Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра програмного забезпечення

БАКАЛАВРСЬКА ДИПЛОМНА РОБОТА

на тему:

Програмна платформа моделювання поведінки суперників у платформерних іграх

Виконав: студент 4 курсу
Групи 1ПІ-20б
Спеціальності
121 – Інженерія програмного забезпечення
(шифр і назва напрямку підготовки спеціальності)
Кушнір М.М.
(прізвище та ініціали)
Керівник: к.т.н., доц. каф. ПЗ Чехместрук Р.Ю.
(прізвище та ініціали)
Рецензент: к.т.н., доц. каф. ОТ Войцеховська О.В.
(прізвище та ініціали)

Допущено до захисту
Завідувач кафедри ПЗ
д.т.н., проф. О.Н. Романюк
(прізвище та ініціали)
«10» червня 2024р.

ВНТУ – 2024

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра програмного забезпечення
Рівень вищої освіти перший (бакалаврський)
Галузь знань 12 «Інформаційні технології»
Спеціальність 121 «Інженерія програмного забезпечення»
Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
Романюк О.Н.
«02» березня 2024р.

ЗАВДАННЯ
НА БАКАЛАВРСЬКУ ДИПЛОМНУ РОБОТУ СТУДЕНТУ

Кушніру Максиму Миколайовичу

1. Тема роботи – Програмна платформа моделювання поведінки суперників у платформених іграх.

Керівник роботи: Чехместрук Роман Юрійович, к.т.н., доц. кафедри ПЗ, затверджені наказом вищого навчального закладу від 11 березня 2024 року №80.

2. Строк подання студентом роботи 3 червня 2024 року.

3. Вихідні дані до роботи: середовище розробки Unity, мова розробки C#. Операційна система – Windows 10, метод аналізу дій користувача та метод поведінки суперників.

4. Зміст текстової частини: аналіз стану питання, аналогів та методів; аналіз розвитку ігрових програм; аналіз аналогів розроблюваного програмного продукту; постановка задачі; розробка інтерфейсу та моделей; аналіз вихідних даних систем; розробка інтерфейсу програмного додатку; розробка 2D моделей; розробка моделей та алгоритмів роботи систем; розробка моделі поведінки суперників; розробка алгоритму поведінки суперників; висновки.

5. Мета, об'єкт та предмет дослідження; новизна отриманих результатів; актуальність розробки; порівняльний аналіз аналогів; UML діаграма; розробка моделі поведінки суперників; розробка алгоритму поведінки суперників; висновки

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв
1-4	Чехместрук Р.Ю., к.т.н., доцент кафедри ПЗ	13.03.24	03.06.24

7. Дата видачі завдання 13 березня 2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської дипломної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз стану питання, аналогів та методів	14.03.2024 – 27.03.2024	Виконано
2	Розробка інтерфейсів та моделей	28.03.2024 – 27.04.2024	Виконано
3	Розробка програмного забезпечення	28.04.2024 – 23.05.2024	Виконано
4	Тестування програми	24.05.2024 – 30.05.2024	Виконано

Студент _____ **М.М. Кушні**
(підпис) (ініціали та прізвище

Керівник бакалаврської дипломної роботи _____ **Р.Ю. Чехместру**
(підпис) (ініціали та прізвище

АНОТАЦІЯ

УДК 004.912.032.26

Кушнір М. М. Розробка програмної платформи моделювання поведінки суперників у платформених іграх: бакалаврська дипломна робота зі спеціальності 121 Інженерія програмного забезпечення, освітня програма – інженерія програмного забезпечення. Вінниця: ВНТУ, 2024. 55 с.

У рамках бакалаврської роботи було здійснено розробку програмної платформи моделювання поведінки суперників у платформених іграх. Було проведено дослідження наявних інструментів для моделювання поведінки персонажів у відеоіграх, оцінено їхні сильні та слабкі сторони, що підкреслило необхідність створення нової платформи.

Розроблена система забезпечує динамічне моделювання поведінки, без використання алгоритму штучного інтелекту.

У процесі роботи було обрано мову програмування та платформу для розробки, з урахуванням їх ефективності та гнучкості у створенні складних ігрових механік. Проект було реалізовано з використанням Unity та C#, що дозволило ефективно інтегрувати штучний інтелект у ігровий процес.

Результати цієї бакалаврської роботи можуть бути використані розробниками ігор для створення більш складних і реалістичних моделей поведінки суперників, що зробить ігровий процес більш захоплюючим та непередбачуваним.

Ключові слова: платформерні ігри, моделювання поведінки, нейронні мережі, машинне навчання.

ABSTRACT

Kushnir M. M. Development of a Software Platform for Modeling Opponents' Behavior in Platform Games: Bachelor's Thesis in Specialty 121 Software Engineering, Educational Program – Software Engineering. Vinnytsia: VNTU, 2024. 55 p.

As part of the bachelor's thesis, a software platform for modeling the behavior of opponents in platform games was developed. Research was conducted on the existing tools for modeling character behavior in video games, assessing their strengths and weaknesses, which highlighted the need for a new platform.

The developed system provides dynamic behavior modeling without using an artificial intelligence algorithm. During the work, a programming language and platform were chosen for development, considering their effectiveness and flexibility in creating complex game mechanics. The project was implemented using Unity and C#, which allowed for the efficient integration of artificial intelligence into the game process.

The results of this bachelor's thesis can be used by game developers to create more complex and realistic models of opponent behavior, making the gameplay more exciting and unpredictable.

Keywords: platform games, behavior modeling, neural networks, machine learning.

ЗМІСТ

ВСТУП

Обґрунтування вибору теми дослідження: В умовах сучасного ринку відеоігор, важливістю набувають реалістичність та інтерактивність ігрових механік, особливо у контексті штучного інтелекту. Ефективне моделювання поведінки опонентів у грах не тільки підсилює занурення у гру, але й створює значні виклики, які приваблюють гравців. Розвиток ігрових алгоритмів штучного інтелекту для супротивників є ключовим елементом для формування глибоких і захоплюючих ігрових досвідів.

Дослідження та застосування передових методів програмування та графічного дизайну в процесі створення платформерних ігор дозволяє комбінувати теоретичні знання та практичні навички, набуті під час навчання у сфері комп'ютерних наук. Обрана тема сприятиме академічному зростанню та підготовці до професійної діяльності в ігровій індустрії.

Завдяки застосуванню технік машинного навчання, розробники можуть створювати більш реалістичних та адаптивних віртуальних супротивників, котрі вчаться і адаптуються до дій гравця[1].

Мета дослідження: Створення детального прототипу платформерної гри, що включає функції моделювання поведінки опонентів. Особлива увага буде зосереджена на розробці інтелектуальних та адаптивних моделей поведінки, здатних взаємодіяти з гравцем, забезпечуючи виклики та залучення в ігровий процес. Завдання дослідження включають:

- Розробку та перевірку алгоритмів для створення багатогранних та переконливих моделей поведінки опонентів.

- Інтеграцію цих алгоритмів в платформерну гру, що полегшить моделювання та тестування поведінки опонентів у різноманітних ігрових ситуаціях.

- Оцінка ефективності розробленої системи через тестування в реальних ігрових умовах.

Зв’язок роботи з науковими програмами, планами, темою. Дослідження, проведене в рамках цієї бакалаврської роботи, тісно пов’язане з актуальними науковими програмами та дослідницькими планами, що реалізуються на факультеті комп'ютерних наук. Проект вписується в ширші наукові ініціативи, спрямовані на розвиток методів штучного інтелекту та їх застосування в різних областях, включаючи розробку відеоігор.

За допомогою сучасних методів можна симулювати складні поведінки, забезпечуючи динамічний ігровий досвід, де НШР реагують і адаптуються залежно від дій гравця[2].

Мета та завдання дослідження. Основною метою дослідження є дослідження є проектування та реалізація платформерної гри, що включає розробку інноваційних методів моделювання поведінки суперників. Це передбачає створення комплексної ігрової системи, здатної забезпечити високий рівень взаємодії та виклику для гравців.

Завданнями дослідження є:

- Огляд та аналіз наявних рішень для моделювання поведінки в іграх, ідентифікація їх сильних та слабких сторін.

- Створення програмних інструментів і інтерфейсів для легкої інтеграції з різними ігровими двигунами.

- Тестування і оцінка розробленої платформи шляхом впровадження у пілотні проекти платформерних ігор.

Об’єктом дослідження є ігровий процес платформерної гри. Особлива увага приділяється механізмам моделювання поведінки суперників. Дослідження зосереджене на вивченні та аналізі ігрових стратегій, які використовуються для створення реалістичної та викликаючої поведінки неігрових персонажів, щоб забезпечити гравцям захоплюючий ігровий досвід.

Предметом дослідження є вивчення алгоритмів штучного інтелекту, застосування методів машинного навчання, а також практичне використання інструментів програмування для розробки ігор. Особлива увага приділяється мові програмування C# та ігровому двигуну Unity, які є основними

інструментами для створення платформерної гри з реалістичною та викликаючою поведінкою суперників.

Методи дослідження. У рамках дослідження розробки програмної платформи для моделювання поведінки суперників у платформерних іграх з використанням штучного інтелекту використовуються наступні методи:

- Аналіз поведінкових патернів: вивчення існуючих стратегій поведінки суперників в іграх для визначення їх ефективності та адаптивності.

- Машинне навчання: розробка та тренування алгоритмів нейронних мереж для генерації складних поведінкових моделей суперників.

- Симуляція та моделювання: створення віртуальних сценаріїв для тестування та вдосконалення поведінкових алгоритмів.

Інноваційність отриманих результатів полягає в наступному:

1. Розвиток алгоритмів здатних динамічно адаптуватися до стратегій гравця, підвищуючи таким чином рівень виклику та занурення в гру;

2. Розробка інтуїтивного і адаптивного графічного інтерфейсу, який буде зручним для користувачів різного віку і досвіду.

Практична цінність отриманих результатів. Практична цінність проекту проявляється у поліпшенні якості ігрового досвіду, забезпечуючи гравцям більш складні та реалістичні виклики. Кінцевий продукт може бути інтегрований у різноманітні ігрові платформи, забезпечуючи гравцям неймовірний занурюючий досвід.

1 АНАЛІЗ СТАНУ ПИТАННЯ, АНАЛОГІВ ТА МЕТОДІВ

1.1 Аналіз розвитку ігрових програм

Останніми роками в ігровій індустрії зросла популярність розробки платформених 2D ігор з реалістичними противниками. Розробка платформи, яка моделює поведінку суперників є важливою, оскільки вона дозволяє глибше розуміти механіки ігрового процесу та реакції суперників на дії гравців. Це досягається за допомогою заздалегідь визначених шаблонів поведінки та алгоритмів, які можуть бути гнучко налаштовані для створення унікальних ігрових викликів.

Основні переваги розробки такої програмної платформи полягають у підвищенні інтерактивності та занурення в ігровий процес. Вона надає розробникам інструменти для детального налаштування поведінки суперників, що дозволяє створювати різноманітні ігрові сценарії без потреби в складних алгоритмах.

Аналізуючи поведінку суперників, платформа допомагає ідентифікувати тенденції та патерни, які можуть бути використані для оптимізації ігрового процесу. Це забезпечує баланс між складністю та доступністю гри для гравців з різним рівнем навичок. Така платформа також сприяє підтримці інтересу гравців протягом усього ігрового процесу, забезпечуючи різноманітність ігрових викликів.

В умовах зростаючої зацікавленості в платформерних іграх, розробка програмних платформ для моделювання поведінки суперників без використання ШІ набуває особливої важливості. Це сприяє створенню ігрових середовищ, де противники використовують визначені стратегії та викликають складні випробування для гравців.

Актуальність розробки такої програмної платформи відображається у наступних пунктах:

1. Зростання інтересу до платформерних ігор: Платформерні ігри приваблюють розробників і гравців завдяки своїй динаміці та вимогам до стратегічного мислення. Цей інтерес підживлює потребу у розвитку суперників, які можуть пропонувати нові та непередбачувані виклики, збагачуючи ігровий досвід.

2. Необхідність у розширеному аналізі поведінки суперників: Швидкі зміни в ігровому процесі та поведінці гравців вимагають від суперників здатності адаптуватися та використовувати складні стратегії для забезпечення постійного виклику. Розробка платформ, що дозволяють моделювати таку поведінку, стає ключовою для підтримки цієї динаміки.

3. Потреба в сучасних аналітичних інструментах: Ринок відеоігор стає все більш складним і конкурентним. Тому розробка програмних платформ, що надають глибокий аналіз поведінки суперників та дозволяють прогнозувати реакції на дії гравців, стає життєво необхідною для створення інноваційних ігрових продуктів.

Основною перевагою розробки програмної платформи для моделювання поведінки суперників є зручність та доступність для розробників ігор. Така платформа дозволяє інтегрувати складні алгоритми поведінки без потреби в глибоких знаннях у галузі машинного навчання, оптимізуючи процес створення ігор. Вона забезпечує інструменти для ефективного налаштування поведінки суперників, які можуть динамічно реагувати на дії гравців, забезпечуючи більш захопливий ігровий процес. Платформа корисна як для досвідчених розробників, так і для новачків, які прагнуть створювати ігри зі складними та непередбачуваними противниками.

1.2 Аналіз аналогів розроблюваного програмного продукту

Ринок комп'ютерних ігор динамічно розвивався за останні кілька десятиліть. Згідно з дослідженнями 2017 року, світова вартість ринку відеоігор

досягла 108 мільярдів доларів США [14]. Очікується, що це значення буде зростати й надалі та перевищуватиме вартість кіноринку. Під

За величиною ігрового ринку Польща посідає 6 місце в Європі та 23 місце у світі.

світ.

Відеоігри мають широку аудиторію, що робить їх домінуючими форма розваги. Ігри можна покращити за допомогою базової форми штучного інтелекту мають значний вплив на ігровий процес і значно урізноманітнюють однокористувацькі ігри.

"Hell is Other Demons" — це платформна булет хелл екшн-гра, створена студією Cuddle Monster Games. Гра розроблена у стилі піксель-арту, а ігровий процес та візуальне оформлення нагадують класичні 8-бітні аркадні стрілялки. На малюнку 3.1 представлено кадр з гри, основною метою якої є перемога над надходящими хвилями суперників. Гра має два режими гри.

Рисунок 1.1 – Гра "Hell is Other Demons"

Основний режим кампанії полягає в тому, що гравець має перемогти суперників на даному рівні, в кінці якого знаходиться фінальний противник, так званий Бос. З прогресом гри зростає рівень складності, а гравець має можливість відкрити нові рівні, здобувати нові навички для своєї персонажі та купувати зброю. Смерть у цьому режимі означає початок рівня знову. Другий

режим — Аркада, характеризується більш вільним ігровим процесом, гравець може вибрати персонажа, кожен з яких має унікальну для себе зброю та навичку. Метою цього режиму є здобуття якомога більшої кількості очок, які, в свою чергу, дають доступ до нових героїв. Смерть у цьому режимі призводить до втрати очок і початку всього спочатку. Гра вирізняється візуальним оформленням, плавним і динамічним геймплеєм та важкою саундтреком у стилі синтвейв.

Однією з платформних ігор типу Metroidvania є "Dust: An Elysian Tail". Випущена вона була в 2012 році на Xbox 360, а також на інші платформи у пізніші терміни. Особливістю гри є те, що більшість її було створено однією особою — Діном Додріллом. У цій грі головний герой, Даст, здобуваючи нові вміння, відкриває наступні раніше недоступні етапи гри. На малюнку 1.2 представлений головний герой гри.

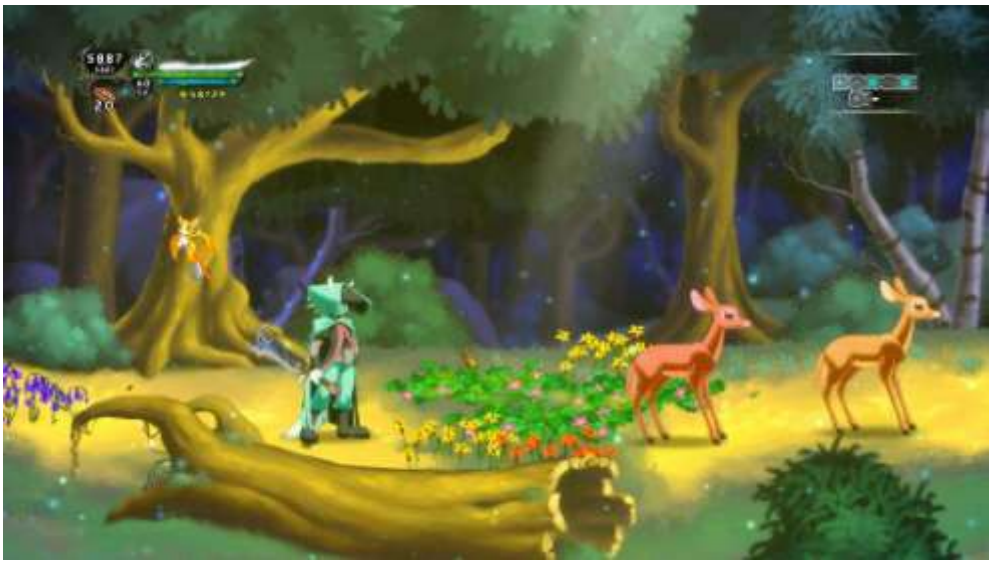


Рисунок 1.2 – Гра "Dust: An Elysian Tail"

Тип гри надихнутий класичними іграми для старих консолей, такими як Metroid і Castelvania, які були першими, що ввели модель прогресу в грі через отримання нових умінь. Ці уміння часто використовуються як атаки, але також можуть слугувати для подальшого відкриття раніше недоступних локацій. Такі ігри класифікуються як Metroidvania (поєднання ігор Metroid і Castelvania). Геймплей зосереджений на одноосібному режимі, де гравець за допомогою

різних умінь і атак перемагає монстрів, керованих штучним інтелектом, що забезпечує йому досвідові очки та золото, які можна витрачати у інших негравальних персонажів.

Ще одним прикладом є платформна аркадна гра Castlevania: Symphony of the Night з елементами RPG, яка є класикою жанру ігор типу Metroidvania. Гру створила Konami Computer Entertainment Tokyo і випустила на консолі PlayStation компанія Konami у 1997 році. Гравець втілюється в мисливця на вампірів, зображеного на малюнках 1.3 та 1.4, який вирушає до замку Дракули, сповненого монстрами та різноманітними пастками.

Рисунок 1.3 – Гра "Castlevania: Symphony of the Night"

Геймплей є нелінійним, оскільки гравець може вільно досліджувати замок, деякі кімнати залишаються заблокованими до моменту отримання необхідного вміння. Під час подорожі замком гравець зустрічає різноманітних с суперників та босів, яких можна перемогти за допомогою розблокованих магічних заклинань або доступної бічної зброї, розкиданої по всьому замку. Персонаж може підніматися на вищі рівні, що підвищує його статистичні показники, здобувати реліквії, які розблоковують нові вміння, та одягати

різноманітні обладунки, що збільшують базові статистичні показники, визначаючи бойові можливості героя. Гра виконана у стилі піксель-арту та утримується в атмосфері мрачного готику, вирізняється досить високим рівнем складності, вимогливими босами та різними завершеннями, доступними в залежності від прийнятих рішень під час гри.

Рисунок 1.4 – Гра "Castlevania: Symphony of the Night"

Малюнок 1.5 показує скріншот з гри Rayman 3 Hoodlum Havoc. Це третє видання гри з серії Rayman і водночас одна з класичних ігор свого жанру, створена студією Ubisoft. Гра є типовою платформною грою дії, в якій гравець втілюється у титульного Реймана, персонажа, створеного з окремих частин рук, ніг, голови, тулуба. Сюжет гри досить простий, завданням гравця є перемога армії так званих Худлумів та врятування світу. Під час гри персонаж проходить різноманітні рівні, на кінці яких чекають вимогливі противники, по дорозі ліквідуючи багатьох ворогів, які мають унікальні механіки. Гравець ні в чому не поступається зустрічним ворогам і також має до свого розпорядження різноманітну зброю у вигляді рукавичок, що мають унікальні здібності. Гра, як на свій час, мала різноманітні механіки та оригінальні рівні, що пропонували різноманітний стиль гри.

Рисунок 1.5 – Гра " Rayman 3: Hoodlum Havoc"

Критерії	Hell is Other Demons	Dust: An Elysian Tail	Castlevania: Symphony of the Night	Rayman 3: Hoodlum Havoc	Власна розробка
Казуальність	-	+	-	+	+
Перегравання	+	-	+	-	+
Варіятивність	+	-	+	+	+
Противники	+	-	-	+	+
Великий вибір спорядження	-	+	-	-	+

Отже, згідно проведеного порівняльного аналізу, було виявлено, що власна розробка є доцільною, оскільки містить всі позитивні якості конкурентів та немає жодних з їхніх мінусів, тому власна розробка має великий потенціал для покращення та доопрацювання всього того, що є в конкурентів.

1.3 Постановка задачі

На основі глибокого аналізу існуючих підходів і технологій у галузі штучного інтелекту, команда визначила ключові етапи для створення інноваційної платформи для моделювання поведінки суперників у платформерних відеоіграх. Основні напрямки розробки включають:

− Дослідження сучасних технологій та методів штучного інтелекту, які можна адаптувати для моделювання складних поведінкових паттернів суперників у грах.

− Розробка архітектури програмної платформи, здатної підтримувати високу працездатність та масштабування, забезпечуючи при цьому гнучкість для подальших модифікацій і оновлень.

− Розробка інтуїтивного і адаптивного графічного інтерфейсу, який буде зручним для користувачів різного віку і досвіду.

− Реалізація програмної платформи, забезпечена потужними модулями штучного інтелекту для симуляції різноманітних стратегій поведінки суперників.

− Проведення модульного тестування для гарантування надійності та ефективності всіх компонентів системи.

− Випробування фінального продукту з метою оптимізації продуктивності та стабільності під час гри, що включає стрес-тестування та аналіз відгуків користувачів.

− Розробка і публікація повного набору інструкцій для користувачів, що описують функціонал та можливості нової платформи.

Технічне завдання та детальний план розробки додаються у додатку Б.

1.4 Висновки

У першому розділі розглянуто існуючі програмні рішення, такі як Hell is Other Demons, Dust: An Elysian Tail, Castlevania: Symphony of the Night та Rayman 3: Hoodlum Havoc, з метою виявлення їхніх сильних і слабких сторін. Детальний аналіз показав, що ці системи не забезпечують всіх необхідних функцій для задоволення специфічних вимог проекту зі створення платформи для моделювання поведінки суперників у платформерних іграх .

2 РОЗРОБКА ІНТЕРФЕЙСІВ ТА МОДЕЛЕЙ

2.1 Аналіз вхідних даних системи

Для реалізації системи моделювання поведінки суперників у платформерній грі будуть використані мова програмування C# та фреймворк Unity. C# є потужною та інтуїтивно зрозумілою мовою програмування, розробленою Microsoft, яка часто використовується для створення додатків Windows, а також ігор на платформі Unity. Unity - це передовий ігровий двигун, який дозволяє розробникам створювати складні та візуально привабливі ігри для різних платформ, включаючи ПК, консолі та мобільні пристрої.

Розробка програмних модулів для платформи моделювання поведінки суперників у платформерній грі передбачає кілька ключових етапів. По-перше, необхідно створити модуль, який може аналізувати дії гравця та відповідати на них у режимі реального часу. Цей модуль буде відповідати за ідентифікацію стратегій гравця та адаптацію поведінки суперників до них, забезпечуючи виклик та залученість у грі.

Далі буде розроблено модуль для аналізу ігрового середовища, який використовуватиме методи штучного інтелекту для оцінки поточного стану гри та оптимізації стратегій суперників. Цей модуль дозволить системі відстежувати розташування об'єктів, перешкод, а також інші динамічні елементи гри, що впливають на поведінку суперників.

Після аналізу середовища і дій гравця буде розроблений модуль для рендерингу поведінки. Цей модуль буде використовувати дані, отримані від попередніх модулів, для генерації реалістичних реакцій суперників. Модуль дозволить противникам не тільки реагувати на дії гравця, але й адаптуватися до змін у стратегії гравця, створюючи більш динамічні та непередбачувані ігрові взаємодії.

Розробка цих програмних модулів вимагає глибокого розуміння штучного інтелекту, графічного програмування та мови програмування C#. Для реалізації

цих модулів будуть використані різноманітні інструменти та бібліотеки програмування, зокрема Unity, Microsoft Visual Studio та інші бібліотеки та фреймворки, які підтримують розробку ігрових додатків. Програмні модулі будуть протестовані та доопрацьовані, щоб забезпечити ефективне моделювання поведінки суперників у динамічному ігровому середовищі[11].
Отже, розробка програмних модулів для платформи моделювання поведінки суперників у платформерній грі є комплексним завданням, яке включає кілька етапів. Модулі повинні бути ретельно спроектовані, реалізовані та протестовані, щоб забезпечити реалістичну та викликаючу поведінку суперників. Використання передових технологій та інструментів, таких як Unity і C#, дозволяє створити потужну систему, яка може значно покращити геймплей і зробити ігровий процес більш захоплюючим та взаємодійним.

2.2 Розробка інтерфейсу програмного додатку

Розробка інтерфейсу користувача (UI) є ключовим аспектом у створенні програмного забезпечення, включаючи платформери з моделюванням поведінки суперників. Інтерфейс користувача є тим мостом, через який гравець взаємодіє з грою. Він охоплює всі засоби взаємодії, такі як інтерактивні кнопки, текстові поля, іконки та меню, що забезпечують простоту і інтуїтивну зрозумілість використання.

Ретельне проектування UI спрямоване на забезпечення ефективності та комфорту користувача під час гри. Це дозволяє легко навігувати функціоналом і досягати цілей у грі без зайвих перешкод. Коли інтерфейс оптимально спроектований, гравець може без проблем взаємодіяти з ігровим світом, виконувати задачі та отримувати задоволення від гри, мінімізуючи зусилля і час, необхідні для вивчення управління.

У процесі створення платформерів засоби взаємодії користувача з програмою відіграють ключову роль. Сучасні ігри можуть використовувати різноманітні типи інтерфейсів, які підсилюють загальний ігровий досвід. Нижче

наведено декілька основних типів користувацьких інтерфейсів, що можуть бути інтегровані у розробку гри:

1. Графічний інтерфейс користувача (GUI): Найчастіше використовується у більшості відеоігор, GUI включає візуальні елементи, такі як кнопки, іконки, вікна та меню, що дозволяють інтуїтивно легко взаємодіяти з грою через графічні об'єкти. Це дозволяє гравцю легко зрозуміти, що потрібно зробити для виконання певної дії або досягнення мети у грі.

2. Текстовий інтерфейс користувача (TUI): Хоча цей тип інтерфейсу менш поширений у відеоіграх, він може використовуватися для ретро-стилізованих або експериментальних ігрових проектів, де гравці взаємодіють з програмою за допомогою текстових команд. Це може створити унікальний ігровий досвід, додаючи елемент ностальгії або виклику для гравців.

3. Сенсорний інтерфейс: Дуже поширений у мобільних іграх, цей інтерфейс дозволяє взаємодіяти з ігровим світом за допомогою дотиків, що забезпечує простоту та доступність управління. Гравці можуть торкатися екрану, змахувати, затискати та розтягувати об'єкти, що створює більш інтерактивний і залучаючий досвід.

4. Жестовий інтерфейс: Використання жестів для керування діями у грі, такі як махання рукою або певні рухи, може додати новий рівень інтерактивності та забави. Цей тип інтерфейсу може бути реалізований за допомогою спеціальних сенсорів або камер, які розпізнають рухи гравця.

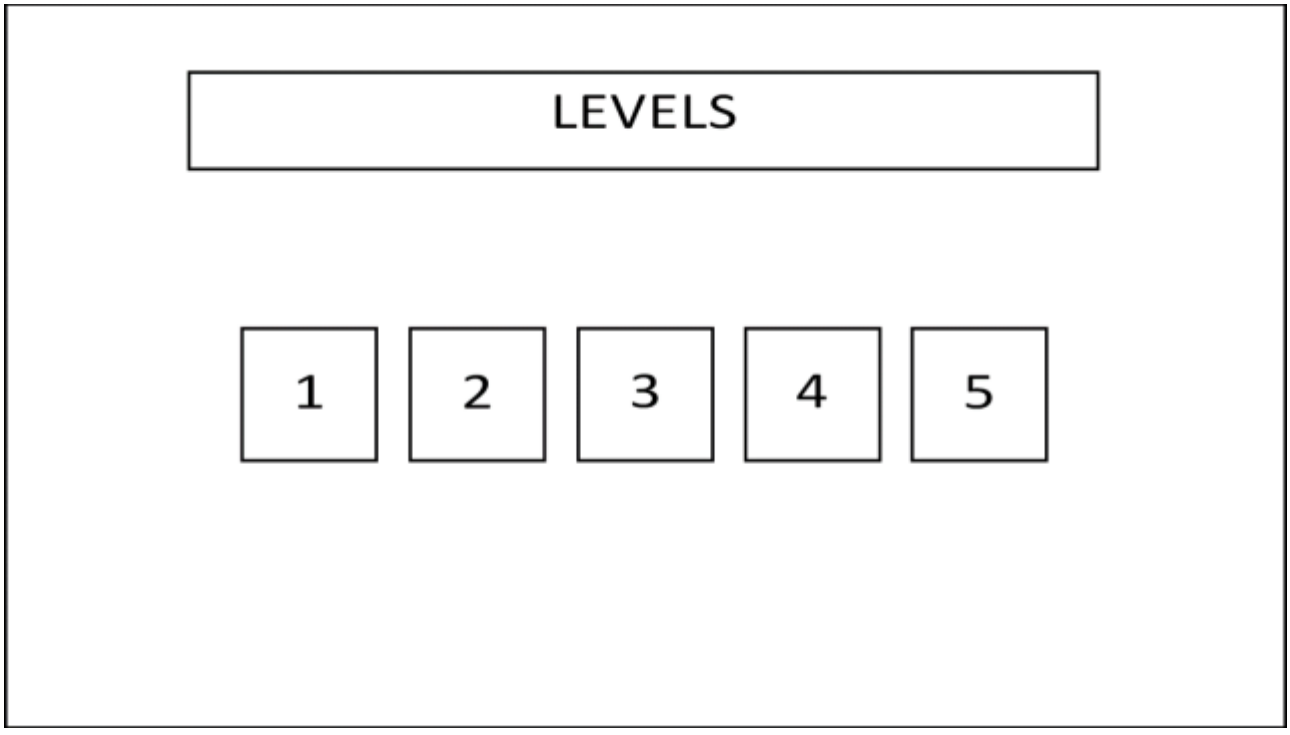
Кожен тип інтерфейсу може використовуватися в залежності від контексту та стилю самої гри, а також від орієнтованої аудиторії. Важливо враховувати, який тип інтерфейсу буде найзручнішим та найефективнішим для конкретної гри та її гравців.

В моїй грі буде використано такі засоби взаємодії:

- Головне меню: Це меню з’являється при запуску гри і дозволяє гравцю вибрати початок нової гри, продовжити збережену гру, змінити налаштування, або вийти з гри. Головне меню має бути інтуїтивно зрозумілим та легко доступним, забезпечуючи гравцям швидкий доступ до основних функцій гри.

- Меню паузи: Під час гри гравець може натиснути кнопку паузи, щоб зупинити гру і відкрити меню паузи. Це меню дозволяє гравцю зберегти прогрес, змінити налаштування, або вийти з гри без втрати поточного стану. Меню паузи повинно бути простим у використанні і швидко відкриватися, щоб не відволікати гравця від ігрового процесу.

- Сенсорний інтерфейс гравця: Для мобільної версії гри, сенсорний інтерфейс дозволяє гравцю керувати персонажем та взаємодіяти з ігровим середовищем за допомогою дотиків. Це може включати кнопки для стрибків, атаки, руху та інших дій, а також жести для виконання спеціальних дій. Сенсорний інтерфейс має бути розроблений так, щоб забезпечити максимальну точність і комфорт управління.



2.1 – Головний екран

На головному екрані нашої гри знаходяться п'ять кнопок, пронумерованих від 1 до 5, що представляють різні рівні гри. Кожна з цих кнопок оформлена у простому, чіткому стилі з чіткими межами та великими цифрами, що забезпечує легке розуміння та взаємодію для гравця. Дизайн

кнопок спеціально розроблений так, щоб гравець міг без зайвих зусиль вибрати потрібний рівень.

Таке оформлення дозволяє гравцю швидко вибрати бажаний рівень, що є особливо корисним для ігор, де гравці часто повертаються до попередніх рівнів для покращення своїх результатів або для проходження різних місій. Кожна кнопка також має динамічні елементи: вона може підсвічуватися або змінювати колір при наведенні курсору або торканні на сенсорному екрані. Ці інтерактивні елементи надають гравцю зворотний зв'язок, підтверджуючи, що обраний рівень готовий до запуску.

Інтерфейс ефективно використовує мінімалістичний підхід, зосереджуючи увагу користувача виключно на виборі рівня, не відволікаючи його додатковими елементами. Такий підхід є ідеальним для швидких і прямих дій, типових для платформерів. Він дозволяє гравцям зосередитися на грі, не витрачаючи час на зайві деталі, що значно покращує користувацький досвід.

Крім того, розташування кнопок може бути таким, що гравець інтуїтивно розуміє, який рівень вже пройдений, а який ще не відкритий. Це може бути реалізовано за допомогою різних кольорів або значків, що позначають статус рівня. Такий дизайн не лише робить інтерфейс більш зрозумілим, але й додає мотивації для проходження нових рівнів та повторного проходження вже завершених для покращення результатів.

2.3 Розробка 2D моделей

Розробка графіки в іграх вимагає від художників та дизайнерів використання комбінованих підходів, включаючи створення концепт-арту, який задає візуальний стиль гри, та роботу над комп'ютерною графікою.

Процес створення графіки починається з концептуалізації, де концепт-художники розробляють ескізи персонажів, ворогів, а також середовища гри. Ці ескізи слугують фундаментом для візуального наративу гри і допомагають уточнити ключові візуальні елементи. Співпраця між концепт-художниками і

геймдизайнерами в цьому процесі є важливою для втілення арт-дирекції гри в життя.

Після стадії концепт-арту, комп'ютерні художники переходять до створення остаточних графічних асетів. Це включає розробку двовимірних спрайтів для персонажів і ворогів, а також деталізованих фонів та текстур. Особливу увагу приділяють анімації, яка має велике значення для передачі руху та станів персонажів.

Для інтеграції створеної графіки в ігровий двигун, спеціалісти з розробки ігор використовують спеціалізовані інструменти та техніки. Це дозволяє забезпечити оптимальну продуктивність ігрового двигуна та залученість гравців, створюючи багатий візуальний досвід.

Крім того, розробка графіки, хоча і не завжди застосовна до ігор, може бути використана для створення елементів з глибиною та об'ємом, що надають грі додаткової реалістичності та привабливості. Такі елементи можуть включати спецефекти, які використовують 2D моделювання для створення більш насиченого візуального контексту.

Спершу необхідно створити візуальні елементи, що забезпечать глибину та деталізацію гри. Це включає процес створення двовимірних об'єктів та їх текстурування, щоб надати їм привабливий вигляд у віртуальному світі.

Одним із ключових етапів є створення персонажів, об'єктів та оточення у грі. Це передбачає розробку геометричних форм та надання їм текстур, що робить об'єкти більш реалістичними. Після створення візуальних елементів слід приділити увагу анімації, яка надає рух і життя персонажам та об'єктам. Моделі були створені за допомогою вбудованого 2D-редактора Unity. Візуалізацію даного процесу можна побачити на ілюстраціях 2.1 – 2.5.

Рисунок 2.2 – Модель гравця

Гравець — лучник, одягнений у традиційне зелене вбрання, що нагадує класичне фентезі або середньовічну тематику. Він стоїть в динамічній позі, тримаючи лук зі стрілою, готовою до пострілу. У дизайні використовується обмежена палітра, зосереджена на відтінках зеленого, коричневого та сірого, щоб зберегти візуальну чіткість і ретро-естетику.

Рисунок 2.3 – Модель містичного лучника

Містичного лучника — цей персонаж є суперником на дальній відстані у грі. Він одягнений у зелений плащ, що ідеально камуфлює його в лісових локаціях, додаючи елемент скритності його зовнішньому вигляду. Він оснащений довгим луком з яскраво-жовтими акцентами, які вказують на магічну природу його зброї. Його блакитне око виділяється на тлі тьмяних кольорів його одягу, створюючи враження надприродного спостереження. Лісовий Стрілець використовує свої навички лучника для проведення далеких атак, що вимагає від гравця швидкості та точності, щоб уникнути його стріл та зменшити дистанцію для ближнього бою.

Рисунок 2.4 – Модель тіньового рейнджера

Тіньовий рейнджер — цей тип противників зустрічається на початкових рівнях гри. Він одягнений у чорні обладунки з яскравими фіолетовими акцентами, що символізують його зв'язок з магією та таємницею. Це відображення його способу боротьби — скритності та несподіванки. Персонаж володіє довгим фіолетовим мечем, що іскриться енергією, надаючи йому здатність здійснювати потужні та руйнівні удари. Цей меч не тільки зброя, але й ключ до його чарівних здібностей, роблячи його важким суперником навіть для досвідчених гравців.

Рисунок 2.5 – Модель демона

Демон — цей тип ворогів зустрічається на ключових етапах гри. Він одягнений в грубі червоно-чорні обладунки, що може символізувати його зв'язок з пекельними силами або кровожерливий характер. Його роги та мускулиста статура підкреслюють його бойову міць і здатність домінувати в бою. В руках він тримає величезний білий меч, який не тільки виглядає лякаюче, але й підкреслює його статус особливо небезпечного противника з елементами містичної сили.

Рисунок 2.6 – Модель мінотавра

Мінотавр — цей тип ворога ідеально підходить для ролі міжнівневого міні боса або потужного суперника. Він має міцне м'язисте тіло з більш світлою шкірою та рогами, що додають йому дикості та варварських рис. Він одягнений у прості, але функціональні шкіряні штани з червоними акцентами на шнурівці та взутті, підкреслюючи його первісний вигляд. В руках він тримає масивний дворучний меч, який свідчить про його силу та здатність завдавати значної шкоди. Мінотавр чудово підходить для битв, які вимагають від гравця швидкість реакції та стратегічного підходу, оскільки його атаки швидкі та смертоносні.

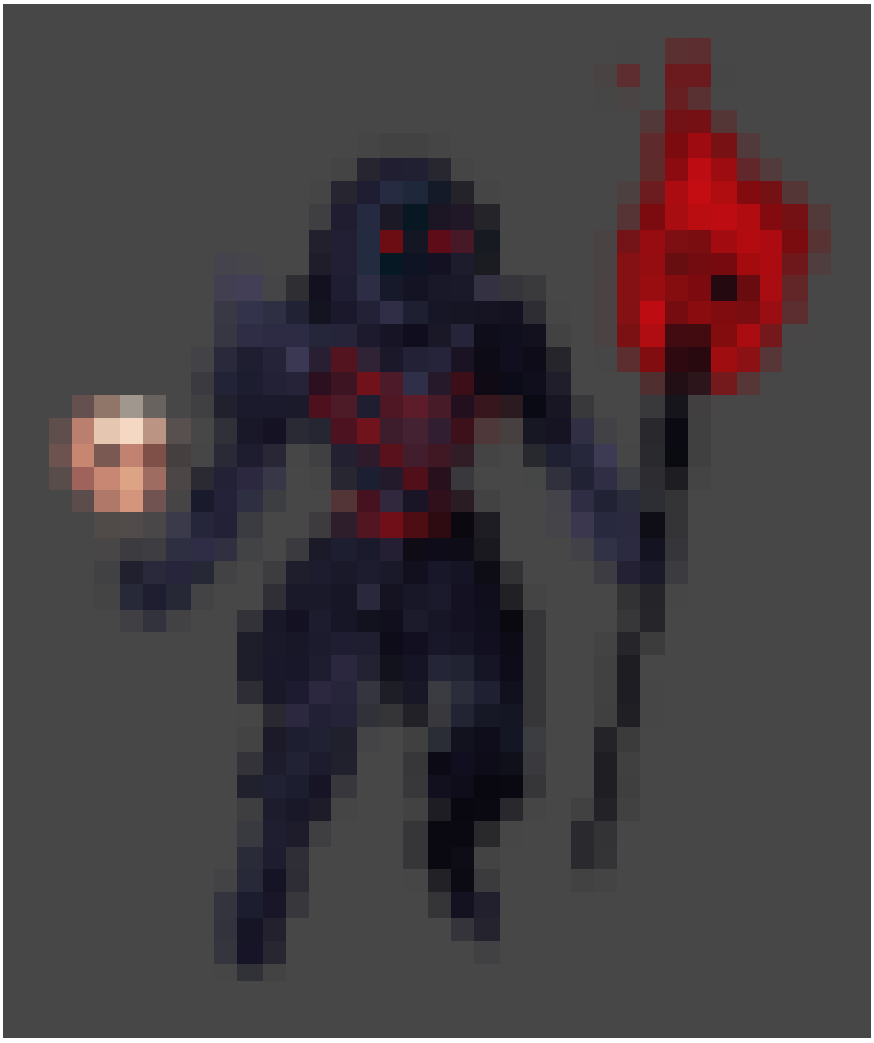


Рисунок 2.7 – Модель тіньового вартового

Тіньовий вартовий — цей персонаж є високорівневим суперником у грі. Він одягнений у темний броньований костюм з червоними акцентами, які підсилюють його загрозливий і містичний вигляд. Червоний прапор, який він тримає, може символізувати його владу або магічний джерело сили. Його обладунки дають йому перевагу в маневреності та захисті від атак. В руках він тримає велику сокиру, що підкреслює його силу та здатність вести бій на близькій відстані. Завдання перемогти Тіньового Вартового вимагає від гравця використання тактичних навичок, оскільки він може швидко атакувати та маневрувати, роблячи його непростим супротивником.

Рисунок 2.8 – Модель боса першого рівня

Скелет — цей тип ворога виступає як бос першого рівня у грі. Він весь складається з кісток, що надає йому особливо моторошний вигляд. Його яскраво-зелені очі випромінюють зловісне світло, підсилюючи його містичний та загрозливий образ. Хоча цей персонаж і не має зброї, його велика стійкість та здатність до маневрування в бою роблять його небезпечним супротивником. Він здатен несподівано нападати, використовуючи свої швидкі та різкі рухи, що вимагає від гравця особливої стратегії та реакції для його перемоги.

Рисунок 2.9 – Модель боса другого рівня

Вікінг скелет — цей тип ворога є босом другого рівня. Він оснащений темними металевими обладунками з рогами, які надають йому воїнський і залякувальний вигляд. Його зелені очі світяться з-під шолома, створюючи містичне враження. Він озброєний важким мечем та великим щитом, які відмінно підходять для оборони та атаки на короткі дистанції. Цей персонаж вимагає від гравця гарного вміння маневрувати та використовувати стратегічне мислення, щоб знайти правильний момент для атаки, уникаючи його потужних ударів мечем та щитом.

Рисунок 2.10 – Модель призивного скелета

Цей призивний скелет — один з підручних, якого викликає бос другого рівня у грі. Він одягнений у примітивні шкіряні шорти з візерунками та має на спині колчан зі стрілами, що підкреслює його роль як дальнобійного воїна. Його голова прикрашена архаїчними стрілами, які слугують як декорації та символізують його бойову суть. Зелене світіння очей надає йому містичного та загрозливого вигляду.

Скелет тримає в руках кістяне коп'є, що є його основною зброєю в бою. Він здатний проводити швидкі атаки на середній дистанції, а також підтримувати боса, відволікаючи супротивників або наносячи додаткові удари. Його магічна аура, що виходить від серцевини на грудях, може мати зцілювальні або підсилювальні властивості, допомагаючи іншим призваним скелетам у битві. Ця сутність ідеально підходить для тактики "розділяй і володарюй", дозволяючи босу другого рівня утримувати контроль на полі бою.

Рисунок 2.11 – Модель боса третього рівня

Мумія фараон — цей тип ворога є босом третього рівня. Він одягнений у яскраві, кольорові єгипетські обладунки з характерним головним убором, який підкреслює його королівський статус. Він озброєний золотим мечем, який світиться таємничим сяйвом, що додає містичності його особистості. Його бойові навички включають використання заклинань стародавнього Єгипту, які можуть викликати духів або прокляття, тим самим ускладнюючи битву для гравця. Здолати цього боса вимагає не тільки фізичної сили, але й здатності адаптуватися до магічних атак та розуміння артефактів, які можуть нейтралізувати його чари.

Рисунок 2.12 – Модель боса четвертого рівня

Володар темряви — цей персонаж є фінальним босом гри. Він одягнений у масивні, темні обладунки з золотими шипами, що роблять його силует ще більш залякувальним. Його головний убір у вигляді шолому з шипами та рогами підкреслює його владні та зловісні якості. В руках він тримає могутню битву з двома клинками, що свідчить про його військові навички та готовність до смертельного бою. Володар темряви використовує різноманітні магічні атаки та може викликати слуг, щоб допомогли йому в бою. Здолання цього боса вимагає не тільки високого рівня майстерності від гравця, але й стратегічного планування бою.

Рисунок 2.13 – Локація з другого рівня

Цей набір структур та елементів із зображення представляє собою типовий ландшафт другого рівня, включає наступні компоненти:

Дерева – різні типи дерев, що варіюються за кольором та формою, можуть використовуватися для створення лісових або паркових зон на карті;

Кущі – менші зелені насадження, що додають різноманітності та деталізації ландшафту;

Скелі та каміння – розкидані по землі, вони можуть служити як природні перешкоди або елементи для скриття від ворогів;

Споруди – невеликі будівлі, що можуть використовуватися для зберігання ресурсів або як укриття;

Артефакти – такі як статуї або обеліски, можуть бути ключовими точками для завдань чи місцями сили;

Фонтан – центральна структура з водою, може бути використана для відновлення здоров'я або як ключовий елемент для розгадування загадок;

Шлюзи та двері – можуть контролювати доступ до наступних рівнів або закритих зон.

Рисунок 2.14 – Моделі оточення

Моделі оточення які включає природні елементи, такі як дерева та рослини, структурні компоненти, такі як кам’яна вежа та дерев’яні паркани, а також дрібні деталі, такі як ящики, камені з мохом і черепи, які додають глибини оточенню. Кожна деталь ретельно заштрихована та виділена, щоб підкреслити її тривимірний вигляд в ізометричному вигляді.

Рисунок 2.15 – Локація з третього рівня

На зображенні показано локацію з третього рівня .Ось що можна виділити в цьому середовищі:

1. Дерева і кущі — по всій карті розташовані різні види дерев і кущів, що створюють відчуття густого, природного лісу;

2. Каміння і пеньки — серед дерев розкидані різні камені та пеньки, що додають текстуру та реалістичність ландшафту;

3. Дорога — серпантином проходить шлях, який служить напрямком для руху персонажів гри;

5. Маленькі об'єкти на землі — різні маленькі об'єкти, такі як камені і квіти, що роблять карту більш деталізованою і візуально привабливою;

6. Галявини і відкриті простори — місця без дерев, що можуть бути використані для зустрічей з персонажами або битв.

2.4 Розробка моделей та алгоритмів роботи системи

Розробка алгоритмів для системи моделювання поведінки суперників передбачає наступні кроки:

1. Підготовка даних: Процес розпочинається зі збору даних про ігрові сесії, де фіксуються різні дії гравців та реакції суперників. Ці дані можуть бути зібрані з ігрових логів або через спеціально розроблені інструменти збору даних. Далі дані обробляються для вилучення ключових характеристик, таких як швидкість руху, зміна позицій, використання предметів тощо. Очищення даних відбувається шляхом видалення шуму та корекції помилок, що забезпечує точність та чистоту даних для наступних етапів.

2. Розділення даних на тренувальний та тестовий набори: Отримані дані розділяються на тренувальний та тестовий набори для оцінки ефективності моделі. Використовується метод хресної перевірки для забезпечення об'єктивності результатів. Цей процес допомагає визначити найкращу архітектуру моделі та параметри, що підходять для конкретних задач.

3. Вибір моделі та параметрів: Вибирається архітектура моделі машинного навчання, така як нейронні мережі або ансамблеві моделі, які показали ефективність у задачах моделювання поведінки. Застосування алгоритмів глибинного навчання дозволяє моделювати складні залежності та виявляти нелінійні патерни поведінки.

4. Навчання моделі: Тренування моделі проводиться на тренувальному наборі даних, де модель навчається розпізнавати патерни поведінки та залежності між діями гравця та реакціями суперників. Процес навчання включає налаштування гіперпараметрів та оптимізацію моделі для досягнення найкращих результатів.

5. Оцінка моделі та налаштування параметрів: Після завершення тренування, модель оцінюється за допомогою тестового набору даних для визначення її точності та ефективності. За потреби, параметри моделі налаштовуються для поліпшення її результатів. Оцінка включає в себе аналіз різних метрик, таких як точність, повнота, F1-середнє тощо.

6. Застосування моделі для моделювання поведінки: Отримавши навчену модель, вона використовується для динамічного моделювання поведінки суперників у реальному часі в іграх, здатна адаптуватися до дій та стратегій гравців. Це дозволяє створювати більш реалістичні та складні ігрові сценарії, які покращують загальний ігровий досвід.[14]

2.5 Розробка UML діаграми

UML є мовою моделювання, що застосовується для створення та візуалізації архітектури програмних систем. Ця мова має в розпорядженні уніфіковані інструменти та нотації для конструювання діаграм, які слугують інженерам програмного забезпечення для проєктування, аналізу та документування систем. UML полегшує комунікацію між програмістами, архітекторами та іншими зацікавленими особами проєкту.

UML охоплює різні види діаграм, включаючи діаграми класів, діаграми послідовностей, діаграми активності, діаграми випадків використання та багато інших. Кожна діаграма призначена для відображення окремих аспектів програмної системи. Наприклад, діаграми класів займаються структуруванням системи, тоді як діаграми послідовностей демонструють порядок операцій всередині системи.

У рамках UML, діаграми класів застосовуються для детального моделювання структури системи, визначення класів, їхніх атрибутів, методів, та взаємин між класами. При створенні програмних платформ для моделювання поведінки суперників у платформних іграх, такі діаграми можуть бути використані для опису структури суперників, їхніх поведінкових патернів, а також їх взаємодій між собою.

Діаграми послідовності ілюструють взаємодію між об'єктами протягом часу, що сприяє зрозумінню логіки поведінки суперників та їхніх взаємодій з іншими елементами гри. Такі діаграми можуть бути застосовані для моделювання атак суперників і їх реакцій на дії гравців.

Діаграма прецедентів у UML служить для зображення взаємодій між системою і зовнішніми агентами з погляду функціональності системи. В контексті платформних ігор, такі діаграми можуть використовуватися для ідентифікації основних функцій суперників та їхніх інтеракцій з гравцем, включаючи атаки, ухилення та інші стратегічні дії.

Діаграма діяльності ілюструє послідовність подій, що відбуваються під час взаємодії гравця з ігровими противниками. Ця діаграма допомагає візуалізувати та аналізувати логіку поведінки суперників, забезпечуючи глибше розуміння механізмів їх взаємодій. На діаграмі можуть бути представлені різні стани, які противник може приймати в різних сценаріях: від початкового стану спокою, реакції на гравця, до вибору атаки та завершення атаки. Також можливі переходи до нових станів, наприклад, перехід у стан оборони або втечі, залежно від ситуації в грі.

Кожен крок на діаграмі вказує на конкретну дію або перевірку, яка відбувається у противника: наприклад, визначення чи гравець знаходиться у полі зору, рішення про напад чи ухилення, активація атаки, що може включати визначені анімації атаки, та врешті-решт – завершення атаки.

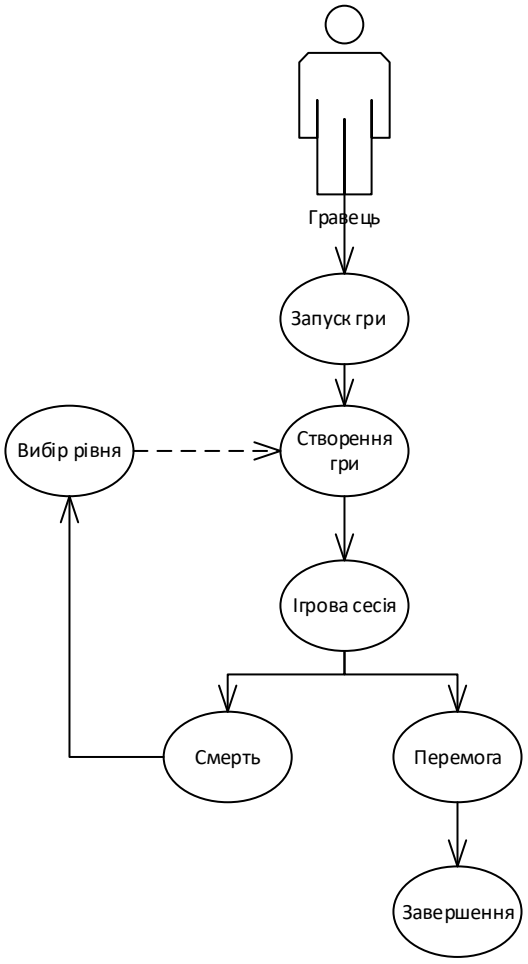


Рисунок 2.6 – Модель поведінки суперників

2.6 Розробка алгоритму поведінки суперників

Алгоритм представляє собою детально розроблений план дій або серію кроків, які необхідно виконати для досягнення визначеної цілі. Як важливий інструмент для програмістів, алгоритм дозволяє систематизувати та наглядно представляти процес розв'язання задач крок за кроком. Він формулюється через

програмне забезпечення, яке налаштовує комп'ютер на виконання заданих інструкцій у відповідній послідовності, аби досягти потрібного результату. Робота ігрової системи включає у себе ряд послідовних етапів:

1. Підготовка - етап, де визначаються первинні кроки або умови перед початком основної діяльності;

2. Атака - дія, яка передбачає активні дії визначеного характеру;

3. Знайти ціль руку - визначення цілі або об'єкта для подальших дій;

4. Вистріл стріл - здійснення запланованого дії, яке могло включати запуск або активізацію певних механізмів або програм;

5. Перебірка атаки - реалізація атаки, включно з оцінкою ефективності та внесенням коригувань;

6. Заборона атаки, акт відсіч атаки - зупинка або оборона від атак, захист;

7. Кінець - завершення дій або процесу.

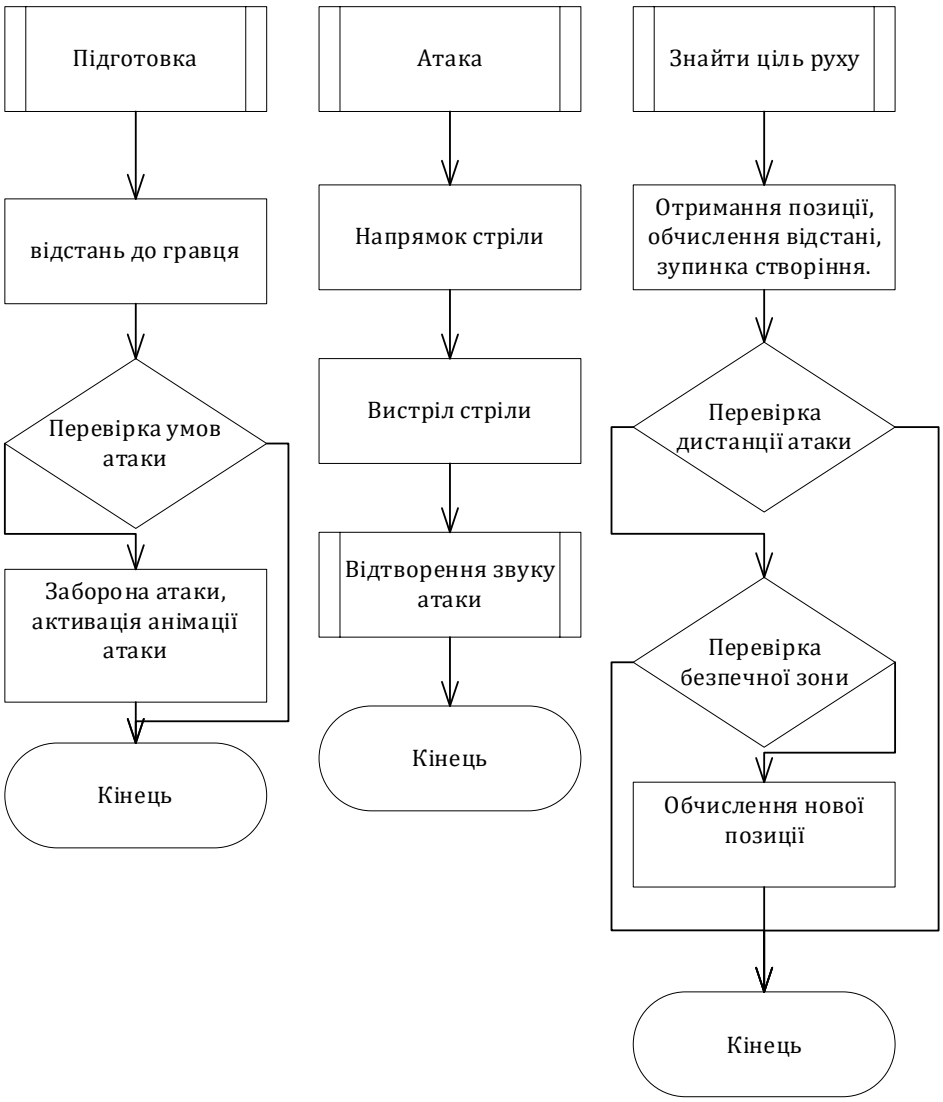


Рисунок 2.6 – Модель поведінки суперників

Також було розроблено алгоритм спавну мін для більної варятивності та ускладнення ігрового процесу. Нижче можна побачити роботу цього алгоритму:

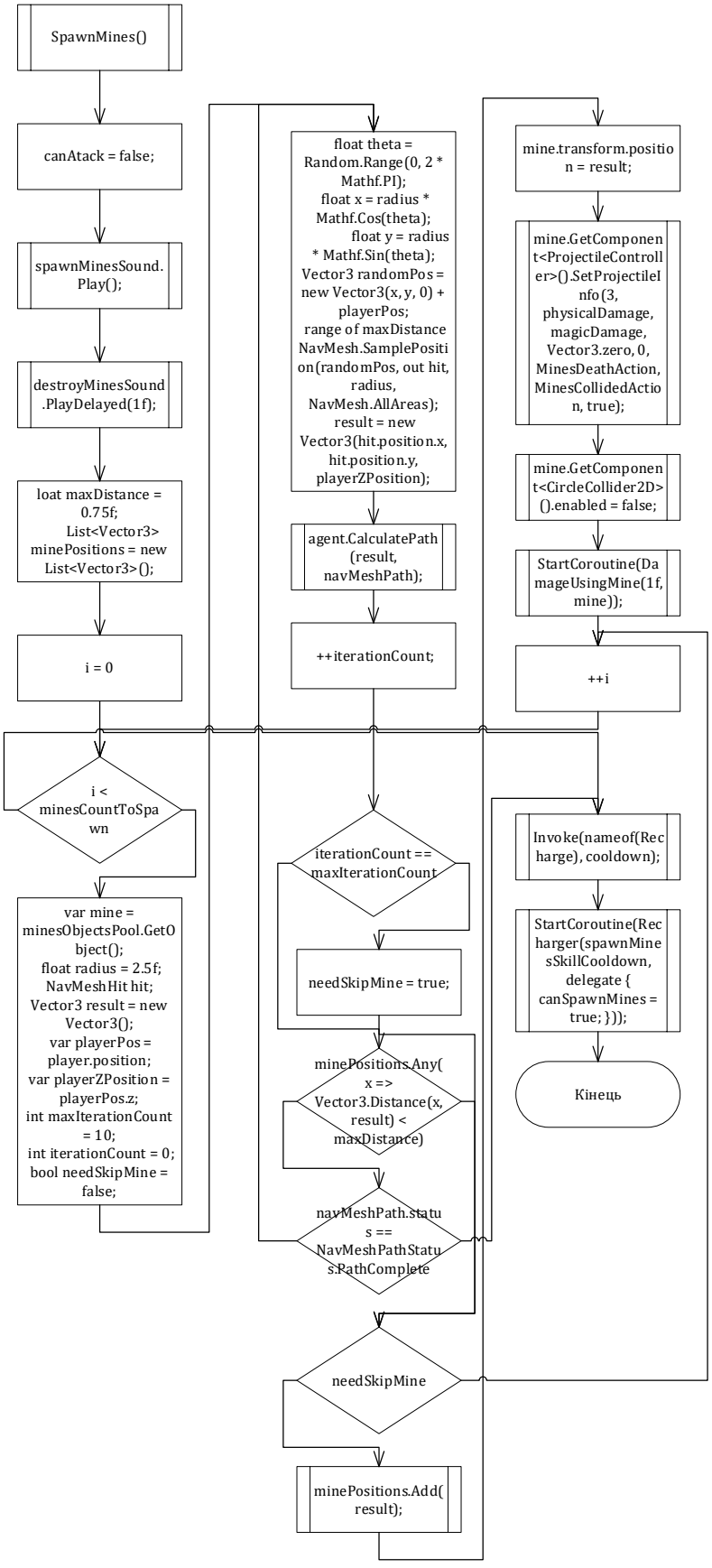


Рисунок 2.7 – Алгоритм появи міни

Цей алгоритм починає свою роботу з перевірки, чи слід активувати процес створення мін. Якщо параметр `spawnMines` істинний, це означає, що міни можуть бути створені. В такому разі алгоритм встановлює параметр `canAttack` в стан "false", що може вказувати на тимчасову заборону атаки під час активації мін. Також відтворюється звук, асоційований з процесом спавнення мін, що може слугувати аудіальним підтвердженням початку дії.

Далі алгоритм переходить до циклічного процесу створення мін. У цьому циклі відбувається генерація випадкових позицій для розміщення мін у віртуальному просторі. Для кожної потенційної позиції використовується метод, який аналізує відстань до перешкод за допомогою променів, що випромінюються з центру до різних точок у просторі. Це дозволяє визначити, чи є місце вільним для безпечного розміщення міни.

Якщо підходяща позиція знайдена, створюється міна, і її позиція фіксується. Все це повторюється до тих пір, поки не буде досягнуто заданої кількості мін або поки алгоритм не розгляне достатню кількість потенційних позицій. Якщо визначається, що додаткові міни не потрібні або ітераційний ліміт вичерпано, процес припиняється.

Окремо в алгоритмі передбачена можливість призупинення спавнення мін, якщо це необхідно, через параметр `needSkipMine`. Цей параметр може активуватися, якщо з'являються певні умови, такі як активація спеціальних здібностей або зміни у грі, які вимагають тимчасового припинення дій з мінами.

Також в алгоритмі є процес перезавантаження або "cooldown" спеціальних здібностей, пов'язаний з мінами. Це вказує на можливість того, що після активації мін відбуваєтьсячасова затримка, протягом якої не можуть бути створені нові міни, доки не завершиться період перезавантаження. Це дозволяє контролювати частоту і тактику використання мін в ігровому процесі, забезпечуючи баланс і стратегічне планування використання ресурсів у грі.

2.7 Висновки

У другому розділі роботи викладено задачі розробки та визначено етапи створення якісної 2D платформної гри. Проведено аналіз інформаційного забезпечення необхідного для створення ігрового продукту. Розроблено структуру інтерфейсу, що відповідає потребам широкої аудиторії користувачів та є інтуїтивно зрозумілою. Створено 2D макети ігрових локацій, а також дизайн персонажів. Розроблено основні методи та алгоритми для системи моделюванняповедінки суперників в грі, що дозволяє створити більш реалістичне та викликаюче захоплення ігрове середовище.

3. РОЗРОБКА ПРОГРАМНИХ ЗАСОБІВ ПЛАТФОРМИ МОДЕЛЮВАННЯ ПОВЕДІНКИ СУПЕРНИКІВ У ПЛАТФОРМЕРНИХ ІГРАХ

3.1 Варіантний аналіз і обґрунтування вибору засобів для реалізації програмного забезпечення

При створенні програмних засобів для платформи моделювання поведінки суперників у платформерній грі критично важливо вибрати правильні технології. C# у поєднанні з ігровим двигуном Unity є відмінним вибором для цієї мети. C# - це об'єктно-орієнтована мова програмування від Microsoft, яка поєднує в собі ефективність, гнучкість та доступність, роблячи її популярною серед розробників ігор.

Unity - це потужний кросплатформенний ігровий двигун, який підтримує розробку ігор для багатьох платформ, включаючи мобільні пристрої, ПК та консолі. Однією з ключових особливостей Unity є його інтеграція з C#, що дозволяє розробникам швидко імплементувати складні ігрові механіки та алгоритми поведінки[15].

Використання Unity забезпечує доступ до обширного набору інструментів для анімації, штучного інтелекту, фізики, рендерингу та багатьох інших аспектів, які є незамінними для розробки сучасних відеоігор. Це дозволяє розробникам не тільки ефективно моделювати реалістичну поведінку суперників, але й легко інтегрувати ці моделі в багатошарові ігрові світи[16].

Таким чином, вибір C# та Unity як основних інструментів для розробки платформи моделювання поведінки суперників є обґрунтованим рішенням, що дозволяє забезпечити високу продуктивність, гнучкість у розробці і широкі можливості для творчості та інновацій.

3.2 Розробка програмного засобу для моделювання поведінки суперників

При створенні програмних засобів для платформи моделювання поведінки суперників у платформерній грі критично важливо вибрати правильні технології. C# у поєднанні з ігровим двигуном Unity є відмінним вибором для цієї мети. C# - це об'єктно-орієнтована мова програмування від Microsoft, яка поєднує в собі ефективність, гнучкість та доступність, роблячи її популярною серед розробників ігор.

Unity - це потужний кросплатформенний ігровий двигун, який підтримує розробку ігор для багатьох платформ, включаючи мобільні пристрої, ПК та консолі. Однією з ключових особливостей Unity є його інтеграція з C#, що дозволяє розробникам швидко імплементувати складні ігрові механіки та алгоритми поведінки. Використання Unity забезпечує доступ до обширного набору інструментів для анімації, штучного інтелекту, фізики, рендерингу та багатьох інших аспектів, які є незамінними для розробки сучасних відеоігор. Це дозволяє розробникам не тільки ефективно моделювати реалістичну поведінку суперників, але й легко інтегрувати ці моделі в багатошарові ігрові світи.

Одним із важливих аспектів вибору технологій для розробки програмного забезпечення є врахування їхньої продуктивності та масштабованості. Unity забезпечує високу продуктивність завдяки оптимізованим алгоритмам рендерингу та ефективному використанню ресурсів системи. Це особливо важливо для створення платформерних ігор, де необхідно обробляти велику кількість об'єктів та їх взаємодій в реальному часі. Крім того, Unity підтримує багатопоточність, що дозволяє ефективно використовувати можливості сучасних багатоядерних процесорів.

Ще одним важливим фактором є доступність великої кількості бібліотек та плагінів, які розширюють можливості Unity. Це дозволяє розробникам легко інтегрувати додаткові функції та розширення, такі як підтримка мережевих з'єднань, системи досягнень, аналітика ігрового процесу тощо. Використання

готових рішень значно скорочує час розробки та дозволяє зосередитися на унікальних аспектах гри.

Важливою перевагою Unity є його активна спільнота розробників та наявність великої кількості навчальних ресурсів. Це сприяє швидкому освоєнню платформи та вирішенню проблем, що виникають під час розробки. Спільнота також активно ділиться готовими рішеннями та прикладами, що може бути корисним для швидкого впровадження нових ідей та інновацій.

Крім Unity, варто розглянути альтернативні платформи, такі як Unreal Engine та Godot. Unreal Engine, наприклад, пропонує високий рівень графіки та потужні інструменти для роботи з тривимірними об'єктами. Проте його використання може бути більш складним і вимагати більше ресурсів. Godot, з іншого боку, є відкритим джерелом ігрового двигуна з простою у використанні системою сценаріїв, але його функціональні можливості можуть бути обмеженими порівняно з Unity.

3.3 Розробка функцій

Розробка функції "Skill" для моделювання спеціалізованих здібностей суперників у платформерній грі має ключове значення для створення глибокої та динамічної геймплейної механіки. Використання програмування на мові C# та бібліотек Unity дозволяє ефективно реалізувати складні інтерактивні здібності, які можуть впливати на стратегії гри.

Функція Skill Ця функція використовується для активації особливих здібностей суперників, таких як стрільба, невидимість або швидкісний ривок, які можуть бути запущені у відповідь на дії гравця або за певних умов у грі. Ця функція має такі параметри:

- enemy: об'єкт противника, який використовує здібність;
- skill_type: тип здібності, що активується (наприклад, "fireball", "stealth"); cooldown: часовий інтервал (у секундах), протягом якого здібність не може бути повторно використана після її активації.

Константи та параметри здібностей противника: У цьому розділі коду визначаються різні константи та параметри, які використовуються для контролю поведінки здібностей. Наприклад, MAX_RANGE - це максимальна дистанція, на якій здібність може бути ефективною, а ENERGY_COST - кількість енергії, яка витрачається на використання здібності.

Активація здібностей: У цьому розділі коду виконується ряд операцій, пов'язаних з активацією здібностей:

- перевірка, чи дозволяє cooldown активувати здібність в даному моменті;
- перевірка, чи достатньо енергії у противника для використання здібності;
- обчислення впливу здібності на ігрове середовище та гравця, якщо здібність була активована;
- оновлення статусу здібності, включаючи оновлення cooldown та ENERGY_COST.

3.4 Висновки

У третьому розділі проведено обґрунтування вибору мови програмування та технологій для розробки системи моделювання поведінки суперників у платформерній грі. Після аналізу потреб програмного продукту було вирішено використовувати мову програмування C# у зв'язці з ігровим двигуном Unity через їх високу продуктивність і гнучкість у розробці ігор. Unity забезпечує потужні інструменти для візуалізації та анімації, що є критично важливим для динамічного відображення поведінки суперників.

Для зручності розробки графічного інтерфейсу та взаємодії з користувачами вибрано фреймворк Unity UI, оскільки він інтегрований безпосередньо в Unity і дозволяє ефективно створювати інтуїтивно зрозумілі інтерфейси. Щодо моделювання поведінки та навчання моделей машинного навчання, було вирішено використовувати ML-Agents Toolkit, який надає

широкі можливості для реалізації навчання з підкріпленням та інших технік штучного інтелекту в ігровому середовищі.

4 ТЕСТУВАННЯ ПРОГРАМИ

4.1 Аналіз методів тестування

Тестування програмного забезпечення є невід'ємною частиною процесу розробки, маючи на меті забезпечення стабільності, якості та ефективності функціонування гри. Залежно від цілей та ресурсів, використовуються різноманітні стратегії тестування, кожна з яких відіграє ключову роль у виявленні помилок і несправностей.

Серед найбільш поширених видів тестування можна виділити такі:

- Тестування чорною скринькою (Black Box Testing): цей метод передбачає перевірку функціональності гри без доступу до внутрішньої структури коду. Тестувальник фокусується на тому, як програма реагує на вхідні дані та які виходять результати, не вникаючи в деталі реалізації.

- Тестування білою скринькою (White Box Testing): у цьому випадку тестувальник має доступ до вихідного коду гри і може проводити більш глибокий аналіз, створюючи тести, які перевіряють внутрішню логіку та структуру програми.

- Тестування сірою скринькою (Grey Box Testing): цей метод є компромісом між чорною та білою скринькою. Тестувальник має обмежений доступ до деяких аспектів внутрішньої структури програми, що дозволяє поєднувати переваги обох підходів для ефективнішого виявлення помилок.

Також, існує багато різновидів методів тестування, кожен з яких призначений для аналізу окремих аспектів програмного забезпечення:

- Функціональне тестування – аналізує, наскільки добре програма виконує свої основні функції з перспективи користувача;

- Тестування відповідності специфікаціям – перевіряє, чи відповідає програма всім визначеним вимогам;

- Тестування продуктивності – вимірює швидкодію та ефективність програми при великому навантаженні;

- Тестування на стійкість до збоїв – оцінює поведінку програми під час збоїв або помилок;

- Тестування безпеки – перевіряє, наскільки програма захищена від потенційних атак;

- Тестування на можливість відновлення – аналізує здатність програми продовжувати роботу після збою;

- Тестування сумісності – перевіряє, як програма працює на різних пристроях та операційних системах;

- Тестування на витривалість – вимірює роботу програми при тривалому використанні;

- Тестування на відновлення даних – тестує здатність програми відновлювати дані після несподіваних відключень;

- Тестування на зручність користування – оцінює інтерфейс і легкість використання програми;

- Тестування на наявність помилок – перевіряє на наявність та обробку помилок в програмі;

- Тестування стабільності – оцінює надійність і стабільність роботи програми.

Для тестування новорозробленої комп’ютерної гри було вибрано метод тестування «Чорна скринька».

4.1 Тестування системи

Модульне тестування в системі моделювання поведінки суперників має ключове значення, оскільки забезпечує перевірку окремих компонентів системи, таких як алгоритми визначення стратегій суперників або реакції на дії гравців. Це тестування виконується за допомогою автоматизованих тестів, які вводять у систему специфічні тестові дані для кожного модуля та перевіряють

очікувані результати, що дозволяє ізолювати та виправити будь-які виявлені помилки.

Системне тестування охоплює перевірку системи в цілому, включаючи всі її функції та можливості, доступні користувачам. Воно здійснюється шляхом відтворення різних ігрових сценаріїв, щоб перевірити реакцію системи на комплексні вхідні дані та аналізувати її загальну продуктивність. Це тестування допомагає виявити проблеми, що можуть виникнути лише при взаємодії різних компонентів системи разом[18].

Приймальне тестування, як правило, проводиться замовником або кінцевими користувачами з метою підтвердження відповідності системи встановленим вимогам та її готовності до впровадження в реальні ігрові умови. Цей етап є критично важливим для забезпечення того, що програмний продукт здатен задовольнити очікування та потреби користувачів[19].

Результати всіх тестувань представлені на рисунку (рис. 4.1-4.3).

Рисунок 4.1 – Екран гравця

Інтеграційне тестування (рис. 4.2) є ключовим елементом у процесі розробки системи моделювання поведінки суперників, оскільки забезпечує перевірку взаємодії між різними компонентами системи. Цей вид тестування допомагає визначити, чи правильно взаємодіють між собою такі елементи, як алгоритми штучного інтелекту, ігровий двигун, база даних та користувацький інтерфейс[20].

Рисунок 4.2 – Меню паузи

На цьому рисунку зображено екран паузи на який гравець може натиснути в будь який ігровий момент щоб зупинити сеанс. Кнопка паузи знаходиться в верхньому правому кутку екрана, що дозволяє легко орієнтуватись і в потрібний момент її використати.

Рисунок 4.3 – Екран смерті гравця

На рисунку 4.3 можем бачити екран смерті гравця, він з’являється коли противнки доводять показник здоров’я до нуля

4.2 Розробка інструкції користувача

Інструкція користувача передбачає визначення технічних вимог для запуску програмного продукту [21]. Деталі щодо мінімальної та

рекомендованої конфігурації серверного комп’ютера можна знайти в таблиці 4.1.

Таблиця 4.1 – Мінімальна конфігурація:

Технічні вимоги	Мінімальні	Рекомендовані
Процесор	Двоядерний, 1.5 ГГц	Чотирьохядерний, 2.5 ГГц або вище
Оперативна пам'ять (RAM)	2 ГБ	8 ГБ або більше
Місце на диску	500 МБ	2 ГБ або більше
Операційна система	Windows 7 або новіше	Windows 10
Відеокарта	Не менше 2ГБ	Більше 4ГБ

1. Завантаження Ігрових Рівнів: Користувачам доступний розділ для завантаження ігрових рівнів.

2. Вибір та Запуск Симуляції: Після завантаження гри, користувач може вибрати, які саме рівні він бажає активувати для тестування поведінки суперників.

4. Взаємодія з Користувачем та Підтримка: У разі виникнення будь-яких питань або проблем під час використання системи, користувачеві надається можливість зв'язатися з адміністраторами системи через спеціальну форму зворотного зв'язку або електронну пошту[22]. Це забезпечує оперативне вирішення проблем та підтримку користувачів на всіх етапах використання системи.

Ця інструкція призначена для забезпечення зручного та ефективного користування системою розробниками ігор та гейм-дизайнерами будь-якого рівня досвіду, допомагаючи їм тестувати та оптимізувати поведінку суперників у платформерних іграх.

4.3 Висновки

У четвертому розділі курсового проєкту було проведено функціональне тестування, спрямоване на перевірку основних функцій системи, зокрема механізмів штучного інтелекту для управління поведінкою суперників. Основна увага була зосереджена на користувацькому інтерфейсу, щоб більш детільніше заглибитись в можливості які, задані розробниками.

ВИСНОВКИ

У бакалаврській роботі було розроблено програмні засоби реалізації системи моделювання поведінки суперників у платформерній грі. Для розробки було обрано середовище програмної розробки VSCode, яке є гнучким і підтримує різні мови програмування, зокрема C#, що використовувався для написання більшості скриптів моделі.

Під час виконання курсового проєкту було здійснено аналіз сучасних тенденцій у розробці ігрових систем. Були вивчені основні аналоги та існуючі методи моделювання поведінки суперників. Визначено ключові недоліки існуючих рішень і порівняно їх з розроблюваним програмним продуктом. Відповідно до отриманих результатів аналізу були сформульовані основні задачі проєкту.

У процесі вибору технологій для розробки було вирішено використовувати C# для реалізації алгоритмів. Для зберігання даних і взаємодії з користувачем було вибрано веб-фреймворк Django, завдяки його високій продуктивності та безпеці.

Було створено загальні схеми алгоритму функціонування системи та докладно описано програмний продукт. Розроблена система була успішно інтегрована в ігровий процес, що підтверджено тестуванням продукту. Також була підготовлена деталь на інструкція для користувачів щодо використання системи.

ЛІТЕРАТУРА

1. Кушнір М.М. Програмної платформа моделювання поведінки суперників у платформених іграх. Електронні інформаційні ресурси: створення, використання, доступ. Збірник матеріалів Міжнародної науково-практичної Інтернет конференції 22-25 травня 2024 р. – Харків: НТУ «Харківський політехнічний інститут», 2024. – С. 1250.
2. Юліан Тогеліус, Нур Шакер, Стефано Каньйоні та ін. (2020). "Застосування еволюційних обчислень у іграх". *Еволюційне обчислення*.
3. Millington, I., & Funge, J. (2009). *Artificial Intelligence for Games*. CRC Press.
4. Rabin, S. (Ed.). (2013). *Game AI Pro: Collected Wisdom of Game AI Professionals*. CRC Press.
5. Yannakakis, G. N., & Togelius, J. (2018). *Artificial Intelligence and Games*. Springer.
6. Buckland, M. (2005). *Programming Game AI by Example*. Wordware Publishing.
7. Champandard, A. J. (2003). *AI Game Development: Synthetic Creatures with Learning and Reactive Behaviors*. New Riders.
8. Isla, D. (2005). Handling complexity in the Halo 2 AI. In *Game Developers Conference*.
9. Sweetser, P. (2008). Emergent storytelling in games. *The International Journal of Computer Game Research*, 8(1).
10. Laird, J. E., & van Lent, M. (2001). Human-level AI's killer application: Interactive computer games. *AI Magazine*, 22(2), 15-25.
11. Woodcock, S. (2001). Game AI: The state of the industry. *Game Developer Magazine*, August.
12. Tozour, P. (2001). The Evolution of Game AI. In *AI Game Programming Wisdom* (pp. 3-15). Charles River Media.
13. Michael, D. & Chen, S. (2005). *Serious Games: Games That Educate, Train and Inform*. Thomson Course Technology.

14. Russell, S., & Norvig, P. (2020). *Artificial Intelligence: A Modern Approach* (4th ed.). Pearson.
15. Unity Technologies. (2020). *Unity Documentation*. [Online]. Available: https://docs.unity3d.com/Manual/index.html
16. Unreal Engine. (2020). *Unreal Engine Documentation*. [Online]. Available: https://docs.unrealengine.com/
17. Microsoft. (2020). *Microsoft Game Development*. [Online]. Available: https://developer.microsoft.com/en-us/games/
18. Bartle, R. (2004). *Designing Virtual Worlds*. New Riders.
19. Charles, F., & Cavazza, M. (2004). AI Planning techniques for virtual storytelling. *IEEE Computer Graphics and Applications*.
20. van der Sterren, W. (2002). Squad tactics: Team AI and positional warfare. In *AI Game Programming Wisdom* (pp. 233-246). Charles River Media.
21. Journal of Game Development. (2005). *JOGD.*
22. Reeves, L., Sherrod, A. (2010). *Beginning Game Programming* (4th ed.). Course Technology PTR.
23. Лисенко Г.Л. Методичні вказівки до оформлення курсових проектів (робіт) у Вінницькому національному технічному університеті /Уклад. Г.Л. Лисенко, А.Г. Буда, Р.Р. Обертюх, – Вінниця: ВНТУ, 2006. – 60 с.
24. Методичні вказівки до виконання курсових проєктів з дисципліни "Проєктний практикум" для студентів спеціальності 121 "Інженерія програмного забезпечення"/Уклад. В.В. Войтко, Г. О. Черноволик – Вінниця: ВНТУ, 2022. – 44с

ДОДАТКИ

Додаток А. Технічне завдання

Міністерство освіти і науки України
Вінницький національний технічний університет
факультет інформаційних технологій та комп'ютерної інженерії

ЗАТВЕРДЖУЮ
д.т.н., проф. О. Н. Романюк
«12» березня 2024р.

Технічне завдання
на бакалаврську дипломну роботу «Програмна платформа моделювання поведінки суперників у платформених іграх» за спеціальністю 121 – Інженерія програмного забезпечення

Керівник бакалаврської дипломної роботи:
_____ к.т.н., доцент Р.Ю. Чехместрук
«12» березня 2024р.

Виконав:
_____студент гр.1ПІ-20Б М.М. Кушнір
«12» березня 2024р.

Вінниця – 2024 року

1. Найменування та галузь застосування

Бакалаврська дипломна робота: «Програмна платформа для моделювання поведінки суперників у платформених відеоіграх». Галузь застосування - розробка відеоігор.

2. Підстава для розробки.

Підставою для виконання бакалаврської дипломної роботи (БДР) є індивідуальне завдання на БДР та наказ №80 від 11 березня 2024р ректора університету про закріплення тем БДР.

3. Мета та призначення розробки.

Метою роботи є підвищення рівня іммерсивності відеоігор шляхом створення реалістичніших і складніших моделей поведінки суперників, що дозволяє забезпечити більш глибокий і залучаючий ігровий досвід.

Призначення роботи – розробка програмної платформи для моделювання поведінки суперників у платформених відеоіграх.

3 Вихідні дані для проведення НДР

Перелік основних літературних джерел, на основі яких буде виконуватись МКР.

1. Хорошилов А.А., Сорокін А. Н. Методи моделювання інтелекту в ігрових застосуваннях / А. А. Хорошилов, А. Н. Сорокін // Журнал прикладної інформатики. — 2009. — № 3. — С. 102—110.
2. Павлов А.В. Огляд алгоритмів штучного інтелекту у відеоіграх / А. В. Павлов // Наукові записки університету ім. Каразіна. — 2010. — № 11. — С. 75—81.
3. Бєлікова С.Ю. Теорія та практика програмування штучного інтелекту для відеоігор. Монографія. / С. Ю. Бєлікова. - Харків : Форт, 2011. — 300 с.

4. Технічні вимоги

Основні алгоритми моделювання поведінки – стейт-машини, поведінкові дерева; методики реалізації штучного інтелекту – навігаційні сітки, планувальники дій; графічний режим – HD, 2D; вихідні дані для моделювання – параметри персонажів, інтерактивні об'єкти; дані про зони впливу, територіальне розташування; коефіцієнт агресивності суперників; координати та характеристики об'єктів сцен; максимальна кількість одночасних об'єктів на сцені – до 100; вихідні дані – кінцеве зображення, взаємодія персонажів, реакції на дії гравця.

5. Конструктивні вимоги.

Конструкція пристрою повинна відповідати естетичним та ергономічним вимогам, повинна бути зручною в обслуговуванні та керуванні.

Графічна та текстова документація повинна відповідати діючим стандартам України.

6. Перелік технічної документації, що пред’являється по закінченню робіт:

– пояснювальна записка до БДР;
– технічне завдання;
– лістинги програми.

8. Вимоги до рівня уніфікації та стандартизації

При розробці програмних засобів слід дотримуватися уніфікації і ДСТУ.

9**. Стадії та етапи розробки:**

№ з/п	Назва етапів бакалаврської дипломної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз стану питання, аналогів та методів	14.03.2024 – 27.03.2024	Виконано
2	Розробка інтерфейсів та моделей	28.03.2024 – 27.04.2024	Виконано
3	Розробка програмного забезпечення	28.04.2024 – 23.05.2024	Виконано
4	Тестування програми	24.05.2024 – 30.05.2024	Виконано

10. Порядок контролю та прийняття.

Виконання етапів бакалаврської дипломної роботи контролюється керівником згідно з графіком виконання роботи. Прийняття бакалаврської дипломної роботи здійснюється ДЕК, затвердженою зав. кафедрою згідно з графіком

Додаток Б – Протокол перевірки на плагіат

61

ПРОТОКОЛ
ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ
НА НАЯВНІСТЬ ТЕКСТОВИХ ЗАПОЗИЧЕНЬ

Назва роботи: Програмна платформа моделювання поведінки суперників у платформених іграх

Тип роботи: БДР

Підрозділ : кафедра програмного забезпечення, ФІТКІ

Науковий керівник: Чехместрук Р. Ю.

Оригінальність	**98%**
Схожість	**2%**

Аналіз звіту подібності

■ **Запозичення, виявлені у роботі, оформлені коректно і не містять ознак плагіату.**

□ Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її автора. Роботу направити на доопрацювання.

□ Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

Особа, відповідальна за перевірку _____ Черноволик Г. О.

Ознайомлені з повним звітом подібності, який був згенерований системою Unicheck

Автор роботи _____ Кушнір М. М.

Керівник роботи _____ Чехместрук Р. Ю.

Додаток В. Лістинг програми

HeroBase.cs

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

public class HeroBase : SingletonComponent<HeroBase>
{
    [SerializeField] protected GameObject dieWindow;
    [SerializeField] protected float detectEmemyRadius = 3f;

    [Header("Animation")]
    [SerializeField] protected Animator heroAnimator;
    public Transform hero;
    //public Character hero;

    [Header("Skills")]
    [SerializeField] private GameObject _bow;
    public GameObject Bow => _bow;
    [SerializeField] private SpriteRenderer _arrowPosition;
    public SpriteRenderer ArrowPosition => _arrowPosition;

    protected SkillBase basicSkill;
    protected SkillBase specialSkill;
    [SerializeField] protected SkillReload spesialSkillReload;
    [SerializeField] protected List<SkillBase> basicSkills = new List<SkillBase>();
    [SerializeField] protected List<SkillBase> specialSkills = new List<SkillBase>();
    [SerializeField] protected ParticleSystem warriorBasicAttackEffect;
    [SerializeField] protected ParticleSystem warriorSpecialAttackEffect;
    [SerializeField] protected ParticleSystem moveEffectPrefab;
    [SerializeField] protected float spawnMoveEffectDelay = 1f;

    protected Coroutine spawnMoveEffectCoroutine;

    [Header("Joystick")]
    [SerializeField] private List<GameObject> focusParts = new List<GameObject>();

    [Header("Stats")]
    [SerializeField] protected float speed;
    public float physicalDamage;
    public float magicDamage;
```

```
        protected float physicalDefense;
        protected float magicDefense;

        [Header("HP")]
        [SerializeField] protected Slider HPSlider;
        [SerializeField] protected TextMeshProUGUI HPText;

        [Header("Resurrect")]
        [SerializeField] protected float _invulnerabilityTime = 3f;

        protected HealthManager HP;

        protected HeroActions heroInput;
        [System.NonSerialized] public Vector2 currentDirection = Vector2.right;

        protected bool flip = false;
        protected bool blockAnyDamage = false;

        public bool isDie = false;
        public bool isGameOver = false;

        public bool readyToOpenChest = false;

        protected HeroFraction currentFraction;

        protected void Start()
        {
            currentFraction = PlayerPrefs.GetInt("CurrentHero") == (int)HeroFraction.Warrior ? HeroFraction.Warrior : PlayerPrefs.GetInt("CurrentHero") == (int)HeroFraction.Archer ? HeroFraction.Archer : HeroFraction.Wizard;

            if (currentFraction != HeroFraction.Null)
            {
                basicSkill = basicSkills[(int)currentFraction - 1];
                specialSkill = specialSkills[(int)currentFraction - 1];
            }

            HP = new HealthManager();
            //HP.MaxHealth = PlayerPrefs.GetInt("Health");
            //HP.CurrentHealth = PlayerPrefs.GetInt("Health");
            //physicalDamage = PlayerPrefs.GetInt("PhysicalDamage");
            //magicDamage = PlayerPrefs.GetInt("MagicDamage");
            //physicalDefense = PlayerPrefs.GetInt("PhysicalDefense");
            //magicDefense = PlayerPrefs.GetInt("MagicDefense");

            HP.MaxHealth = 1600;
```

```
    HP.CurrentHealth = 1600;
    physicalDamage = 300;
    magicDamage = 600;
    physicalDefense = 220;
    magicDefense = 240;

    HPSlider.value = 1;
    HPText.text = HP.CurrentHealth + "/" + HP.MaxHealth;
}

protected void OnEnable()
{
    heroInput = new HeroActions();
    heroInput.Enable();
}

protected void OnDisable()
{
    if (heroInput != null)
        heroInput.Disable();
}

protected void Update()
{
    if (isDie)
        return;

    if (!isGameOver)
        Move();

    if(!basicSkill.isActive && !specialSkill.isActive)
    {
        if (heroInput.Player.BasicAttack.IsPressed())
        {
            ActiveBasicAttackEffect();

            basicSkill.ActiveSkill(heroAnimator);
        }

        if (!spesialSkillReload.IsReloading && heroInput.Player.SpecialAttack.triggered)
        {
            ActiveSpecialAttackEffect();

            specialSkill.ActiveSkill(heroAnimator);
            spesialSkillReload.StartReload(specialSkill.ReloadTime);
        }
```

```
        }
    }

    protected void Move()
    {
        var inputDirection = heroInput.Player.Move.ReadValue<Vector2>();

        if (inputDirection != Vector2.zero)
        {
            if (!SoundManager.Instance.isMoveSoundPlaying)
            {
                spawnMoveEffectCoroutine = StartCoroutine(SpawnMoveEffect());
                SoundManager.Instance.StartPlayMoveSound();
            }

                                                    var nearEnemyDirection =
EnemyContainer.Instance.IsEnemyNearby(detectEmemyRadius);

                flip = nearEnemyDirection == Vector2.left ? true : nearEnemyDirection ==
Vector2.right ? false : inputDirection.x > 0 ? false : inputDirection.x < 0 ? true :  flip;

            heroAnimator.SetBool("IsWalking", true);

            hero.GetComponent<Rigidbody2D>().velocity = inputDirection * speed;

              currentDirection = nearEnemyDirection != Vector2.zero ? nearEnemyDirection :
inputDirection.x < 0 ? Vector2.left : inputDirection.x > 0 ? Vector2.right : currentDirection;

            int setFocus = inputDirection.x > 0 && inputDirection.y < 0 ? 1 : inputDirection.x <
0 && inputDirection.y < 0 ? 2 : inputDirection.x < 0 && inputDirection.y > 0 ? 3 : 0;

            if (setFocus != -1)
                focusParts[setFocus].SetActive(true);

            for (int i = 0; i < focusParts.Count; i++)
                if (i != setFocus)
                    focusParts[i].SetActive(false);
        }

        else
        {
            if (SoundManager.Instance.isMoveSoundPlaying)
            {
                StopCoroutine(spawnMoveEffectCoroutine);
                SoundManager.Instance.StopPlayMoveSound();
            }
```

```
            heroAnimator.SetBool("IsWalking", false);
            focusParts.ForEach( focus => focus.SetActive(false) );
        }

        var scaleX = flip ? -Mathf.Abs(hero.localScale.x) : Mathf.Abs(hero.localScale.x);
        hero.localScale = new Vector3(scaleX, hero.localScale.y, hero.localScale.z);
    }

    private IEnumerator SpawnMoveEffect()
    {
        while(true)
        {
            yield return new WaitForSeconds(spawnMoveEffectDelay);
            Instantiate(moveEffectPrefab, hero.position, Quaternion.identity);
        }
    }

    public void TakeDamage(int physicalDamage, int magicDamage)
    {
        if (blockAnyDamage)
            return;

          var damage = ((physicalDamage - this.physicalDefense) >= 0 ? (physicalDamage -
this.physicalDefense) : 0) + ((magicDamage - this.magicDefense) >= 0 ? (magicDamage -
this.magicDefense) : 0);

        damage = Mathf.Clamp(damage, 1, 10000000000);

        if (HP.CurrentHealth - damage <= 0 && !isGameOver)
        {
            SoundManager.Instance.PlayDeathSound();
            HP.CurrentHealth = 0;
            HPSlider.value = 0;
            HPText.text = "0";

            heroAnimator.SetBool("Death", true);

            Invoke(nameof(ShowGameOverScreen), 3);
            isDie = true;
            isGameOver = true;

            return;
        }

        SoundManager.Instance.PlayTakeDamageSound();
```

```
    HP.CurrentHealth -= damage;
    HPSlider.value = HP.CurrentHealth / HP.MaxHealth;
    HPText.text = HP.CurrentHealth + "/" + HP.MaxHealth;
}

void ShowGameOverScreen()
{
    dieWindow.SetActive(true);
}

public void Resurrect()
{
    isDie = false;
    isGameOver = false;
    HP.CurrentHealth = HP.MaxHealth;

    HPSlider.value = 1;
    HPText.text = HP.CurrentHealth + "/" + HP.MaxHealth;

    heroAnimator.SetTrigger("Blinking");

    ActivateInvulnerability();
}

public void ActivateInvulnerability()
{
    blockAnyDamage = true;
}

public void DeactivateInvulnerability()
{
    heroAnimator.SetTrigger("ResetBlinking");
    blockAnyDamage = false;
}

public void ActiveBasicAttackEffect()
{
    if (currentFraction == HeroFraction.Warrior)
        warriorBasicAttackEffect.Play();
}

public void ActiveSpecialAttackEffect()
{
    if (currentFraction == HeroFraction.Warrior)
        warriorSpecialAttackEffect.Play();
```

```
    }
}
```

HeroActions.cs

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

public partial class @HeroActions: IInputActionCollection2, IDisposable
{
    public InputActionAsset asset { get; }
    public @HeroActions()
    {
        asset = InputActionAsset.FromJson(@"{
    ""name"": ""HeroActions"",
    ""maps"": [
        {
            ""name"": ""Player"",
            ""id"": ""f9dbae51-b9bc-4997-86c3-1f182377bf5a"",
            ""actions"": [
                {
                    ""name"": ""Move"",
                    ""type"": ""Value"",
                    ""id"": ""518bb448-e192-45a8-8b03-c9c78a4b0b8a"",
                    ""expectedControlType"": ""Vector2"",
                    ""processors"": """",
                    ""interactions"": """",
                    ""initialStateCheck"": true
                },
                {
                    ""name"": ""BasicAttack"",
                    ""type"": ""Button"",
                    ""id"": ""04331a8d-2591-4f79-b8a4-fefadbcbc52c"",
                    ""expectedControlType"": ""Button"",
                    ""processors"": """",
                    ""interactions"": """",
                    ""initialStateCheck"": false
```

```
    },
    {
      ""name"": ""SpecialAttack"",
      ""type"": ""Button"",
      ""id"": ""39d34717-174b-413c-839a-09e64ecc42bb"",
      ""expectedControlType"": ""Button"",
      ""processors"": """",
      ""interactions"": """",
      ""initialStateCheck"": false
    },
    {
      ""name"": ""UseElixir"",
      ""type"": ""Button"",
      ""id"": ""7529d084-3c47-411d-b918-bb654dea931b"",
      ""expectedControlType"": ""Button"",
      ""processors"": """",
      ""interactions"": """",
      ""initialStateCheck"": false
    }
  ],
  ""bindings"": [
    {
      ""name"": ""WASD"",
      ""id"": ""03b729ac-3211-449a-ba62-9ea7f14893b1"",
      ""path"": ""2DVector"",
      ""interactions"": """",
      ""processors"": """",
      ""groups"": """",
      ""action"": ""Move"",
      ""isComposite"": true,
      ""isPartOfComposite"": false
    },
    {
      ""name"": ""up"",
      ""id"": ""31e67c16-3a31-444d-9759-245e8ba67c9f"",
      ""path"": ""<Keyboard>/w"",
      ""interactions"": """",
      ""processors"": """",
      ""groups"": ""KeyboardMouse"",
      ""action"": ""Move"",
      ""isComposite"": false,
      ""isPartOfComposite"": true
    },
    {
      ""name"": ""down"",
      ""id"": ""70ed1c6a-8016-40f6-9aaf-46e26f282532"",
```

```
    ""path"": ""<Keyboard>/s"",
    ""interactions"": """",
    ""processors"": """",
    ""groups"": ""KeyboardMouse"",
    ""action"": ""Move"",
    ""isComposite"": false,
    ""isPartOfComposite"": true
  },
  {
    ""name"": ""left"",
    ""id"": ""1004f2ae-0e5e-4728-8802-63ec50ca5f46"",
    ""path"": ""<Keyboard>/a"",
    ""interactions"": """",
    ""processors"": """",
    ""groups"": ""KeyboardMouse"",
    ""action"": ""Move"",
    ""isComposite"": false,
    ""isPartOfComposite"": true
  },
  {
    ""name"": ""right"",
    ""id"": ""282f226d-37a6-47e5-9236-8dfc56b70a01"",
    ""path"": ""<Keyboard>/d"",
    ""interactions"": """",
    ""processors"": """",
    ""groups"": ""KeyboardMouse"",
    ""action"": ""Move"",
    ""isComposite"": false,
    ""isPartOfComposite"": true
  },
  {
    ""name"": """",
    ""id"": ""663b9c20-91da-4666-b32a-21954981ba75"",
    ""path"": ""<AndroidJoystick>/stick"",
    ""interactions"": """",
    ""processors"": """",
    ""groups"": """",
    ""action"": ""Move"",
    ""isComposite"": false,
    ""isPartOfComposite"": false
  },
  {
    ""name"": """",
    ""id"": ""ef94ff80-c36a-4a86-967c-81d1d3c2069d"",
    ""path"": ""<AndroidGamepad>/buttonEast"",
    ""interactions"": """",
```

```
    ""processors"": """",
    ""groups"": """",
    ""action"": ""BasicAttack"",
    ""isComposite"": false,
    ""isPartOfComposite"": false
  },
  {
    ""name"": """",
    ""id"": ""f3054c0e-7ab8-49de-9590-fd0d2ce58b29"",
    ""path"": ""<Keyboard>/space"",
    ""interactions"": """",
    ""processors"": """",
    ""groups"": """",
    ""action"": ""BasicAttack"",
    ""isComposite"": false,
    ""isPartOfComposite"": false
  },
  {
    ""name"": """",
    ""id"": ""611bcd04-7c77-43af-8963-456310d91243"",
    ""path"": ""<AndroidGamepad>/buttonNorth"",
    ""interactions"": """",
    ""processors"": """",
    ""groups"": """",
    ""action"": ""SpecialAttack"",
    ""isComposite"": false,
    ""isPartOfComposite"": false
  },
  {
    ""name"": """",
    ""id"": ""a729d548-468d-4b1a-81a2-4d67d7224739"",
    ""path"": ""<Keyboard>/n"",
    ""interactions"": """",
    ""processors"": """",
    ""groups"": """",
    ""action"": ""SpecialAttack"",
    ""isComposite"": false,
    ""isPartOfComposite"": false
  },
  {
    ""name"": """",
    ""id"": ""a8eb3dd1-0d92-4708-8446-10eb67edec46"",
    ""path"": ""<AndroidGamepad>/rightTrigger"",
    ""interactions"": """",
    ""processors"": """",
    ""groups"": """",
```

```
                ""action"": ""UseElixir"",
                ""isComposite"": false,
                ""isPartOfComposite"": false
            },
            {
                ""name"": """",
                ""id"": ""ac6f3ffb-e0af-4361-abf5-77e252005d56"",
                ""path"": ""<Keyboard>/m"",
                ""interactions"": """",
                ""processors"": """",
                ""groups"": """",
                ""action"": ""UseElixir"",
                ""isComposite"": false,
                ""isPartOfComposite"": false
            }
        ]
    }
],
""controlSchemes"": []
}");
        // Player
        m_Player = asset.FindActionMap("Player", throwIfNotFound: true);
        m_Player_Move = m_Player.FindAction("Move", throwIfNotFound: true);
        m_Player_BasicAttack = m_Player.FindAction("BasicAttack", throwIfNotFound: true);
        m_Player_SpecialAttack = m_Player.FindAction("SpecialAttack", throwIfNotFound: true);
        m_Player_UseElixir = m_Player.FindAction("UseElixir", throwIfNotFound: true);
    }

    public void Dispose()
    {
        UnityEngine.Object.Destroy(asset);
    }

    public InputBinding? bindingMask
    {
        get => asset.bindingMask;
        set => asset.bindingMask = value;
    }

    public ReadOnlyArray<InputDevice>? devices
    {
        get => asset.devices;
        set => asset.devices = value;
    }
```

```
        public ReadOnlyArray<InputControlScheme> controlSchemes => asset.controlSchemes;

        public bool Contains(InputAction action)
        {
            return asset.Contains(action);
        }

        public IEnumerator<InputAction> GetEnumerator()
        {
            return asset.GetEnumerator();
        }

        IEnumerator IEnumerable.GetEnumerator()
        {
            return GetEnumerator();
        }

        public void Enable()
        {
            asset.Enable();
        }

        public void Disable()
        {
            asset.Disable();
        }

        public IEnumerable<InputBinding> bindings => asset.bindings;

        public InputAction FindAction(string actionNameOrId, bool throwIfNotFound = false)
        {
            return asset.FindAction(actionNameOrId, throwIfNotFound);
        }

        public int FindBinding(InputBinding bindingMask, out InputAction action)
        {
            return asset.FindBinding(bindingMask, out action);
        }

        // Player
        private readonly InputActionMap m_Player;
                private List<IPlayerActions> m_PlayerActionsCallbackInterfaces = new
List<IPlayerActions>();
        private readonly InputAction m_Player_Move;
        private readonly InputAction m_Player_BasicAttack;
```

```
            private readonly InputAction m_Player_SpecialAttack;
            private readonly InputAction m_Player_UseElixir;
            public struct PlayerActions
            {
                private @HeroActions m_Wrapper;
                public PlayerActions(@HeroActions wrapper) { m_Wrapper = wrapper; }
                public InputAction @Move => m_Wrapper.m_Player_Move;
                public InputAction @BasicAttack => m_Wrapper.m_Player_BasicAttack;
                public InputAction @SpecialAttack => m_Wrapper.m_Player_SpecialAttack;
                public InputAction @UseElixir => m_Wrapper.m_Player_UseElixir;
                public InputActionMap Get() { return m_Wrapper.m_Player; }
                public void Enable() { Get().Enable(); }
                public void Disable() { Get().Disable(); }
                public bool enabled => Get().enabled;
                public static implicit operator InputActionMap(PlayerActions set) { return set.Get(); }
                public void AddCallbacks(IPlayerActions instance)
                {
                                                        if (instance == null ||
m_Wrapper.m_PlayerActionsCallbackInterfaces.Contains(instance)) return;
                    m_Wrapper.m_PlayerActionsCallbackInterfaces.Add(instance);
                    @Move.started += instance.OnMove;
                    @Move.performed += instance.OnMove;
                    @Move.canceled += instance.OnMove;
                    @BasicAttack.started += instance.OnBasicAttack;
                    @BasicAttack.performed += instance.OnBasicAttack;
                    @BasicAttack.canceled += instance.OnBasicAttack;
                    @SpecialAttack.started += instance.OnSpecialAttack;
                    @SpecialAttack.performed += instance.OnSpecialAttack;
                    @SpecialAttack.canceled += instance.OnSpecialAttack;
                    @UseElixir.started += instance.OnUseElixir;
                    @UseElixir.performed += instance.OnUseElixir;
                    @UseElixir.canceled += instance.OnUseElixir;
                }

                private void UnregisterCallbacks(IPlayerActions instance)
                {
                    @Move.started -= instance.OnMove;
                    @Move.performed -= instance.OnMove;
                    @Move.canceled -= instance.OnMove;
                    @BasicAttack.started -= instance.OnBasicAttack;
                    @BasicAttack.performed -= instance.OnBasicAttack;
                    @BasicAttack.canceled -= instance.OnBasicAttack;
                    @SpecialAttack.started -= instance.OnSpecialAttack;
                    @SpecialAttack.performed -= instance.OnSpecialAttack;
                    @SpecialAttack.canceled -= instance.OnSpecialAttack;
                    @UseElixir.started -= instance.OnUseElixir;
```

```
            @UseElixir.performed -= instance.OnUseElixir;
            @UseElixir.canceled -= instance.OnUseElixir;
        }

        public void RemoveCallbacks(IPlayerActions instance)
        {
            if (m_Wrapper.m_PlayerActionsCallbackInterfaces.Remove(instance))
                UnregisterCallbacks(instance);
        }

        public void SetCallbacks(IPlayerActions instance)
        {
            foreach (var item in m_Wrapper.m_PlayerActionsCallbackInterfaces)
                UnregisterCallbacks(item);
            m_Wrapper.m_PlayerActionsCallbackInterfaces.Clear();
            AddCallbacks(instance);
        }
    }
    public PlayerActions @Player => new PlayerActions(this);
    public interface IPlayerActions
    {
        void OnMove(InputAction.CallbackContext context);
        void OnBasicAttack(InputAction.CallbackContext context);
        void OnSpecialAttack(InputAction.CallbackContext context);
        void OnUseElixir(InputAction.CallbackContext context);
    }
}
```

BattleZone.cs

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

public class BattleZone : MonoBehaviour
{
    [SerializeField] private GameObject border;
    [SerializeField] private List<GameObject> _enemies = new List<GameObject>();
    [SerializeField] private bool _lastBattleZone = false;

    [SerializeField] float minSpawnCooldown = 2;
    [SerializeField] float maxSpawnCooldown = 5;
```

```
        [SerializeField] private List<GameObject> _spawnedEnemies = new List<GameObject>();
        [SerializeField] private List<GameObject> _enemiesToSpawn = new List<GameObject>();

    public int EnemiesCount => _spawnedEnemies.Count + _enemiesToSpawn.Count;

    Transform player;

    NavMeshPath meshPath;

    private void Start()
    {
        player = HeroBase.Instance.transform.GetChild(0);
        meshPath = new NavMeshPath();
    }

    public void StartBattle()
    {
        border.SetActive(true);

        foreach (var enemy in _enemies)
            enemy.GetComponent<Enemy>().Activate();
    }

    private void EndBattle()
    {
        border.SetActive(false);

        Invoke(nameof(PlayBarrierSound), 0.5f);

        if (_lastBattleZone && !HeroBase.Instance.isDie)
            Invoke(nameof(ShowResult), 1f);
    }

    private void ShowResult() => LevelHUD.Instance.ShowResultWindow();

    void PlayBarrierSound() => SoundManager.Instance.BarrierSound();
    public void AddEnemy(GameObject enemy)
    {
        _enemies.Add(enemy);
    }

    public void RemoveEnemy(GameObject enemy)
    {
        _enemies.Remove(enemy);
```

```
            if (_enemies.Count == 0 && _enemiesToSpawn.Count == 0)
            {
                EndBattle();
                return;
            }

                        Invoke(nameof(SpawnEnemy), Random.Range(minSpawnCooldown,
maxSpawnCooldown));
        }

        void SpawnEnemy()
        {
            if (_enemiesToSpawn.Count == 0)
                return;

            SoundManager.Instance.SpawnEnemySound();

            var enemy = Instantiate(_enemiesToSpawn[0], player.position, Quaternion.identity);

            _enemiesToSpawn.RemoveAt(0);

                                                    enemy.transform.position =
GetRandomPointInBattleZone(enemy.GetComponent<NavMeshAgent>(), 5);
            enemy.GetComponent<Enemy>().Activate();
            enemy.GetComponent<Enemy>().SetBattleZone(this);

            AddEnemy(enemy.gameObject);
        }

        Vector3 GetRandomPointInBattleZone(NavMeshAgent enemy, float maxDistance = 3)
        {
            NavMeshHit hit; // NavMesh Sampling Info Container
            Vector3 result = new Vector3();

            var playerPos = player.position;

            var playerZPosition = playerPos.z;

            do
            {
                // Get Random Point inside Sphere which position is center, radius is maxDistance
                Vector3 randomPos = Random.insideUnitSphere * maxDistance + playerPos;

                // from randomPos find a nearest point on NavMesh surface in range of maxDistance
                NavMesh.SamplePosition(randomPos, out hit, maxDistance, NavMesh.AllAreas);
```

```
            result = new Vector3(hit.position.x, hit.position.y, playerZPosition);

            enemy.CalculatePath(result, meshPath);

        } while (meshPath.status != NavMeshPathStatus.PathComplete);

        return result;
    }
}
```

Archer.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.AI;
using Assets.HeroEditor.Common.CharacterScripts;

public class Archer : Enemy
{
    [SerializeField] protected float saveZoneRadius = 1.5f;
    [SerializeField] protected float cooldown = 1;
    [SerializeField] float projjectileSpeed = 8;
    [SerializeField] GameObject projectilePrefab;
    [SerializeField] Transform projectileSpawnPoint;

    protected ObjectsPool objectsPool;

    bool isRecharging = false;

    protected override void VirtualStart()
    {
        base.VirtualStart();
        objectsPool = new ObjectsPool(3, projectilePrefab);
    }

    protected override void Atack()
    {
        var distanceToPlayer = Vector2.Distance(_transform.position, player.position);

        if (!canAtack || distanceToPlayer > atackRange || distanceToPlayer < saveZoneRadius)
            return;
```

```
            canAtack = false;

            animator.SetTrigger("Attack");
        }

        protected override void Fight(string s)
        {
            var playerPos = player.position;
            var pos = projectileSpawnPoint.position;

            var projectileDirection = (playerPos - pos).normalized;

            //Rotation
              var projectileRotation = Mathf.Atan(projectileDirection.y / projectileDirection.x) *
Mathf.Rad2Deg;
            projectileRotation = pos.x > playerPos.x ? projectileRotation + 180 : projectileRotation;

            var projectile = objectsPool.GetObject();

                                        //projectile.GetComponent<SpriteRenderer>().sprite =
arrow.GetComponent<SpriteRenderer>().sprite;
                        projectile.transform.SetPositionAndRotation(pos, Quaternion.Euler(0, 0,
projectileRotation));
            projectile.GetComponent<ProjectileController>().SetProjectileInfo(2, physicalDamage,
magicDamage, Vector3.zero, 0, ProjectileDeathAction, CollidedAction, true);
                        projectile.GetComponent<Rigidbody2D>().AddForce(projectileDirection *
projjectileSpeed);

            Invoke(nameof(Recharge), cooldown);

            atackAudio.Play();
        }

        protected void Recharge() => canAtack = true;

        protected override Vector3 FindMovementTarget()
        {
            var playerPos = player.position;
            var currentPos = _transform.position;

            var distanceToPlayer = Vector2.Distance(currentPos, playerPos);

            agent.stoppingDistance = 0;

            if (distanceToPlayer > atackRange)
```

```
            return playerPos;

        if(distanceToPlayer < saveZoneRadius)
        {
            var dirToPlayer = currentPos - playerPos;

            var newPos = currentPos + dirToPlayer;

            return newPos;
        }

        return currentPos;
    }

    protected void ProjectileDeathAction(GameObject obj) => objectsPool.AddObject(obj);

    protected void CollidedAction(GameObject obj) => ProjectileDeathAction(obj);
}
```

Додаток Г – Ілюстративна частина

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра програмного забезпечення

БАКАЛАВРСЬКА ДИПЛОМНА РОБОТА

На тему: Програмна платформа
моделювання поведінки противників у платформених іграх

Виконав:
студент групи 1ПІ-20Б Кушнір М.М.

Науковий керівник:
к.т.н., доц. каф. ПЗ Чехместрук Р.Ю.

Рисунок Г.1 – Титульна сторінка

Мета, предмет та об'єкт дослідження

Мета роботи :
є дослідження є проектування та реалізація платформерної гри, що включає розробку інноваційних методів моделювання поведінки противників. Це передбачає створення комплексної ігрової системи, здатної забезпечити високий рівень взаємодії та виклику для гравців.

Об'єктом дослідження :
є процес розробки 2D гри з використанням Unity.

Предметом дослідження :
є методи та засоби реалізації платформерної гри для моделювання поведінки противників.

Рисунок Г.2 – Мета, об’єкт та предмет роботи

Рисунок Г.3 – Постановка задач

Рисунок Г.4 – Наукова новизна розробки

Аналіз аналогів

Критерії	Hell is Other Demons	Dust: An Elysian Tail	Castlevania: Symphony of the Night	Rayman 3: Hoodlum Havoc	Власна розробка
Казуальність	-	+	-	+	+
Перегравання	+	-	+	-	+
Варіятивність	+	-	+	+	+
Поведінка противників	+	-	-	+	+
Різнобарвність противників	-	+	-	-	+

Рисунок Г.5 – Аналіз аналогів

Практична цінність

Практична цінність проекту проявляється у поліпшенні якості ігрового досвіду, забезпечуючи гравцям більш складні та реалістичні виклики.

Рисунок Г.6 – Практична цінність

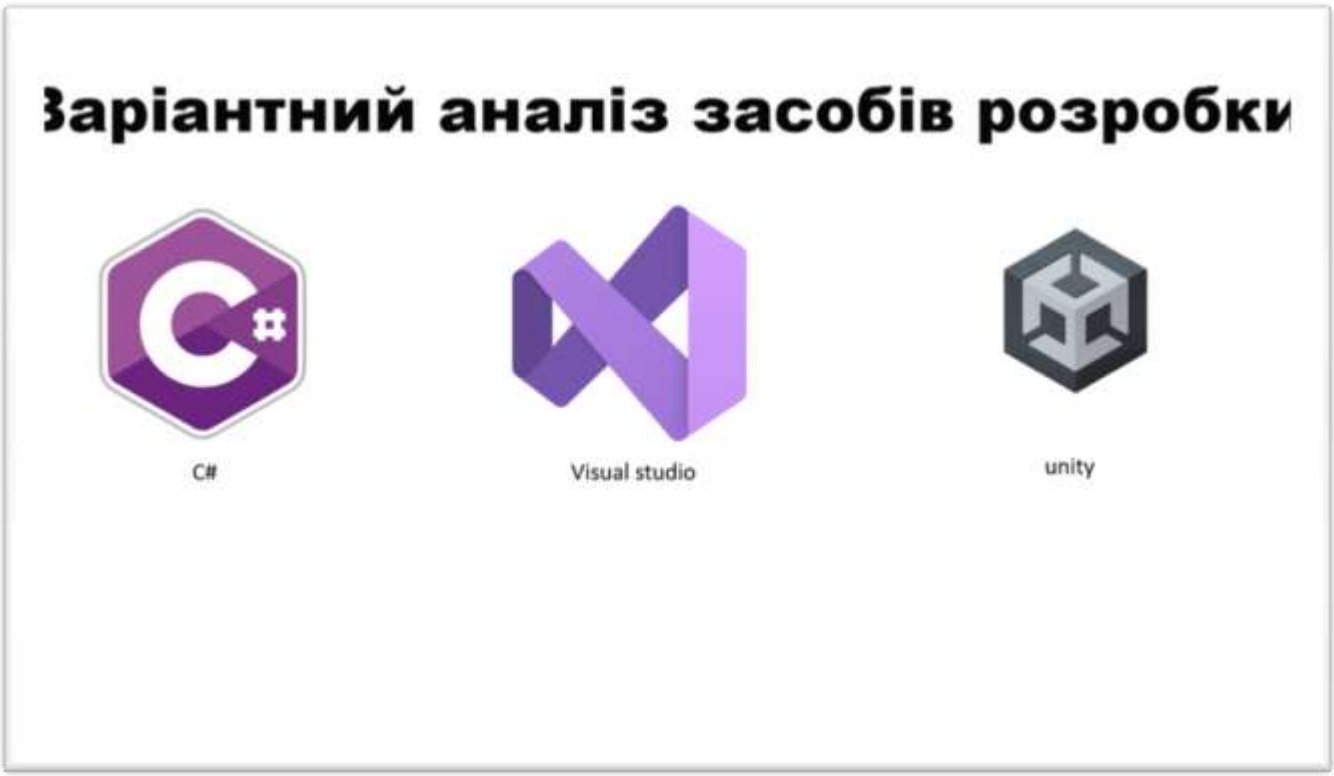


Рисунок Г.7 – Варіантний аналіз засобів розробки

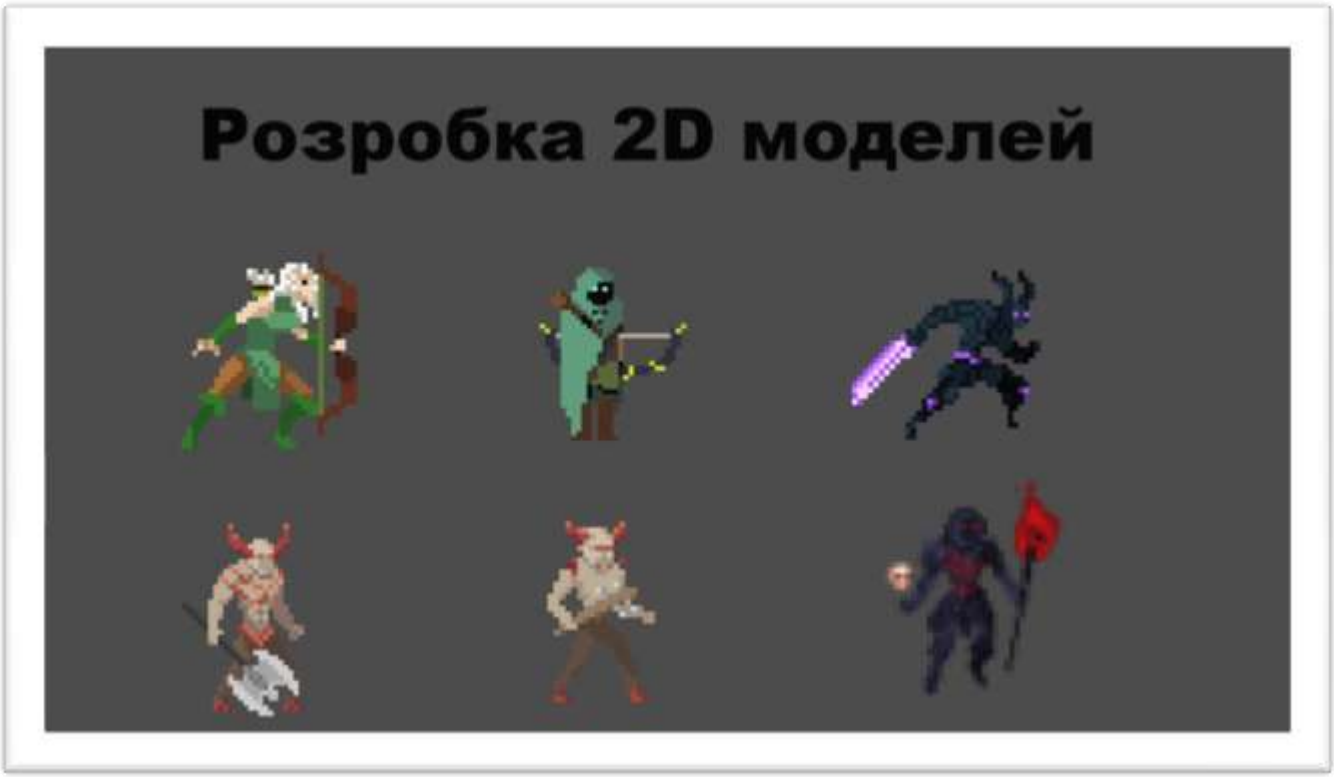


Рисунок Г.8 – Розробка 2D моделей

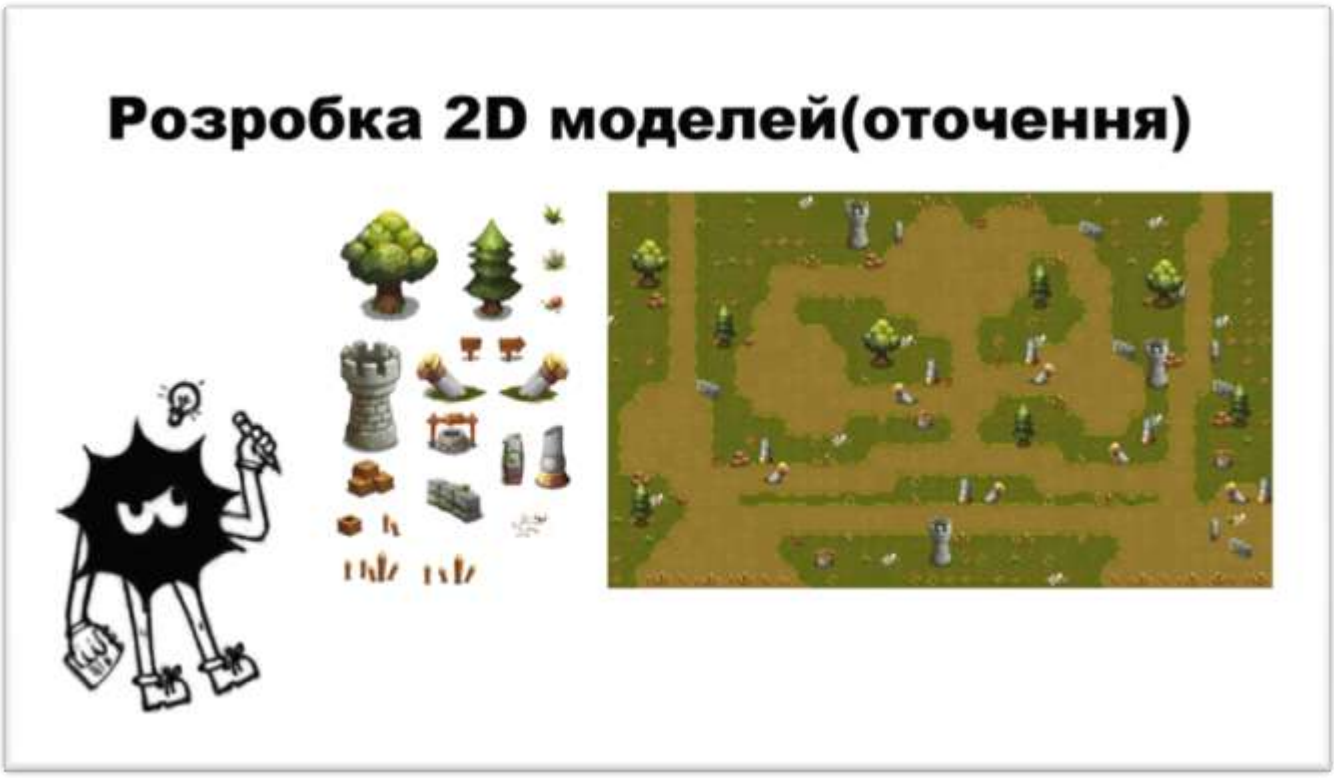


Рисунок Г.9 – Розробка 2D моделей(оточення)

Рисунок Г.10 – Головне меню

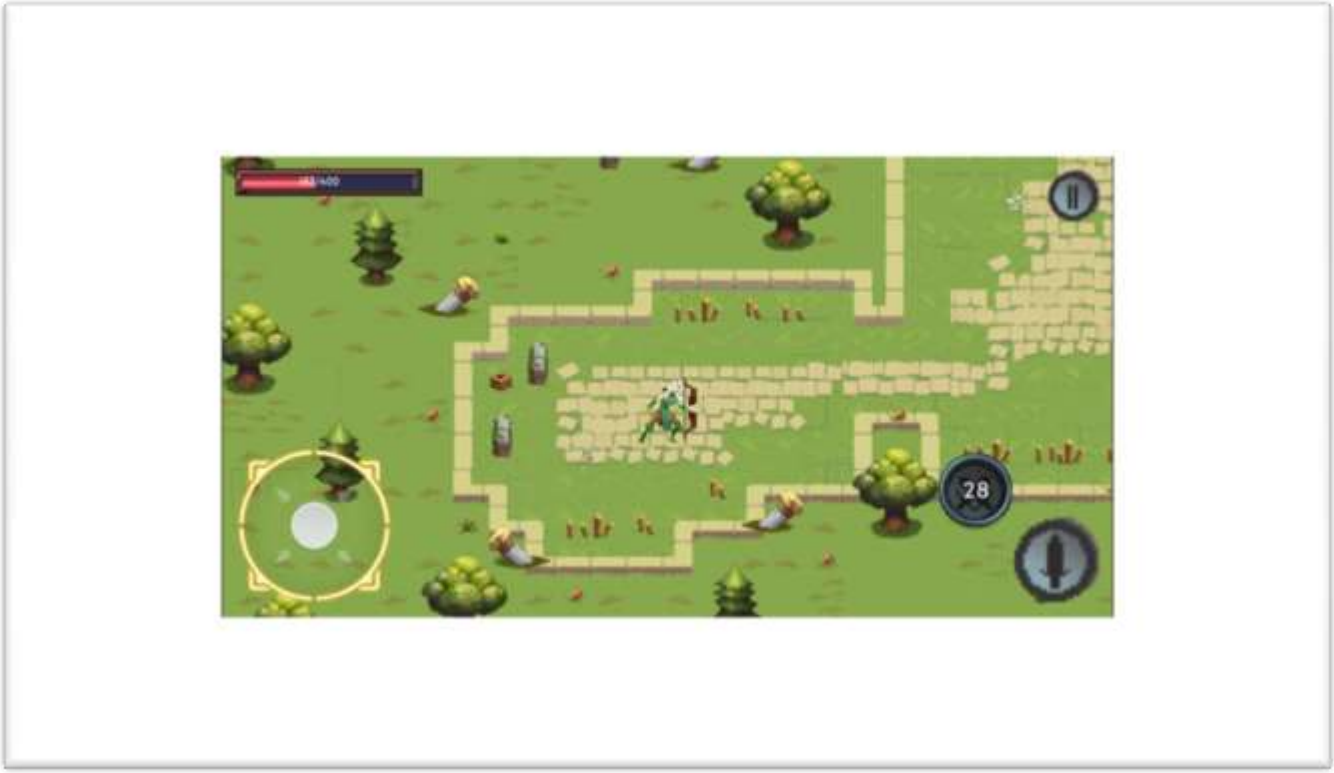


Рисунок Г.11 – Інтерфейс гравця

Рисунок Г.12 – Розробка моделі прецендентів взаємодії гравця та застосунку

Рисунок Г.13 – Розробка алгоритму поведінки противників

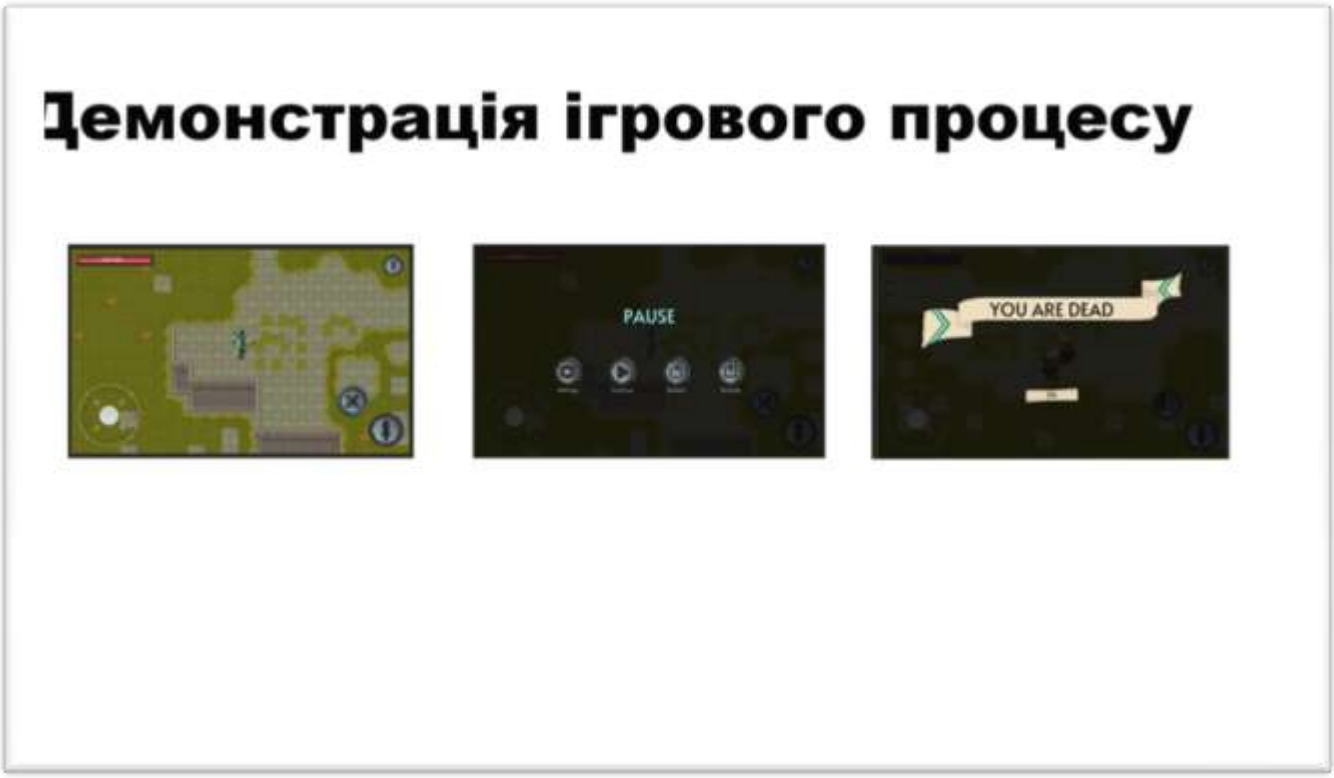


Рисунок Г.14 – Демонстрація ігрового процесу

Публікація

1. Кушнір М.М. Програмної платформа моделювання поведінки противників у платформених іграх. Електронні інформаційні ресурси: створення, використання, доступ. Збірник матеріалів Міжнародної науково-практичної Інтернет конференції 22-25 травня 2024 р. – Харків: НТУ «Харківський політехнічний інститут», 2024. – С. 1250.

Рисунок Г.15 – Публікація

Висновки

- Досліджено технології та методи для моделювання поведінки противників;
- Розроблено метод аналізу та обробки дій противників;
- Розроблено інтуїтивного і адаптивного графічного інтерфейсу, який буде зручним для користувачів різного віку і досвіду;
- Розроблено моделі та алгоритми роботи ігрової системи;
- Розроблено програмні модулі компонентів ігрової платформи;
- Проведено тестування ігрової системи.

Рисунок Г.16 – Висновки

Рисунок Г.17 – Дякую за увагу