання різних станів гри та переходів між ними. У грі-платформері є кілька основних станів для головного персонажа, таких як стояння, ходіння, стрибки, атака тощо.

Основними станами для юніта попуги Pio є:

1. Idle (Очікування): персонаж стоїть на місці.
2. Walking (Ходьба): персонаж рухається вліво або вправо.
3. Jumping (Стрибок): персонаж здійснює стрибок.
4. Falling (Падіння): персонаж падає вниз під час поразки.
5. Attacking (Атака): персонаж здійснює атаку (кидання яблуком).
6. Taking Damage (Отримання пошкодження): персонаж отримує пошкодження від ворога або перешкоди.
7. Dying (Смерть): персонаж гине (віднімається одне життя з 3х можливих).

На рис. 3.3 представлено діаграму станів для головного героя.

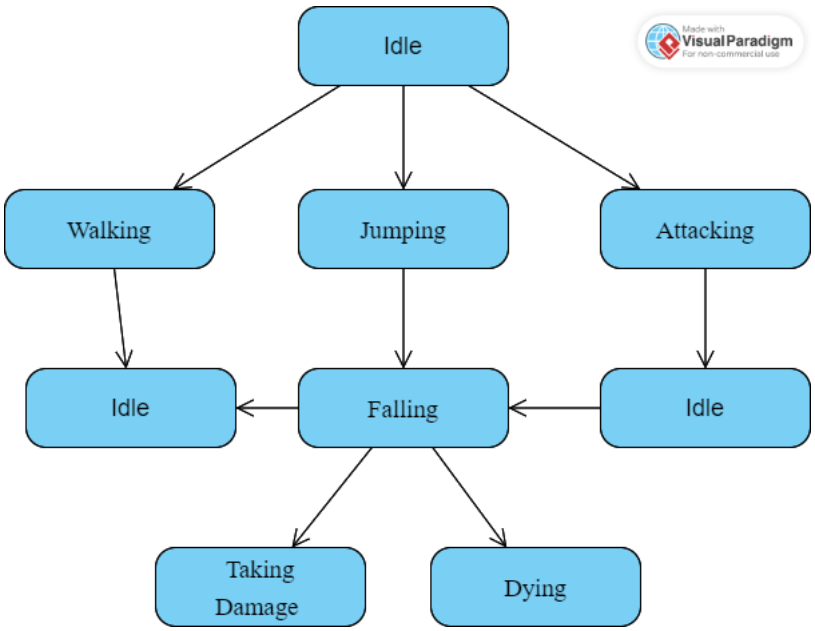

Рисунок 3.3 – Діаграма станів головного героя

Переходи між станами:

- «Idle -> Walking»: натискання клавіші для руху (вліво або вправо);
- «Walking -> Idle»: відпускання клавіші для руху;
- «Walking -> Jumping»: натискання клавіші для стрибка під час руху;
- «Idle -> Jumping»: натискання клавіші для стрибка;

- «Jumping -> Falling»: досягнення піку стрибка та початок падіння;
- «Falling -> Idle» : персонаж досягає землі/нижньої точки платформи;
- «Walking -> Attacking»: натискання клавіші для атаки під час руху;
- «Idle -> Attacking»: натискання клавіші для атаки;
- «Attacking -> Idle»: авершення анімації атаки;
- «Jumping->Taking Damage»: отримання пошкодження під час стрибка;
- «Falling->Taking Damage»: отримання пошкодження під час падіння;
- «Taking Damage->Idle»: завершення анімації отримання пошкодження, персонаж повертається в стан очікування.
- «Taking Damage -> Dying»: отримання критичного пошкодження (здоров'я персонажа зменшується до нуля);
- «Dying -> Idle»: перезавантаження гри або рівня, персонаж відроджується.

4 ПРОГРАМНА РЕАЛІЗАЦІЯ ІГРОВОГО ЗАСТОСУНКУ

4.1 Основні механіки і ігровий баланс

Ігрова механіка – це набір правил, систем та процедур, які визначають, як гравець взаємодіє з грою. Ігрова механіка є фундаментальною частиною дизайну гри, оскільки вона визначає ігровий процес, завдання, які ставляться перед гравцем, і способи їх вирішення.

Основні механіки гри:

1. Рух та навігація: гравець керує персонажем, направляючи його вліво або вправо. Задля переходу між локаціями персонаж проходитиме крізь портал.
2. Бойова система:
   - гравець має можливість атакувати своїх ворогів методом бросків накоплених яблук;
   - персонаж гравця може отримувати урон від атак ворогів, що зменшуватиме його здоров'я.
3. Збирання ресурсів – гравець повинен збирати яблука, щоб мати можливість боротись із ворогами.
4. Накопичення балів – механіка дозволяє шляхом збору яблук отримувати бали. Якщо гра буде пройдена, то гравець збереже свої бали у рекорд якщо немає рекорду із більшою кількістю балів.

Баланс гри – це концепція в геймдизайні, що стосується створення умов, при яких гра є справедливою, викликає інтерес і забезпечує задоволення для гравців. Досягнення балансу в грі означає, що різні елементи гри (персонажі, рівні, зброя тощо) перебувають у гармонії, забезпечуючи відповідний рівень складності і виклику для гравців.

Баланс досягається шляхом розробки декількох видів складності: легка, середня та складна. При обиранні кожної окремої складності гри скрипт передає певні параметри із новими значеннями, а саме:

- швидкість переміщення дерева та боса;

- швидкість пересування гравця;
- шанс появи отруйного яблука;
- шанс появи яблука, що відновлює здоров'я;
- інтервал появи нових елементів;
- кількість здоров'я головного персонажу;
- отримання балів за одне яблуко.

```
 case "easy":
       setDifficultyParamerts(5f,10f,0.15f,0.05f,1f,6f,1);
       break;

case "middle":
     setDifficultyParamerts(7.5f,15f,0.3f,0.025f,0.7f,4f,2);
       break;

case "hard":
     setDifficultyParamerts(10f,20f,0.6f,0.0125f,0.5f,2f,3);
```

Вище зазначено частину коду, яка передає вказані параметри.

4.2 Опис інтерфейсу користувача

Графічні елементи гри є ключовими складовими, які визначають візуальний стиль і естетику гри, а також впливають на ігровий досвід гравця. Основні графічні елементи, які використовуються в розробці ігор: спрайти персонажів, елементів меню, декорації, фон та інше.

Основні графічні елементи гри формують візуальну складову, яка є критично важливою для занурення гравця у світ гри. Вони включають спрайти, фони, інтерфейс, анімації, текстури, шрифти, ефекти освітлення та камери. Кожен з цих елементів вимагає ретельної розробки та гармонійного поєднання для створення привабливої та функціональної гри.

Для головного героя – попуги Pio, необхідно мати ассет спрайтів для створення анімації руків птаха (рис. 4.1). Спрайти взяти з Unity AssetStore, де вся інформація предоставляється у вільному доступі.

Рисунок 4.1 – Ассет спрайтів для головного героя Pio

Для формування меню гри були обрані наступні спрайти: рекорд за балами (рис. 4.2а), набрані бали гравцем (рис. 4.2б):

а) б)

Рисунок 4.2 – Спрайти для позначки балів

Кількість спійманих яблук попугом та рівень його життєвих силвідображаються спрайтами, які представлено на рисунку 4.3.

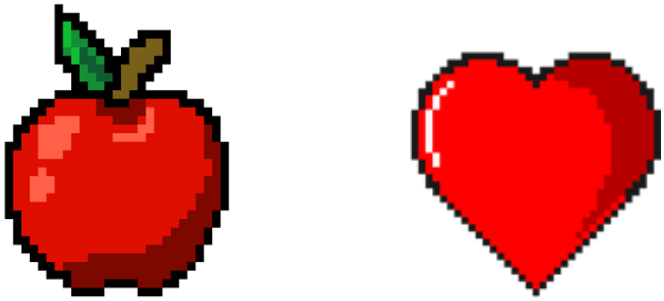

Рисунок 4.3 – Спрайти життя та ресурсів попуги

Спрайт гнилого яблука, що буде наносити урок папуги та спрайт життєвого еліксиру представлено на рисунку 4.4.

Рисунок 4.4 – Спрайти гнилого яблука та еліксиру

Скрін головного меню представлено на рисунку 4.5:

Рисунок 4.5 – Кнопки головного меню ігрового додатку

Кнопка «Settings» дозволяю гравцю налаштувати вікно додатку за власними потребами (рис. 4.6). На рисунку 4.7 представлено HUD: інформаційну панель, яка відображає життєві показники персонажа, кількість бонусів, рівень тощо. В верхньому лівому куту екрану розміщено кнопку «Pause» для зручності гравця.

Отже, після налаштувань гравцю відкривається перший рівень гри. попуга Ріо повинен як можна більше зібрати яблук, слідкувати за переміщенням чарівного дерева та появою ворогів. На старті рівень бонусів та

кількість яблук нульова, healthbar життєвих ресурсів героя на максимальній позначці.

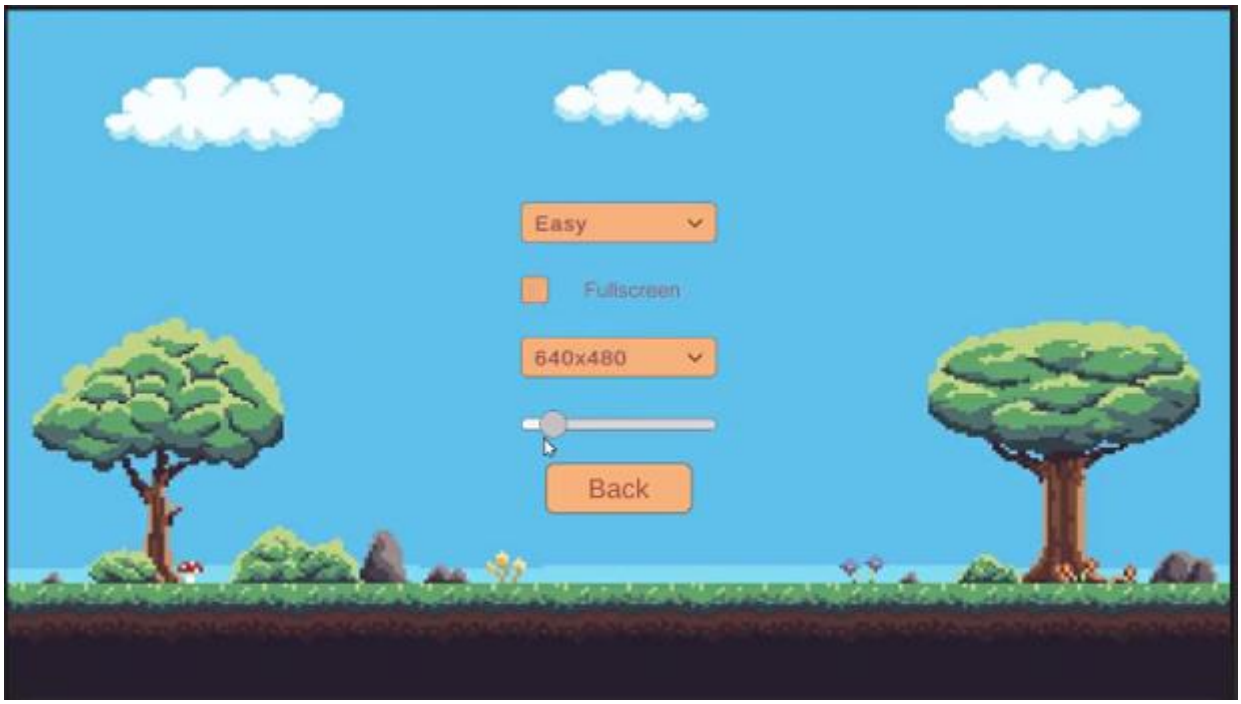

Рисунок 4.6 – Налаштування ігрового додатку та рівня складності

Рисунок 4.7 – Ігрове меню

Коли кількість спійманих яблук зростає – рівень починає ускладнюватись шляхом прискорення появи нових елементів, таким чином ухилятись від гнилого яблука стає складніше (рис. 4.8). Дерево постійно змінює напрям свого руху.

Рисунок 4.8 – Поява отруєного яблука

При кожній події на сцені необхідно викликати методи оновлення HUD:

```
public class PlayerController : MonoBehaviour
{
    public HUDController hudController;
    void Start()
    {
        hudController.UpdateLives(3);
        hudController.UpdateBonuses(0);
        hudController.UpdateLevel(1);
    }
    void TakeDamage()
    {
        // Зменшення кількості життів
        hudController.UpdateLives(hudController.lives - 1);
    }
    void CollectBonus()
    {
```

```
            // Збільшення кількості бонусів
            hudController.UpdateBonuses(hudController.bonuses + 100);
        }
}
```

Коли герою вдається зібрати необхідну кількість яблук (10 одиниць) алгоритм гри генерує появу еліксиру, який відновлює його життєві ресурси (рис. 4.9). При завершенні місії відкривається портал, який переміщує героя на наступний рівень, де він буде боротися проти боса.

Рисунок 4.9 – Поява еліксиру та відкриття порталу

Для підтримки балансу гри є декілька босів, які чекають на головного героя у кінці кожного левела. На даний час реалізовано чотири рівня, для кожного з яких є свій бос. Вони відрізняються зовнішнім виглядом та рівнем власної сили. З Кожним новим випробуванням гравцю стає складніше одержувати перемогу, але гра побудована таким чином, щоб навіть при поразки гравець мав можливість почати гру з останнього рівня.

На рисунку 4.10 представлено скрін ігрової сцени, де з’являється бос першого рівня. Його задача схопити попугу Pio, який намагається обходити його пастки. Якщо гравець перемагає і healthbar рівня життєвих сил боса обнуляється, в грі висвічується вікно з надписом перемоги (рис. 4.11) і гравець може або вийти з гри, або в меню обрати наступний левел для проходження.

Рисунок 4.10 – Скрін битви з босом першого рівня

Рисунок 4.11 – Вікно з повідомленням про перемогу гравця

4.3 Програмна реалізація основних механік

Розглянемо детально реалізацію основних функцій ігрового застосунку. Один з головних ігрових юнітів – це яблука, в яких однаковий алгоритм руху. Він описан у файлі проекту «Apple.cs»: тут знадобиться статична змінна, що визначає нижню межу по осі Y «bottomY» та змінні для спрайтів самих яблук.

```
Apple.cs:
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;

public class Apple : MonoBehaviour
{
     public static float bottomY = -20f;

         // Змінні для спрайтів яблук
    public SpriteRenderer spriteRenderer;
    public Sprite applePoisonedSprite;
    public Sprite appleHealingSprite;
```

Прапорці для визначення, чи яблуко отруйне чи лікувальне:

```
    public bool isPoisoned;
    public bool isHealing;
    void Start()
    {
        // Отримуємо компонент SpriteRenderer
        spriteRenderer = gameObject.GetComponent<SpriteRenderer>();

        // Встановлюємо спрайт яблука відповідно до його властивостей
        if(isPoisoned == true) {
            spriteRenderer.sprite = applePoisonedSprite;
        } else if(isHealing == true) {
            spriteRenderer.sprite = appleHealingSprite;
        }
    }
```

Далі йде виклик кожного кадру, якщо яблуко виходить за межі екрану знизу чи зверху (це платформер – тож дуже важливо перевіряти межі) та

яблуко є отруйним або лікувальним, викликаємо метод «AppleDestroyed» у класі «ApplePicker».

```
    void Update()
    {
        // Якщо яблуко виходить за межі екрану знизу
        if(transform.position.y < bottomY) {

        // Якщо яблуко не є отруйним або лікувальним, викликаємо метод
AppleDestroyed у класі ApplePicker
            if(isPoisoned != true && isHealing != true) {
                ApplePicker applePicker =
Camera.main.GetComponent<ApplePicker>();
                applePicker.AppleDestroyed();
            }

            // Видаляємо яблуко з екрану
            Destroy(this.gameObject);
        }
        // Якщо яблуко виходить за межі екрану зверху
        if(transform.position.y > -bottomY) {
            // Видаляємо яблуко з екрану
            Destroy(this.gameObject);
        }
    }
}
```

Наступний файл «ApplePicker.cs» включає алгоритм перевірки кількості спійманих яблук та оновлення стану життєвих ресурсів.

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

public class ApplePicker : MonoBehaviour
{
    [Header("Set in inspector")]

    // Префаб кошика
    public GameObject basketPrefab;
    // Кількість кошиків
    public int basketCount = 1;
    // Нижня межа по осі Y для кошиків
    public float basketBottomY = -9f;
    // Відстань між кошиками по осі Y
    public float basketSpacingY = 2f;
    // Список кошиків
    public List<GameObject> basketList;
```

```
    void Start()
    {
        // Створюємо вказану кількість кошиків
        for(int i = 0; i < basketCount; i++) {
            GameObject tBasketGo =
Instantiate<GameObject>(basketPrefab);
            Vector3 pos = Vector3.zero;
            pos.y = -7.5f;
            tBasketGo.transform.position = pos;
            basketList.Add(tBasketGo);
        }
    }
```

Наступна функція викликається, коли яблуко знищено: знаходимо всі яблука за тегом і знищуємо їх, шкала здоров’я зменшується на одну позицію:

```
  public void AppleDestroyed()
    {
         GameObject[] appleArray =
GameObject.FindGameObjectsWithTag("AppleTag");
        foreach(GameObject apple in appleArray) {
            Destroy(apple);
        }
        // Зменшуємо шкалу здоров'я на 1
        HealthBar.takeDamage(1f);
    }
}
```

Файл «AppleTree.cs» містить реалізацію механіки дерева з яблуками:

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

public class AppleTree : MonoBehaviour
{
    [Header("Set in inspector")]
       // Префаб яблука
    public GameObject applePrefab;
```

Тут необідні змінні для швидкість руху дерева, межі руху та зміна ймовірності зміни напрямку руху дерева:

```
    public float movementSpeed;
```

```
    // Ліва і права межі руху
    public float leftAndRightEdge = 16f;
    // Ймовірність зміни напрямку руху
    public float chanceToChangeDirection = 0.02f;
```

Отруєне яблуко буде випадати випадково як і інтервал появи такого об’єкту:

```
    public float chanceToSpawnPoisonedApple;
    // Ймовірність появи лікувального яблука
    public float chanceToSpawnHealingApple;
    // Інтервал між випаданням яблук
    public float secondsBetweenAppleDrops;
```

У методі Start() встановлюємо параметри дерева та викликаємо метод DropApple() через 2 секунди:

```
void Start()
    {
            setTreeParametrs(
             PlayerPrefs.GetFloat("treeMovementSpeed"),
             PlayerPrefs.GetFloat("chanceToSpawnPoisonedApple"),
             PlayerPrefs.GetFloat("chanceToSpawnHealingApple"),
             PlayerPrefs.GetFloat("secondsBetweenAppleDrops")
        );
        Invoke("DropApple", 2f);
    }
```

Далі викликається кожен кадр та виконується рух дерева. Якщо дерево дійшло до лівої або правої межі, змінюємо напрямок руху:

```
 void Update()
    {
        Vector3 position = transform.position;
        position.x += movementSpeed * Time.deltaTime;
        transform.position = position;

        if(position.x < -leftAndRightEdge) {
            movementSpeed = Mathf.Abs(movementSpeed);
        } else if(position.x > leftAndRightEdge) {
            movementSpeed = -Mathf.Abs(movementSpeed);
        } }
void FixedUpdate()
```

```
    {//Якщо випадкове значення менше ймовірності зміни напрямку
руху, змінюємо напрямок

        if(Random.value < chanceToChangeDirection) {
            movementSpeed *= -1;
        }
    }
```

Метод для випадання яблук DropApple():

```
  private void DropApple()
    {
       GameObject apple = Instantiate<GameObject>(applePrefab);
        apple.transform.position = transform.position;

        // Генеруємо випадкове значення
        float randomValue = Random.value;
        Apple tApple = apple.GetComponent<Apple>();

        // Встановлюємо властивості яблука відповідно до
випадкового значення
        if(randomValue <= chanceToSpawnHealingApple) {
            tApple.isHealing = true;
        } else if (randomValue >= chanceToSpawnHealingApple &&
            randomValue <= chanceToSpawnHealingApple +
chanceToSpawnPoisonedApple) {
            tApple.isPoisoned = true;
        }
```

Викликаємо метод DropApple() через певний інтервал часу та встановлюємо параметри дерева:

```
        Invoke("DropApple", secondsBetweenAppleDrops);
    }
   private void setTreeParametrs
    (float movementSpeed, float chanceToSpawnPoisonedApple,
    float chanceToSpawnHealingApple, float
secondsBetweenAppleDrops)
    {
        this.movementSpeed = movementSpeed;
        this.chanceToSpawnPoisonedApple =
chanceToSpawnPoisonedApple;
        this.chanceToSpawnHealingApple =
chanceToSpawnHealingApple;
        this.secondsBetweenAppleDrops =
secondsBetweenAppleDrops;
    }}
```

«Basket.cs» містить логіку для підрахунків бонусів, з урахуванням параметрів атаки:

«Basket.cs»

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

public class Basket : MonoBehaviour
{
    [Header("Set parameters")]
```

Тут знадобиться фактор збільшення рахунку та швидкість руху:

```
    public int increaseFactor;
    public float movementSpeed;

    // Текстове поле для відображення рахунку
    public TextMeshProUGUI scoreGT;
    public int score = 0;           // Рахунок
    private SpriteRenderer sprite; // Компонент спрайту
    public string direction;       // Напрямок руху кошика
    public float attackCooldown;    // Час відновлення атаки
    private float coolDownEndTime = -Mathf.Infinity;
    public bool canAttack;         // Можливість атаки
    public int applesAmount;        // Кількість яблук
    public GameObject applePrefab; // Префаб яблука

    // Текстове поле для відображення кількості яблук
    private TextMeshProUGUI bulletsText;
    private GameObject bulletsTextGO;
    private GameObject appleCooldownImageGO;
    private Image appleCooldownImage;
    public float leftAndRightEdge;   // Ліва та права межі руху

  private void Start()
    {
        GameObject scoreGO = GameObject.Find("CurrentScore");
        scoreGT = scoreGO.GetComponent<TextMeshProUGUI>();
        scoreGT.text = "0";
        movementSpeed =
PlayerPrefs.GetFloat("playerMovementSpeed");
        increaseFactor = PlayerPrefs.GetInt("scoreIncrease");
        attackCooldown = 2f;
```

```
        leftAndRightEdge = 18f;
        applesAmount = PlayerPrefs.GetInt("PlayerApplesAmount");
        sprite = this.GetComponent<SpriteRenderer>();

        PlayerPrefs.DeleteKey("PlayerCanAttack");
        if (SceneManager.GetActiveScene().name == "Boss Scene")
        {
            canAttack = true;
            bulletsTextGO = GameObject.Find("AppleAmount");
            bulletsText =
bulletsTextGO.GetComponent<TextMeshProUGUI>();
            bulletsText.text = applesAmount + "";

            appleCooldownImageGO =
GameObject.Find("AppleImgBg");
            appleCooldownImage =
appleCooldownImageGO.GetComponent<Image>();
            appleCooldownImage.fillAmount = 0f;

            score = PlayerPrefs.GetInt("PlayerCurrentScore");
            scoreGT.text = "" + score.ToString();
        }
    }
```

В грі передбачено управління за допомогою стрілок або кнопок руху:

```
  void Update()
    {
        if (Input.GetKey(KeyCode.RightArrow) &&
this.transform.position.x < leftAndRightEdge)
        {
            this.transform.position += Vector3.right *
movementSpeed * Time.deltaTime;
            if (Time.timeScale == 1f) sprite.flipX = false;
            direction = "right";
        }
        else if (Input.GetKey(KeyCode.LeftArrow) &&
this.transform.position.x > -leftAndRightEdge)
        {
            this.transform.position += Vector3.left *
movementSpeed * Time.deltaTime;
            direction = "left";
            if (Time.timeScale == 1f) sprite.flipX = true;
        }

        if (SceneManager.GetActiveScene().name == "Boss Scene")
        {
            if (Time.time < coolDownEndTime)
            {
                float elapsed = coolDownEndTime - Time.time;
```

```
                float percent = elapsed / attackCooldown;

                appleCooldownImage.fillAmount = percent;
            }
            else
            {
                appleCooldownImage.fillAmount = 0f;
            }
        }

        if (Time.time >= coolDownEndTime && canAttack == true &&
applesAmount > 0 && Input.GetKeyDown(KeyCode.Space))
        {
            DropApple();
            coolDownEndTime = Time.time + attackCooldown;
            applesAmount--;
            bulletsText.text = applesAmount + "";
        }
        scoreGT.text = "" + score.ToString();
    }

    void OnCollisionEnter(Collision coll)
    {
        GameObject collidedWith = coll.gameObject;
        Apple tApple = collidedWith.GetComponent<Apple>();

        if (collidedWith.tag == "AppleTag")
        {
            if (tApple.isPoisoned == true)
            {
                ApplePicker applePicker =
Camera.main.GetComponent<ApplePicker>();
                applePicker.AppleDestroyed();
            }
            else if (tApple.isHealing == true)
            {
                Destroy(collidedWith);
                HealthBar.heal(1f);
            }
            else
            {
                Destroy(collidedWith);
                increasePoints();
                Levels.currentKeys += 1;
                PlayerPrefs.SetInt("PlayerApplesAmount",
Levels.currentKeys);
           } }
```

Метод для збільшення рахунку:

```
  public void increasePoints()
    {
        score += 1 * increaseFactor;
        scoreGT.text = "" + score.ToString();

        if (score > HighScore.highScore)
        {
            HighScore.highScore = score;
        }
    }
```

Метод для випадання яблук:

```
  private void DropApple()
    {
        GameObject apple = Instantiate<GameObject>(applePrefab);
        Rigidbody appleRigidbody =
apple.GetComponent<Rigidbody>();
        apple.transform.position = transform.position + new
Vector3(0, 3f, 0);
        appleRigidbody.useGravity = false;
        appleRigidbody.AddForce(0, 20f, 0, ForceMode.Impulse);
        Apple tApple = apple.GetComponent<Apple>();
    }
}
```

Файл «Boss.cs» містить реалізацію механіки босу: тут необхідні змінні інтервалу атаки боса, час останньої атаки. Крім цього, поточне та максимальне здоров'я боса для розрахунку умови перемоги над ворогом головного героя:

```
    public float attackCooldown; // Інтервал атаки
    private float lastAttackTime = -Mathf.Infinity;    // Час
останньої атаки
    private SpriteRenderer sprite;    // Компонент спрайту
    private GameObject player; // Гравець (кошик)
    public static float currentHealth, maxHealth;
    public Image hpImage; // Зображення для відображення
здоров'я
    public Animator animator;
    public GameObject menu;    // Меню закінчення гри
    public float firstAttackTime; // Час першої атаки
```

Метод для встановлення шкали здоров'я setHealthBar() та метод для завдання пошкодження takeDamage():

```
    public void setHealthBar()
    {
        hpImage.fillAmount = currentHealth / maxHealth * 1f; }
    public static void takeDamage(float damage)
    {
        currentHealth -= damage;
    }
```

Обробник зіткнень:

```
  void OnCollisionEnter(Collision coll)
    {
        GameObject collidedWith = coll.gameObject;
        Apple tApple = collidedWith.GetComponent<Apple>();
        if (collidedWith.tag == "AppleTag" && tApple.isPoisoned
== false)
        {
            Destroy(collidedWith);
            takeDamage(1f);
        }
    }
```

«GameOverMenu.cs» описує логіку роботи головного меню, а саме реакції на події в ігровій сцені. Ту використовується панель меню, масив кнопок паузи,

```
     public class GameOverMenu : MonoBehaviour
     {
         public GameObject menu;
          private GameObject[] pausedButtons;
        private bool isPaused = false;

         // Викликається при старті
         void Start()
         {
             menu.SetActive(false);
         }
```

Якщо здоров'я закінчилося, йде вимкнення кнопок паузи та скидання кількості яблук гравця:

```
      void Update()
         {
             if (HealthBar.currentHealth == 0)
             {
                 pauseGame();
                 menu.SetActive(true);
                 pausedButtons                                    =
GameObject.FindGameObjectsWithTag("PauseButton");
                foreach (GameObject button in pausedButtons)
                 {
                   button.GetComponent<Button>().enabled = false;
                 }

                 PlayerPrefs.SetInt("PlayerApplesAmount", 0);
             }
```

Програмна реалізація методу для паузи гри:

```
       public void pauseGame()
         {
             Time.timeScale = 0f;
         }

          public void resumeGame()
         {
             Time.timeScale = 1f;
         }
     }
```

Останній з важливих механік гри – це оновлення станом healthbar-у («HealthBar.cs»):

```
public class HealthBar : MonoBehaviour
{
          // Поточне та максимальне здоров'я
    public static float currentHealth;
    public static float maxHealth;
           // Зображення для відображення здоров'я
    public Image hpImage;
```

Далі йде Ініціалізація максимального здоров'я зі збережених налаштувань та встановлення шкали здоров'я:

```
    void Start()
```

```
    {
        maxHealth = PlayerPrefs.GetFloat("playerHealth");
        currentHealth = maxHealth;
       setHealthBar();   }
```

Для оновлення шкали здоров’я та встановлення її значень використовуються наступні методи:

```
 void Update()
    {
        setHealthBar();
    }
    public void setHealthBar()
    {
        hpImage.fillAmount = currentHealth / maxHealth * 1f;
    }
    // Метод для завдання пошкодження
    public static void takeDamage(float damage)
    {
        currentHealth -= damage;
    }
    // Метод для відновлення здоров'я
    public static void heal(float healingAmount)
    {
        if (currentHealth + healingAmount > maxHealth)
        {
            currentHealth = maxHealth;
            return;
        }
        currentHealth += healingAmount;
    }
}
```

ВИСНОВКИ

За результати дипломної роботи досягнена поставлена мета, а саме розроблен захоплюючий, цікавий та інтуїтивно зрозумілий ігровий процес з викликами та головоломками, які сприятимуть підтримки зацікавленості гравців. Унікальний казковий світ папуги Ріо, що привертає увагу своєю неповторною атмосферою та привабливими істотами, в тому числі різноманітність ворогів, босів та локацій, що створює динамічний та захоплюючий геймплей. Елементи загадок та секретів, які заохочують гравця досліджувати світ гри та знаходити приховані скарби. Контраст між казковим світом та ігровими можливостями залишає потужний слід у свідомості гравця.

Під час виконання дипломної роботи були пройдені всі етапи розробки ігрового застосунку, а саме визначені вимоги до ігрового застосунку шляхом побудови діаграми варіантів використання, описано концепцію гри, обрані інструментальні засоби розробки, виконана декомпозиція робіт та побудовано діаграму Ганта, крім цього описані ризики розробки ігрового застосунку.

На етапі проектування описані основні елементи гри та побудовано діаграму станів для головного героя. Для реалізації застосунку використовувався рушій Unity та мова програмування C# . Після завершення розробки ігровий застосунок протестовано на коректність роботу всіх функцій.

СПИСОК ВИКОРИСТОВАНИХ ДЖЕРЕЛ

1. Why 2d-games still popular? URL: https://retrostylegames.com/blog/are-2d-games-still-popular/ (дата звернення 15.04.2024)
2. Top 2ed-games. URL: https://retrostylegames.com/blog/are-2d-games-best/ (дата звернення 15.04.2024)
3. What made those old, 2D platformers so great? URL: https://www.gamedeveloper.com/design/what-made-those-old-2d-platformers-so-great-#close-modal (дата звернення 22.04.2024)
4. The best retro-games. URL: https://retrododo.com/best-2d-platform-games/ (дата звернення 22.04.2024)
5. Why 2d-games so popular? URL: https://retrododo.com/why-2d-so-popular/ (дата звернення 29.04.2024)
6. Що таке функціональні вимоги: приклади, визначення, повний посібник. URL: https://visuresolutions.com/uk/функціональні-вимоги/ (дата звернення 30.04.2024)
7. What is Use Case Diagram? URL: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-use-case-diagram/ (дата звернення 05.05.2024)
8. Work Breakdown Structure (WBS). URL: https://www.projectmanager.com/guides-work-breakdown-structure (дата звернення 05.05.2024)
9. Що таке Діаграма Ганта і якправильно користуватися? URL: https://worksection.com/ua/blog/what-is-gantt-chart.html (дата звернення 05.05.2024)
10. Діаграма Ішікави для управління ризиками. URL: https://www.coopzone.ca/produit/1327579-diagrama-isikavi-dlya-upravlinnya-rizikami (дата звернення 11.05.2024)

11. Comprehensive Comparison of Unreal Engine vs Unity. URL: https://ilogos.biz/comprehensive-comparison-of-unreal-engine-vs-unity/ (дата звернення 12.05.2024)
12. Матеріали на тему «Unreal Engine». URL: https://gamedev.dou.ua/tags/Unreal%20Engine/ (дата звернення 18.05.2024)