МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ІНЖЕНЕРНИЙ НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ім. Ю.М. Потебні
ЗАПОРІЗЬКОГО НАЦІОНАЛЬНОГО УНІВЕРСИТЕТУ

КАФЕДРА ЕЛЕКТРОНІКИ, ІНФОРМАЦІЙНИХ СИСТЕМ ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Кваліфікаційна робота

Перший (бакалаврський)
(рівень вищої освіти)

на тему **Розробка ігрового застосунку з використанням системи Unity**

Виконав: студент 4 курсу, групи 6.1219-пзс
спеціальності 121 Інженерія програмного забезпечення
(код і назва спеціальності)
освітньої програми Програмне забезпечення систем
(код і назва освітньої програми)
Ю.О. Денисенко
(ініціали та прізвище)
Керівник доцент, к.ф.-м.н., доцент І.А.Скрипник
(посада, вчене звання, науковий ступінь, підпис, ініціали та прізвище)
Рецензент директор ТОВ «Дискус»
Р.О.Лютий
(посада, вчене звання, науковий ступінь, підпис, ініціали та прізвище)

Запоріжжя
2023

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ІНЖЕНЕРНИЙ НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ім. Ю.М. Потебні
ЗАПОРІЗЬКОГО НАЦІОНАЛЬНОГО УНІВЕРСИТЕТУ

Кафедра електроніки, інформаційних систем та програмного забезпечення

Рівень вищої освіти _____перший (бакалаврський)_____

Спеціальність ___121_Інженерія програмного забезпечення_____
(код та назва)

Освітня програма ___Програмне забезпечення систем
(код та назва)

ЗАТВЕРДЖУЮ

Завідувач кафедри_____Т.В. Критська_
"_01_" ___березня__ 2023_року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

_____Денисенку Юрію Олександровичу_____
(прізвище, ім'я, по батькові)

1.Тема роботи __Розробка ігрового застосунку з використанням системи Unity

керівник роботи Скрипник Ірина Анатоліївна, к. ф.-м. н., доцент,_____
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від 29.12.2022 р. № 1893-с

2. Строк подання студентом кваліфікаційної роботи__14.06.2023_____

3. Вихідні дані бакалаврської роботи

- комплект нормативних документів ;
- технічне завдання до роботи.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

- огляд та збір літератури стосовно теми кваліфікаційної роботи;
- огляд та аналіз існуючих рішень та аналогів;
- дослідження проблеми розробки ігор, особливості розробки в Unity;
- створення програмного продукту та його опис;
- дослідження поставленої проблеми та розробка висновків.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)

_____слайдів презентації

6. Консультанти розділів бакалаврської роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата Завдання прийняв

7. Дата видачі завдання 01.03.2023

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів кваліфікаційної роботи	Примітка
1	Аналіз предметної області	22.04.23	виконано
2	Формулювання основної задачі дипломної роботи та узгодження її з науковим керівником	25.04.23	виконано
3	Аналіз існуючих методів рішення	26.04-30.04.23	виконано
4	Аналіз методів розробки ігор на Unity	01.05-04.05.23	виконано
5	Аналіз засобів розробки ігор на Unity	05.05-08.05.23	виконано
6	Узгодження подальших дій з науковим керівником	10.05.23	виконано
7	Проектування гри	12.05.23	виконано
8	Програмна реалізація застосунку	14.05-22.05.23	виконано
9	Представлення отриманих результатів науковому керівнику та узгодження плану подальшого дослідження	25.05.23	виконано
10	Реалізація ШІ та процедурної генерації	26.05-03.06.23	виконано
11	Тестування застосунку та можливостей ШІ та процедурної генерації	05.06.23	виконано
12	Оформлення звіту	8.06-11.06.23	виконано
13	Оформлення презентації.	12.06-15.06.23	виконано

Студент _____ Ю.О.Денисенко
(підпис) (прізвище та ініціали)

Керівник роботи _____ І.А.Скрипник
(підпис) (прізвище та ініціали)

Нормоконтроль пройдено
Нормоконтролер _____ І.А. Скрипни

АНОТАЦІЯ

Сторінок – 64

Рисунків – 18

Джерел – 10

Денисенко Ю. О. Розробка ігрового застосунку з використанням систем и Unity: кваліфікаційна робота бакалавра спеціальності 121 «Інженерія програмного забезпечення» / наук. керівник Скрипник. І.А. Запоріжжя : ЗНУ. 2023. 64 с.

Дана дипломна робота присвячена розробці ігрового застосунку жанру Rogue-like на базі платформи Unity та мові програмування C#, з використанням процедурної генерації та засобів штучного інтелекту. В сучасному світі ігрова індустрія розвивається швидкими темпами, але все ще існує потреба у нових, цікавих ігрових досвідах.

Метою цієї роботи є розробка ігрового застосунку, який використовує процедурну генерацію контенту та засоби штучного інтелекту для створення конкурентноспроможного проекту та показу переваг обраних технологій у сучасному ринку. Завдання дослідження включають аналіз існуючих ігрових проектів, їх недоліків та переваг, а також саме дослідження методів та алгоритмів процедурної генерації, вибір та реалізацію відповідних алгоритмів та інструментів штучного інтелекту, а також розробку ігрового застосунку з використанням отриманих знань.

Дана кваліфікаційна робота докладно описує розробку зазначеного застосунку, та важливість обраного дослідження, що робить внесок у розвиток технологій процедурної генерації та ШІ у ігрових середовищах.

Ключові слова: *ігровий застосунок, процедурна генерація, штучний інтелект, варіативність, ігрове середовище*, *алгоритми, адаптація*.

ANNOTATION

Pages – 64

Pictures – 18

Sources – 10

Denysenko Y.O. Development of a game application using the Unity system: bachelor's thesis in specialty 121 "Software Engineering" / scientific adviser, Skrypnyk I.A. Zaporizhzhia: ZNU, 2023. 64p.

This thesis is devoted to the development of a Rogue-like game application based on the Unity platform and the C# programming language, using procedural generation and artificial intelligence. In today's world, the gaming industry is evolving rapidly, but there is still a need for new, interesting gaming experiences.

The purpose of this work is to develop a game application that uses procedural content generation and artificial intelligence tools to create a competitive project and show the advantages of selected technologies in the modern market. The objectives of the study include the analysis of existing game projects, their disadvantages and advantages, as well as the study of methods and algorithms for procedural generation, the selection and implementation of appropriate algorithms and artificial intelligence tools, as well as the development of a game application using the knowledge gained.

This qualification work describes in detail the development of this application, and the importance of the selected study, which contributes to the development of procedural generation technologies and AI in game environments.

Keywords: *game application, procedural generation, artificial intelligence, variability, game environment, algorithms, adaptation.*

ЗМІСТ

ВСТУП

Актуальність теми

Швидкий розвиток інформаційних технологій та постійний зростання конкуренції в ігровій індустрії стимулюють розробників ігрових застосунків до пошуку нових технологічних рішень для створення захоплюючих та інноваційних ігрових середовищ. У цьому контексті, розробка ігрового застосунку з використанням процедурної генерації та застосуванням засобів штучного інтелекту виявляється актуальною та перспективною темою.

Процедурна генерація в ігровій розробці дозволяє створювати геймплей, рівні, світи та ігрові об'єкти автоматично, за допомогою алгоритмів, замість традиційного ручного розміщення та проектування. Цей підхід відкриває безмежну кількість можливостей для геймдизайнерів, дозволяючи створювати непередбачувані та унікальні геймплейні ситуації, що забезпечує свіжість та цікавість гри з кожним запуском.

Застосування засобів штучного інтелекту в ігровій розробці забезпечує найкращий досвід у взаємодії з ігровим світом. Алгоритми прийняття рішень неігрових персонажів дозволяють їм самостійно приймати важливі рішення, забезпечуючи більш оптимізований досвід.

Окрім того, розробка ігрових застосунків з використанням процедурної генерації та штучного інтелекту сприяє оптимізації процесу розробки та забезпечує гнучкість у виробництві ігор. Відповідно до потреб гравців, гра може генерувати нові рівні, завдання та елементи геймплею, забезпечуючи безкінечний потік контенту без необхідності ручного створення та включення в оновлення.

Отже, розробка ігрового застосунку з використанням процедурної генерації та застосуванням засобів штучного інтелекту має великий потенціал для створення захоплюючих та інноваційних ігрових досвідів, задовольняючи зростаючі вимоги геймерів та забезпечуючи ефективність та гнучкість у процесі розробки.

Мета дослідження

Розробка ігрового застосунку, що використовує процедурну генерацію та штучний інтелект, для створення непередбачуваних та унікальних ігрових досвідів.

Завдання дослідження

Аналіз існуючих рішень для використання процедурної генерації та ШІ. Розглянути існуючі методи та технології до розробки ігор та вибрати відповідну для створення гри з використанням ШІ.

Об’єкт дослідження

Процес розробки ігрового застосунку та імплементування процедурної генерації та ШІ.

Предмет дослідження

Предметом дослідження є розгляд технологій процедурної генерації та штучного інтелекту неігрових персонажів, та їх застосування у обраному проекті.

Методи дослідження

Теоретичні – обробка літературних джерел, синтез дослідження та аналіз досліджуваного матеріалу.

Практичне значення одержаних результатів

Практичне значення даного дослідження полягає у розробці ігрових застосунків з використанням процедурної генерації та штучного інтелекту, що дозволить досягти унікального і неповторного досвіду, та реіграбельності.

Глосарій

Unity: Інтегроване середовище розробки (IDE), яке використовується для створення і редагування графічних ігор. Unity є потужним інструментом для розробки ігор на різних платформах.

NPC: Неігрові персонажі у ігровому середовищі. Це персонажі, які керуються комп'ютером і не керуються гравцем.

Фізика: моделювання реалістичного руху, взаємодії об'єктів та поведінки відповідно до законів фізики. Unity має вбудовану фізичну систему, яка дозволяє симулювати реалістичні фізичні ефекти, такі як гравітація, колізії, твердість та м'якість об'єктів, динамічний рух та інші фізичні властивості.

Колайдер: компонент, який додається до об'єкта у грі і використовується для визначення його фізичної області або форми. Колайдери використовуються для виявлення зіткнень між об'єктами та взаємодії з фізичною системою у грі.

Ієрархія: структури об'єктів у сцені або ієрархічної організації об'єктів. У Unity об'єкти можуть бути розміщені в ієрархічній структурі, де батьківські об'єкти мають дочірні об'єкти, а дочірні об'єкти можуть мати свої власні дочірні об'єкти. Така ієрархічна організація дозволяє керувати і маніпулювати групами об'єктів як один цілий об'єкт.

Процедурна генерація: метод створення вмісту або дизайну, який використовується в іграх і комп'ютерних програмах. Вона включає в себе використання алгоритмів та правил для створення випадкових, але аутентичних об'єктів, сценаріїв, світів або інших елементів гри. У контексті розробки ігрових застосунків в середовищі Unity, процедурна генерація використовується для створення ігрових рівнів, ландшафтів, текстур, міст, об'єктів, музики та багато іншого.

Компонент: основна одиниця функціональності, яка призначена для прикріплення до об'єктів у сцені і задає їх поведінку, властивості та

можливості. Компоненти дозволяють розробникам додавати різні функції до об'єктів і керувати їхньою поведінкою, взаємодією з іншими об'єктами та зовнішніми подіями.

Подія: основний механізм сповіщення і взаємодії між різними об'єктами та компонентами гри. Події використовуються для передачі інформації про виникнення певних подій або станів, що можуть спричинити реакцію інших об'єктів або компонентів гри.

Об'єкт: Елемент ігрової ієрархії, який може мати свою графіку, поведінку та інші характеристики. Може бути представлений у виді гравця, ворогів, об'єктів оточення, інтерфейсу, генераторів подій.

Скрипт: Файл програмного коду, який містить інструкції для керування поведінкою об'єктів в грі. У Unity використовується мова програмування C# для написання скриптів.

Тайл: основний елемент фізичного оточення, плитка, яка є частиною загальної структури рівня (підлоги, стін, т.п.).

Анімація: процес створення рухомих образів (або змін стану) для об'єктів гри, щоб надати їм візуальний рух та живість.

Штучний інтелект (ШІ): Галузь комп'ютерних наук, яка займається розробкою програм та алгоритмів, що дозволяють комп'ютеру виконувати завдання, що зазвичай потребують інтелекту людини.

Сцена: Простір, в якому розгортається гра. Сцена включає в себе всі елементи ігрової ієрархії, які утворюють візуальне середовище гри.

Спрайт: графічний об'єкт, який представляє собою 2D-зображення або анімацію, яку можна використовувати для візуалізації об'єктів, персонажів, тайлів та інших елементів у 2D-середовищі гри.

Rogue-like: піджанр рольових відеоігор, визначними особливостями якого є випадкове, процедурне створення рівнів, покроковий ігровий процес, плиткова (тайлова) або ASCII-графіка та перманентна смерть персонажа у випадку поразки. Назва «rogue-подібні ігри» походить від гри Rogue, котра з'явилася у 1980 році й вважається початком піджанру.

Детектор: частина логіки гри, що відповідає за виявлення певних подій або станів у грі. Детектор може використовуватися для виявлення зіткнень, взаємодії з об'єктами, виконання певних умов або подій.

Вектор: структура даних, яка представляє собою числову величину зі специфічним напрямом та довжиною. У Unity вектори використовуються для визначення положення, напрямку, швидкості, сили та багатьох інших фізичних та математичних величин у 2D або 3D просторі.

Палітра тайлів: інструменту, який дозволяє розробникам створювати та керувати набором тайлів для використання у рівнях або локаціях гри. Палітра тайлів надає зручний інтерфейс для створення, організації та редагування тайлів. Розробники можуть імпортувати або створювати власні графічні тайли, такі як текстури, зображення або спрайти, і додавати їх до палітри.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Огляд літературних джерел

Пропонована література досліджує різні аспекти штучного інтелекту (ШІ) і процедурної генерації в контексті розробки ігор на Unity. Вона пропонує практичні приклади, які допоможуть розробникам зрозуміти концепції і застосувати їх у своїх проектах. За допомогою цих ресурсів, розробники зможуть створити гру, де інтелектуальні персонажі здатні бути сильними супротивниками, а світ може бути кожного разу унікальним та цікавим [1].

Крім того, в літературі розглядаються різні алгоритми та методи, які допоможуть досягти більшої реалістичності інтелектуальних персонажів у грі [6]. Вона пропонує широкий спектр завдань і рішень, пов'язаних з розробкою інтелектуальних персонажів, та дає можливість ознайомитись з різноманітними техніками та підходами до розробки ШІ в середовищі Unity.

Розробка ігор на Unity, як правило, зустрічається з деякими викликами. По-перше, це складність процесу розробки. Особливо для одиночних розробників або невеликих команд, створення ефективного ігрового застосунку вимагає значних зусиль, часу, знань мови програмування C# та технічної експертизи.

Другий виклик пов'язаний з вимогами до продуктивності. Для забезпечення плавного геймплею та уникнення затримок, ігровий застосунок повинен бути оптимізованим. Це може вимагати значних інвестицій у апаратне забезпечення та програмне забезпечення, а також знань щодо оптимізації коду [4].

Останнім викликом є нестабільність ігрового ринку. Розробники ігор повинні підтримувати зацікавленість гравців у своїх продуктах, враховуючи зміни і потреби ринку, який є непередбачуваним.

Нарешті, література розглядає різні способи імплементації штучного інтелекту у ігровий застосунок. Сьогодні існує багато засобів для створення

інтелектуальних супротивників, таких як машинне навчання або скінченні автомати з заданими скриптами на мові C#, які надають персонажам здатність змінювати свій стан та реагувати на зміни у середовищі [2].

Узагалі, розробка ігрового застосунку з використанням Unity, що поєднує процедурну генерацію і штучний інтелект, вимагає певних знань і технічної підготовки. Але вона також відкриває широкі можливості для створення захоплюючих та інноваційних ігор, де інтелектуальні персонажі здатні до складних поведінкових рішень та генерації унікальних ігрових просторів.

1.2 Дослідження розробки ігор з процедурною генерацією та ШІ

Rogue-like ігри є популярним жанром серед геймерів, завдяки своїй високій реіграбельності та процедурно згенерованим рівням. Розробники постійно прагнуть створити захоплюючий геймплей і різноманітність у кожній грі, а використання штучного інтелекту (ШІ) та процедурної генерації дозволяє досягти цих цілей.

Штучний інтелект у rogue-like іграх може бути використаний для створення реалістичних та вишуканих ворогів або монстрів, з якими гравці будуть змагатися. Це створює виклик та необхідність розвивати вміння та тактику у гравців.

Одним з ключових аспектів rogue-like ігор є процедурна генерація рівнів. Використання процедурної генерації дозволяє створювати унікальні та непередбачувані рівні, що розширює геймплей та забезпечує безкінечну кількість варіантів для гравців. ШІ може бути задіяний у процесі генерації рівнів, допомагаючи створити балансовану та цікаву ігральну площину, адаптуючи рівень складності до навичок гравця, а також оптимізувати поведінку ворогів у випадковому середовищі.

Unity є однією з платформ, що активно використовуються для розробки rogue-like ігор з процедурною генерацією та ШІ. Unity надає розробникам

широкий спектр інструментів, бібліотек та плагінів для реалізації цих концепцій. У цьому розділі ми розглянемо деякі з них.

Алгоритми шуму: такі як Perlin Noise і Simplex Noise (Див. рис. 1), є потужними інструментами для генерації реалістичних випадкових форм і текстур. Вони широко використовуються в комп'ютерній графіці, комп'ютерних іграх і комп'ютерній графіці для створення різноманітних ефектів [3].

Perlin Noise і Simplex Noise - це алгоритми, які генерують псевдовипадкові значення на основі градієнтних шаблонів. Вони створюють плавні переходи між значеннями, що дозволяє отримати керовані форми і текстури з випадковим виглядом.

Ці алгоритми можуть бути використані для створення різних ефектів і об'єктів. Наприклад, вони можуть бути використані для генерації ландшафтів і теренів з реалістичними гірськими формами і деталізацією. Шум може бути застосований до висотної карти, де висота кожної точки визначається значенням шуму в цій точці.

Також, алгоритми шуму можуть бути використані для генерації реалістичних печер або підземних систем, де форма кристалів або проходів визначається шумом.

Крім того, алгоритми шуму можуть бути використані для створення деталей на поверхні об'єктів, таких як текстури або рельєф. Вони можуть додати натуралізм і випадковість до вигляду об'єктів, зробити їх більш реалістичними і природними.

Загалом, використання алгоритмів шуму відкриває безліч можливостей для створення випадкових, але керованих форм і текстур, що знаходять широке застосування в графічних іграх, візуалізації, комп'ютерній графіці та інших областях.

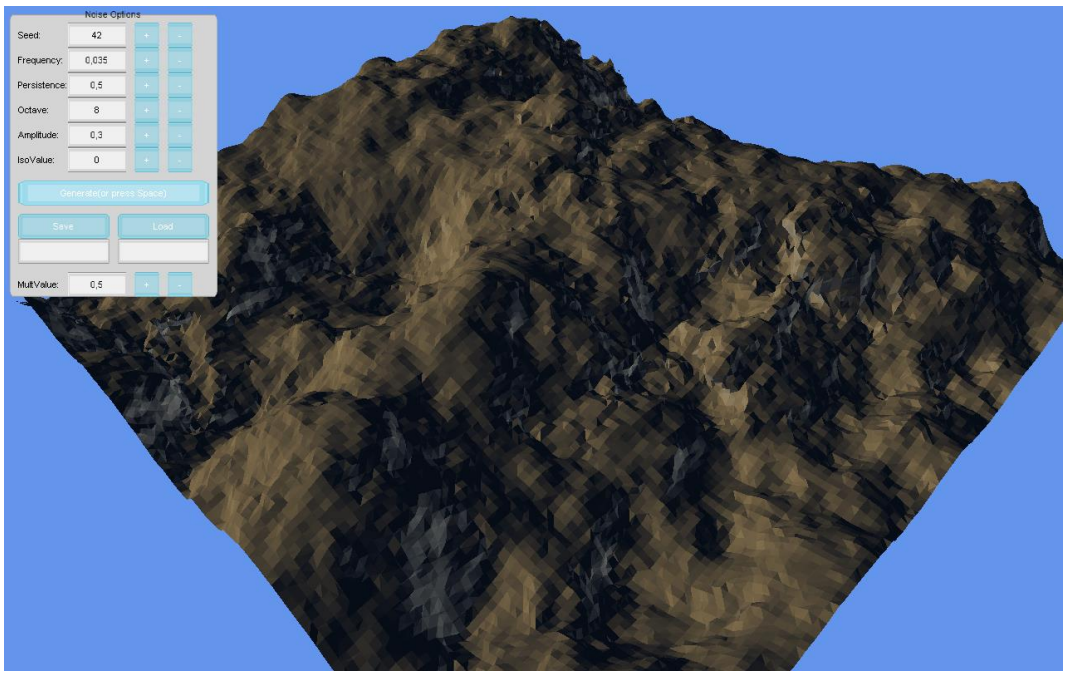

Рис.1 Simplex Noise usage

Клітинний автомат (Див. рис. 2)*:* є потужним інструментом для моделювання систем, де клітини взаємодіють за заданими правилами. В цьому підході простір поділяється на клітини, кожна з яких може перебувати в певному стані. Потім відбуваються ітерації, під час яких стани клітин оновлюються згідно з заданими правилами взаємодії.

Клітинний автомат знаходить широке застосування в багатьох галузях, зокрема в комп'ютерній графіці, моделюванні фізичних систем, біології та соціальних науках.

Один з прикладів використання клітинних автоматів - створення лабіринтів. Кожна клітина може мати стан "прохід" або "стіна", і за допомогою правил взаємодії можна генерувати лабіринти з різними формами та складностями. Такий підхід дозволяє створювати цікаві та випадкові лабіринти для використання у відеоіграх, комп'ютерних симуляціях або графічних дизайнах.

Крім того, клітинні автомати можуть бути використані для генерації різноманітних структур, таких як мікроорганізми, кристали, дерева тощо.

Правила взаємодії можуть моделювати процеси росту, розвитку та взаємодії між клітинами, що веде до створення різноманітних форм і структур.

Клітинні автомати також можуть бути використані для генерації об'єктів у випадковому порядку. Задавши правила взаємодії, можна створити систему, в якій об'єкти з'являються, розміщуються та зникають у випадковому порядку, що дає випадкові та цікаві результати.

Клітинні автомати є гнучким інструментом для моделювання складних систем і створення випадкових, але контрольованих структур та об'єктів. Вони дозволяють відтворити різноманітні природні процеси та створити унікальні графічні ефекти, що знайшли широке застосування в різних галузях.

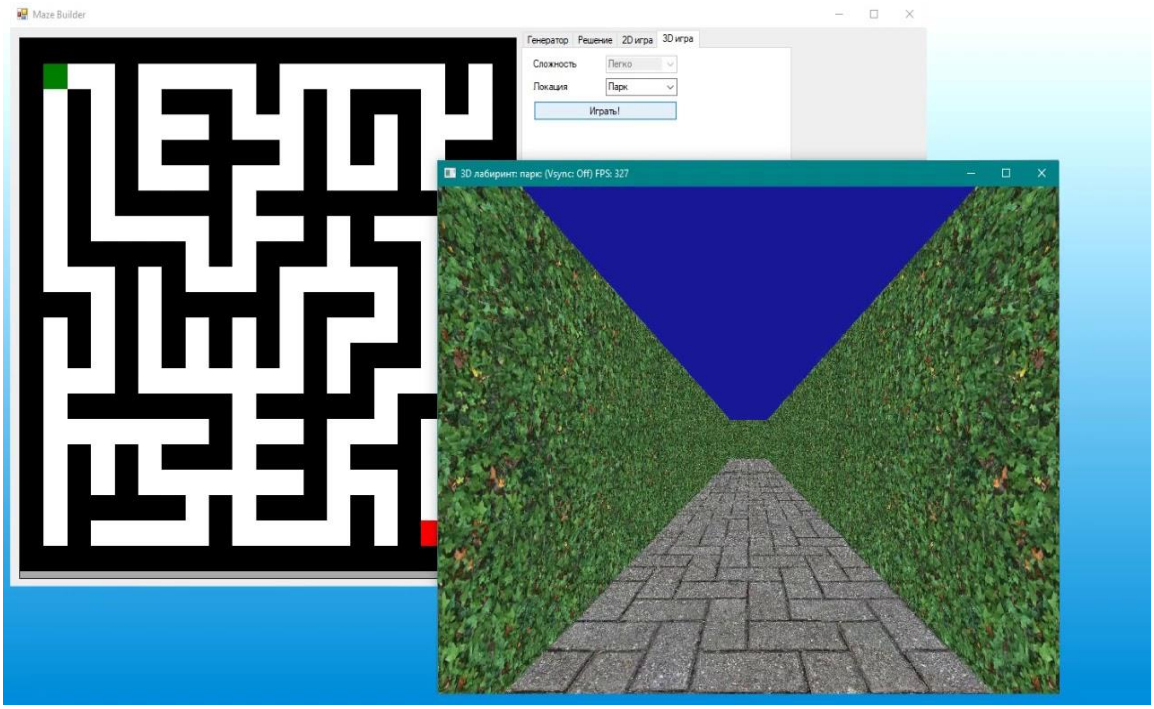

Рис. *2 Використання клітинного автомата у генерації лабіринтів*

Генерація об'єктів включає процес автоматичного розташування об'єктів на рівні гри з метою створення різноманітності та балансу. Цей підхід дозволяє створювати унікальні та цікаві ігрові ситуації, забезпечувати різноманітність геймплею та впливати на взаємодію гравця з оточенням.

Один із алгоритмів, що використовується для генерації об'єктів, - це розподіл Пуассона. Цей алгоритм базується на випадковому розташуванні об'єктів з таким розподілом, що забезпечує мінімальну взаємодію між ними. Він гарантує, що об'єкти будуть розташовані належним чином та не перекриватимуть один одного, створюючи рівномірну розташованість.

Ще один алгоритм - це зони впливу. Він використовується для створення областей, де об'єкти мають вплив на своє оточення. Наприклад, можуть бути визначені зони, в яких об'єкти мають певний радіус впливу, і їх розташування визначається таким чином, щоб забезпечити баланс між цими зонами. Це може бути корисно для створення різноманітних типів об'єктів з різними характеристиками або ефектами.

Крім цього, існує багато інших алгоритмів і підходів для генерації об'єктів на рівні гри, таких як випадкове розташування, генетичні алгоритми, фрактали та інші. Вибір конкретного алгоритму залежить від конкретних потреб і вимог вашої гри чи програми.

Вище були зазначені деякі приклади алгоритмів для реалізації процедурної генерації, далі будуть зазначені вивчені технології імплементації ШІ у ігрових середовищах.

Кінцеві автомати: Кінцеві автомати є потужним інструментом для моделювання поведінки об'єктів в ігровому середовищі. Вони дозволяють описати різні стани, у яких можуть перебувати об'єкти, і визначити переходи між цими станами на основі певних умов [2].

У Unity, компонент Animator надає можливість створювати прості або складні кінцеві автомати для керування рухами, анімаціями та взаємодією персонажів з оточенням. Ви можете використовувати візуальний інтерфейс для створення станів, переходів, анімаційних параметрів та умов, що визначають переходи між станами.

Кожен стан може мати власні анімації, звуки, логіку та дії, що відповідають певному поведінковому сценарію. Переходи між станами

можуть бути викликані різними подіями або умовами, наприклад, коли гравець натискає певну кнопку або досягає певного місця на карті.

Створення складних кінцевих автоматів дозволяє моделювати різноманітну поведінку персонажів, таку як ходьба, біг, атака, взаємодія з об'єктами тощо. Ви можете програмувати логіку переходів і дій в реальному часі, забезпечуючи багатошаровий та реалістичний геймплей.

Компонент Animator в Unity надає розширені можливості для керування анімаціями та поведінкою персонажів. Ви можете використовувати скрипти C# для програмування перехоплення подій, зміни параметрів кінцевого автомата та виконання спеціальних дій під час роботи персонажа.

Машини підтримки прийняття рішень: Машини підтримки прийняття рішень (МППР) є потужними і складними моделями штучного інтелекту, які дозволяють агентам приймати рішення на основі аналізу великої кількості факторів і вхідних даних. Вони можуть використовуватися в різних сферах, включаючи ігрову індустрію, фінанси, медицину та багато інших.

Для реалізації МППР у ваших проектах ви можете скористатися пакетами, такими як Unity ML-Agents (Див. рис. 3). Unity ML-Agents надає потужні засоби для тренування агентів на основі нейронних мереж. Ви можете визначити структуру агента, включаючи вхідні дані, вихідні дії і нейронну мережу, а потім використовувати алгоритми навчання, такі як зворотне поширення помилки або алгоритми зміцненого навчання, для тренування агента на основі вхідних даних та бажаного результату.

За допомогою Unity ML-Agents ви можете створювати навчальні середовища, в яких агент взаємодіє зі своєю оточуючим середовищем і навчається приймати оптимальні рішення. Агент може навчатися на основі нагород і покарань, що його отримує за свої дії, і з часом самостійно розвивати оптимальні стратегії прийняття рішень.

Unity ML-Agents також надає засоби для візуалізації та аналізу навчальних процесів, що дозволяє вам отримувати інформацію про виконання агента, збирати статистику та налагоджувати його поведінку.

Рис. *3 ML Agents Unity logo*

Дерева поведінки: Дерева поведінки є потужним і гнучким підходом до моделювання поведінки об'єктів у іграх. Вони дозволяють структурувати поведінку в ієрархічну форму, де кожен вузол представляє конкретну дію або умову. Дерева поведінки дозволяють легко організовувати складні логічні структури, визначати пріоритети дій та забезпечувати гнучке управління поведінкою об'єктів.

У середовищі Unity ви можете використовувати фреймворк Behavior Designer для розробки дерев поведінки. Behavior Designer надає інтуїтивно зрозумілий інтерфейс для створення, візуалізації і редагування дерев поведінки без необхідності писати код.

За допомогою Behavior Designer ви можете визначити різні типи вузлів, такі як вузли вибору, послідовності, умови, декоратори та інші. Ви можете створювати власні вузли або використовувати готові компоненти для взаємодії з об'єктами, виконання дій та прийняття рішень.

Behavior Designer також підтримує візуальне налагодження дерев поведінки, що дозволяє вам відстежувати виконання дерева та виявляти можливі проблеми або помилки.

Завдяки фреймворку Behavior Designer ви можете легко створювати складні поведінки для своїх об'єктів у Unity, забезпечуючи їм інтелектуальну поведінку та реалістичну взаємодію з оточенням.

1.3 Аналіз існуючих ігрових застосунків використовуючих платформу Unity

Існує чимало rogue-like ігор, які були створені на базі платформи Unity. Деякі з них також створені з використанням технологій процедурної генерації та засобів штучного інтелекту.

Перша розглянута гра, — "Subnautica" (Див. рис. 4) [7]. У цій грі гравець розташований на плавучому острові на планеті з великим океаном, який досліджує і знаходиться в боротьбі за виживання. Гра створена на платформі Unity, в ній застосовані базові алгоритми ШІ для неігрових персонажів, та засоби процедурної генерації. Розглянемо плюси та мінуси цієї гри в індустрії ігрової розробки на Unity:

Рис. *4 Subnautica*

Переваги:

Ця гра має доволі потужні алгоритми процедурної генерації, що дозволяє гравцеві кожну гру мати новий та унікальний досвід, вражаюча графіка, та великий відкритий світ для дослідження.

Недоліки:

Доволі неповноцінні системи ШІ для неігрових персонажів, через що часто можуть ставатися поведінкові помилки.

Погана оптимізація: ця проблема дуже часта для розробки ігор на Unity, і ця гра як раз одна з них.

Рис. *5 No Man’s sky*

Інший приклад - "No Man's Sky" (Див. рис. 5) [8], є космічною процедурно згенерованою грою, розробленою компанією Hello Games. Гравці отримують можливість досліджувати безмежний відкритий світ, який складається з мільярдів процедурно згенерованих планет. Гра надає вам можливість зіграти за володаря космічного корабля і виконувати різноманітні завдання, такі як дослідження нових планет, виявлення ресурсів, торгівля та боротьба з ворогами.

ШІ: "No Man's Sky" використовує штучний інтелект для створення унікальних форм життя, тварин і рослин на кожній планеті. ШІ також відповідає за поведінку ворожих і сутностей в грі, що робить світ гри живим і реалістичним.

Переваги:

Безмежний відкритий світ з процедурно згенерованими планетами робить гру дуже різноманітною та непередбачуваною. Кожна планета унікальна, з власним ландшафтом, кліматом, рослинністю та тваринами.

Унікальна графіка і атмосфера, що вражають своєю красою та деталізацією.

ШІ створює різноманітність і життєвий світ на кожній планеті, забезпечуючи гравцеві неперервний досвід дослідження та взаємодії зі світом гри.

Недоліки:

У деяких випадках можуть виникати проблеми з оптимізацією та пов'язані зі швидкістю завантаження, особливо на менш потужних системах.

Ціна: ця гра пропонує дуже незадовільні цінові політики для гравців, що є великим недоліком для більшості з користувачів.

1.4 Особливості розробки ігрових застосунків з використанням процедурної генерації та штучного інтелекту

Розробка ігрових застосунків з використанням процедурної генерації та штучного інтелекту є складним процесом, який вимагає уваги до деталей та особливих підходів. В цьому контексті, існує кілька особливостей, які слід враховувати.

Процедурна генерація контенту вимагає розробки алгоритмів та систем, що забезпечують створення різноманітних об'єктів, локацій або рівнів. Це дозволяє створити унікальний та непередбачуваний геймплей для кожного запуску гри.

Штучний інтелект в іграх вимагає розробки алгоритмів та моделей, що дозволяють ворогам або неприятелям у грі виявляти розумне та адекватне поводження. Це може включати прийняття рішень на основі отриманих даних, планування поведінки, реалізацію алгоритмів пошуку або навіть машинне навчання.

Балансування геймплею є важливим аспектом розробки. Застосування процедурної генерації та штучного інтелекту може впливати на складність гри та її викликів для гравця. Розробники повинні тестувати та налаштовувати

рівень складності, а також контролювати процес генерації контенту, щоб забезпечити належний баланс, який забезпечить задоволення гравця.

Отже, розробка ігрових застосунків з використанням процедурної генерації та штучного інтелекту є захоплюючим завданням, яке вимагає специфічного підходу та ретельного планування. Використання процедурної генерації дозволяє створювати унікальний та неповторний контент, забезпечуючи постійну варіативність геймплею. Водночас, впровадження штучного інтелекту дозволяє створювати реалістичних та інтелектуальних ворогів, з якими гравець може взаємодіяти.

1.5 Постановка завдання

Розробка ігрових застосунків з використанням процедурної генерації та штучного інтелекту є актуальною та цікавою темою в галузі ігрової розробки. Основним завданням проекту є створення 2D Rogue-like гри з процедурною генерацією рівнів та використанням штучного інтелекту для ворожих персонажів.

У процесі розробки гри необхідно зосередитись на створенні захоплюючого геймплею, що включатиме у себе реалістичні бої з ворогами та інтуїтивне керування персонажем у 2D середовищі. Гравець повинен мати можливість виконувати різні атаки та уникати ворожих атак, а також збирати ресурси та предмети для поліпшення свого персонажа.

Одним з ключових аспектів проекту є процедурна генерація рівнів. Необхідно розробити алгоритми, що забезпечать генерацію випадкових рівнів з різноманітними елементами, такими як перешкоди, предмети, вороги та секретні кімнати. Важливо досягти балансу складності рівнів, щоб гра булаз викликом, але не занадто складною або простою. Рівні повинні бути достатньо різноманітними, щоб забезпечити повторну грати, уникнути одноманітності та збільшити цікавість гри.

Другим важливим аспектом є впровадження штучного інтелекту для ворожих персонажів. Вороги повинні мати адекватну поведінку, включаючи рух, атаку, уникання та стратегічні рішення. Алгоритми штучного інтелекту повинні бути оптимізовані для ефективної роботи в реальному часі та забезпечувати відповідну реакцію ворогів на дії гравця.

Після реалізації проекту необхідно провести тестування гри, виявити та виправити можливі помилки та недоліки.

Загальним висновком проекту має бути успішна розробка 2D Rogue-like гри з процедурною генерацією рівнів та використанням штучного інтелекту для ворожих персонажів. Результатом повинна бути захоплююча та складна гра, яка надихне гравців продовжувати грати та насолоджуватись геймплеєм.

Головними факторами унікальності даного проекту є націленість на максимальну оптимізацію під всі системи, та відкриті можливості для поточних оновлень та модифікації під конкурентоспроможного продукту на ринці у майбутньому.

2 ДОСЛІДЖЕННЯ ПРОГРАМНИХ ЗАСОБІВ ДЛЯ РЕАЛІЗАЦІЇ ІГРОВИХ ЗАСТОСУНКІВ

2.1 Огляд технологій створення ігрових застосунків з процедурною генерацією

Огляд технологій створення ігрових застосунків з процедурною генерацією дозволяє розробникам створювати унікальні та незалежні від авторського втручання ігральні простори, персонажів та ігрові елементи. Використання процедурної генерації дозволяє створити безкінечні світи, уникнути монотонності ігрового досвіду, а також забезпечити велику реіграбельність.

Однією з популярних технологій для розробки ігрових застосунків з процедурною генерацією є Unity. Unity надає розробникам потужний інструментарій та підтримку для реалізації процедурної генерації в іграх. Вона має різноманітні функції, такі як генерація ландшафту, розміщення об'єктів, створення місій та завдань, генерація противників та багато іншого.

Іншою популярною технологією є Unreal Engine. Unreal Engine також надає широкі можливості для реалізації процедурної генерації. Вона має потужний графічний редактор, систему матеріалів, фізичний двигун та різноманітні інструменти для створення унікальних світів та ігрових елементів.

Ще однією популярною технологією є Godot Engine. Godot Engine є відкритим інструментарієм для розробки ігор, який також підтримує процедурну генерацію. Вона надає широкий набір готових модулів та інструментів для створення процедурних ефектів, ландшафтів, рівнів та інших ігрових елементів.

Крім цього, існують спеціалізовані мови програмування та бібліотеки для реалізації процедурної генерації, такі як Python з бібліотеками які Pygame та Panda3D, або JavaScript з бібліотеками Three.js та Phaser. Вони надають

більш гнучкі можливості для програмістів і дозволяють реалізовувати свої власні алгоритми процедурної генерації.

Окрім того, ігрові двигуни, такі як Unity та Unreal Engine, надають розширені можливості для процедурної генерації. Вони мають вбудовані інструменти та бібліотеки, які допомагають розробникам створювати процедурно згенеровані об'єкти, ландшафти, рівні гри та інше. Ці двигуни надають програмістам доступ до API для створення власних алгоритмів генерації та інтеграції їх у свої проекти.

Крім того, існують спеціалізовані бібліотеки, такі як LibNoise, FastNoise і Noiselib, які дозволяють генерувати різні типи шуму і різноманітних форм, що можуть бути використані для процедурної генерації ландшафтів, текстур і об'єктів у грі.

Продовжуючи огляд технологій створення ігрових застосунків з процедурною генерацією, варто згадати про такі інструменти, як Houdini та Substance Designer.

Houdini - це потужний візуальний редактор для процедурного моделювання, анімації та візуалізації. Він широко використовується в ігровій індустрії для створення процедурно згенерованих об'єктів, ландшафтів та спеціальних ефектів. Houdini надає багато інструментів та алгоритмів, які можна використовувати для створення різноманітних ігрових елементів.

Substance Designer - це інструмент для створення процедурних текстур і матеріалів. Він дозволяє розробникам створювати складні, реалістичні текстури, які можуть бути застосовані до об'єктів у грі. Substance Designer використовує графічний інтерфейс для створення текстур, дозволяючи користувачам контролювати різні параметри і ефекти.

2.2 Види штучного інтелекту в ігрових середовищах

Штучний інтелект (ШІ) в ігрових середовищах включає різні види алгоритмів та технологій, які забезпечують поведінку комп'ютерних

персонажів (NPC), опонентів або союзників у грі. Ось кілька видів штучного інтелекту, які часто використовуються в ігровій індустрії [2]:

Розподілений штучний інтелект (Distributed Artificial Intelligence):

Опис: Розподілений ШІ використовується, коли ігральний світ потребує великої кількості автономних персонажів, які взаємодіють один з одним із зовнішнім середовищем. Застосування розподіленого ШІ дозволяє створити більш реалістичну та динамічну соціальну та колективну поведінку неігрових персонажів. Замість того, щоб керувати кожним персонажем окремо, розподілений ШІ дає можливість неігровим персонажам взаємодіяти, спілкуватися, співпрацювати або конкурувати один з одним на основі встановлених правил та цілей.

Це досягається шляхом розподілення обчислень та прийняття рішень між різними агентами в ігральному середовищі. Агенти можуть використовувати методи розподіленого навчання, комунікації та координації для досягнення спільних цілей або виконання певних завдань. Вони можуть обмінюватися інформацією, передавати команди, співпрацювати в групах або взаємодіяти з гравцем.

Приклад: MMORPG (масові онлайн-рольові ігри) використовують розподілений ШІ для моделювання поведінки тисяч гравців у великому віртуальному світі.

Машинне навчання (Machine Learning):

Опис: Машинне навчання використовується для навчання алгоритмів на основі даних, щоб вони можливо наближалися до розумової поведінки. ШІ може навчатися на основі гравців або експертних даних, щоб покращити свої рішення та стратегії.

Основними компонентами процесу машинного навчання є вхідні дані (навчальний набір даних), обрана модель, алгоритми навчання, процес навчання (підгонка моделі до даних) та процес оцінки та тестування моделі. У результаті успішного навчання модель може здати прийнятні рішення або

зробити передбачення на основі нових даних, які не були використані під час навчання.

Приклад: Гра "AlphaGo" розроблена компанією DeepMind використовувала машинне навчання, для того щоб навчитися перемагати професійних гравців у го.

Скінчені автомати (Finite State Machines):

Опис: Скінчені автомати використовуються для моделювання поведінки ШІ на основі конкретних станів і переходів між ними. Кожен стан представляє певну дію або поведінку, яка залежить від поточного стану гри або дій гравця.

Скінчений автомат складається зі скінченого набору станів, переходів між станами і дій, пов'язаних з кожним станом. Кожен стан може мати певні умови, які визначають, коли ШІ переходить до іншого стану. Умови можуть бути пов'язані зі станом гри, такими як наявність певного об'єкта, відстань до гравця, рівень здоров'я тощо, або можуть залежати від дій гравця, таких як атака, взаємодія або переміщення.

Коли виконується певна умова, ШІ переходить до нового стану і виконує пов'язані з ним дії або поведінку. Цикл переходу від одного стану до іншого продовжується залежно від встановлених умов і дій ШІ.

Приклад: В багатьох шутерах від першої особи, неігрові персонажі можуть мати скінчений автомат, що дозволяє їм перехоплювати, атакувати або ухилятися від ворожих дій, залежно від ситуації.

Генетичні алгоритми (Genetic Algorithms):

Опис: Генетичні алгоритми моделюють еволюцію і прогресують шляхом відбору і мутації найкращих рішень. Вони використовуються для оптимізації параметрів або стратегій ШІ шляхом здійснення "еволюції" через покоління.

Процес еволюції включає в себе два основних кроки: відбір і мутацію. На кожній ітерації алгоритму вибирається певна кількість найкращих рішень (на основі визначеної функції пристосованості), які будуть продовжувати еволюцію у наступне покоління. Вибір може бути здійснений, наприклад, на

основі принципу "найкращих з кращих" (елітарний відбір) або з використанням інших стратегій, таких як рулетковий відбір або турнірний відбір. Після відбору проводиться мутація, яка випадковим чином змінює деякі генетичні властивості рішень. Це дозволяє ввести різноманітність і новину в популяцію, що може призвести до знаходження ще кращих рішень або стратегій.

Таким чином, генетичні алгоритми працюють за принципом поступового поліпшення популяції рішень з кожним поколінням. Повторюючи відбір та мутацію протягом багатьох ітерацій, алгоритм може зближатися до оптимального рішення або стратегії для даної задачі.

Приклад: У грі "Creatures" гравці могли створювати і навчати власних істот, які використовували генетичні алгоритми для розвитку та адаптації до середовища.

Ці види штучного інтелекту є лише кількома з багатьох, що використовуються в ігровій індустрії. Комбінація різних методів може дати більш складну та реалістичну поведінку NPC та опонентів у грі.

Оптимальний метод реалізації штучного інтелекту (ШІ) в якості противника в іграх на платформі Unity залежить від кількох факторів. Перш за все, необхідно враховувати тип гри і вимоги до поведінки суперника. Наприклад, якщо гра базується на стратегії та прийнятті рішень, може бути використана методика прийняття рішень на основі правил або дерево пріоритетів.

Також важливо враховувати обмеження ресурсів, такі як обчислювальна потужність або час реакції. У випадку обмежених ресурсів, можуть використовуватися спрощені алгоритми або методи зниження складності обчислень.

Кінцевим метою є досягнення бажаного рівня інтелектуальності та поведінки суперників в межах обмежень, що визначені в контексті гри. Тому розробники повинні вибирати той метод реалізації ШІ, який найкраще відповідає поставленим завданням і потребам гри.

2.3 Огляд методів розробки процедурної генерації та ШІ у Unity

Огляд методів розробки процедурної генерації та штучного інтелекту (ШІ) в середовищі Unity може допомогти розробникам створювати більш цікаві та динамічні ігрові досвіди. Нижче наведено кілька основних методів інтеграції процедурної генерації та ШІ у Unity [3].

Генеративні алгоритми: Генеративні алгоритми використовуються для створення процедурно згенерованих об'єктів, таких як ландшафти, міста, печери та інше. Ці алгоритми базуються на математичних принципах та випадкових параметрах для створення унікальних об'єктів у реальному часі. Наприклад, алгоритм шуму Перлина використовується для створення реалістичних текстур або нерегулярних форм ландшафту.

Системи правил: Системи правил використовуються для визначення поведінки об'єктів у грі. Вони зазвичай базуються на наборі правил, які визначають, як об'єкти реагують на певні події або взаємодіють між собою. Наприклад, можна створити систему правил для ворожих NPC, яка визначає їхні рухи, атаки та реакції на дії гравця.

Машинне навчання: Машинне навчання використовується для навчання суперників на основі даних та алгоритмів. У Unity існують пакети, такі як Unity ML-Agents, які дозволяють розробникам тренувати моделі штучного інтелекту, використовуючи методи навчання з підсиленням або навчання з наглядом. Наприклад, це може бути застосовано для створення суперників зі здатністю самостійно вчитися та адаптуватися до стратегій гравця.

Еволюційні алгоритми: Еволюційні алгоритми використовують принципи природного відбору та мутації для еволюції популяції індивідів. Вони можуть бути використані для оптимізації параметрів або поведінки суперників у грі. Наприклад, можна використовувати генетичні алгоритми для підбору оптимальних параметрів руху суперників у віртуальному середовищі.

Ці методи можуть використовуватися окремо або комбіновано для створення складної та реалістичної поведінки суперників у грі. Вибір конкретного методу залежить від типу гри, бажаного рівня інтелектуальності суперників та ресурсних обмежень розробки.

2.4 Засоби розробки ігрових застосунків на Unity

Unity Editor: Unity надає інтуїтивно зрозумілий графічний інтерфейс, відомий як Unity Editor, який дозволяє розробникам створювати, редагувати і візуалізувати гру у реальному часі. Він включає різноманітні редактори для роботи зі сценами, об'єктами, матеріалами, анімацією, скриптами та іншими елементами гри.

Система компонентів: Unity використовує систему компонентів, що дозволяє розробникам створювати об'єкти гри, додаючи до них різні компоненти. Компоненти визначають функціональність об'єкта, таку як графіка, фізика, звук, колізії, штучний інтелект тощо. Використовуючи систему компонентів, розробники можуть швидко створювати складні об'єкти та налаштовувати їх взаємодію.

Система скриптів: Unity підтримує розробку гри за допомогою скриптів, які базуються на мові програмування C#. Розробники можуть створювати власні скрипти для визначення поведінки об'єктів, реалізації логіки гри, обробки подій тощо. Скрипти можуть бути прив'язані до компонентів об'єктів і викликатися під час різних подій гри.

Asset Store: Unity має вбудований Asset Store, де розробники можуть придбати або завантажити безкоштовні ресурси, такі як готові моделі, текстури, звуки, ефекти тощо. Це дозволяє значно зекономити час і ресурси на створення всіх аспектів гри з нуля.

Фізична система: Unity має вбудовану фізичну систему, яка дозволяє створювати реалістичну поведінку об'єктів у грі. Розробники можуть встановлювати фізичні властивості об'єктів, такі як маса, колізії, гравітація,

динаміка руху тощо. Це дозволяє створювати реалістичні фізичні ефекти та взаємодію об'єктів.

Анімаційна система: Unity має потужну анімаційну систему, яка дозволяє розробникам створювати рухи та анімації для об'єктів у грі. Вона підтримує ієрархічні анімації, скелетну анімацію, морфінг та багато інших технік. Розробники можуть керувати анімаціями через скрипти або використовувати вбудований графічний інтерфейс для налаштування анімаційних параметрів.

Звукова система: Unity має вбудовану звукову систему, яка дозволяє розробникам додавати та керувати звуковими ефектами у грі. Вона підтримує різні формати звуку, можливість змішування звуків, налаштування ефектів ехо, реверберації, 3D звуку та інших аспектів звукової обробки.

Графічна система: Unity має потужну графічну систему, яка дозволяє створювати вражаючі візуальні ефекти у грі. Розробники можуть налаштовувати освітлення, тіні, текстури, матеріали, спеціальні ефекти та інші графічні параметри для досягнення бажаної візуальної якості гри.

Visual Studio: це інтегроване середовище розробки (IDE), розроблене компанією Microsoft, яке широко використовується для розробки програмного забезпечення, включаючи ігрові застосунки на платформі Unity. Visual Studio надає розробникам потужні інструменти для програмування, налагодження та управління проектами. Воно підтримує кілька мов програмування, включаючи C#, яка є однією з основних мов програмування для розробки ігор на Unity.

Tile Palette: це вбудований інструмент в Unity, призначений для створення та редагування тайлових карт (tilemaps) у ігрових застосунках. Tile Palette дозволяє розробникам швидко та зручно створювати різноманітні рівні та локації, використовуючи набір заздалегідь підготовлених тайлів (tiles). Тайли - це малий графічний елемент, який може бути повторно використаний для будівництва складних структур, таких як стіни, підлоги, об'єкти тощо.

Також до цих основних функціональних можливостей, Unity також має багато інших корисних функцій і інструментів, які полегшують розробку ігрових застосунків:

Мультиплатформеність: Unity дозволяє розробляти ігри для різних платформ, таких як ПК, консолі, мобільні пристрої, віртуальна реальність (VR) і змішана реальність (AR). Це дає можливість досягти більшої аудиторії і розповсюдити гру на різних пристроях.

Інтеграція зі сторонніми інструментами: Unity підтримує інтеграцію з різними зовнішніми інструментами і сервісами. Наприклад, ви можете використовувати зовнішні програми моделювання і анімації, такі як Blender або Maya, і імпортувати їхні ресурси безпосередньо в Unity. Також є можливість інтеграції з різними сервісами, наприклад, для аналітики, монетизації, соціальних функцій тощо.

Розширення функціональності: Unity надає можливість розширювати функціональність рушія шляхом використання розширень (плагінів) і скриптованої системи. Ви можете створювати власні розширення для додавання нових функцій і інструментів у Unity або використовувати розширення, розроблені спільнотою розробників Unity.

Кросплатформна розробка: Unity дозволяє використовувати загальний код для різних платформ, що спрощує розробку і підтримку кросплатформних ігор. Ви можете писати один набір коду на мові C# і використовувати його для різних платформ без необхідності написання окремого коду для кожної платформи.

Спільна робота: Unity надає інструменти для спільної роботи над проектом. Кілька розробників можуть працювати над одним проектом одночасно, синхронізуючи свої зміни та обмінюючись ресурсами через систему контролю версій.

Монетизація: Unity надає інструменти для монетизації ігрових застосунків. Ви можете інтегрувати рекламу, внутрішні покупки, підписки та інші механізми генерації доходу безпосередньо в свою гру.

Unity є потужним рушієм для створення ігор, який поєднує в собі різноманітні функціональні можливості і інструменти, що полегшують розробку і поліпшують якість ігрового досвіду. За допомогою методів процедурної генерації контенту та штучного інтелекту ви можете додати унікальність та глибину до своєї гри, що може привести до успіху на сучасному ринку ігор [10].

3 РОЗРОБКА ІГРОВОГО ЗАСТОСУНКУ З ВИКОРИСТАННЯМ ПРОЦЕДУРНОЇ ГЕНЕРАЦІЇ ТА ШТУЧНОГО ІНТЕЛЕКТУ

3.1 Опис цілей розробки

Розробка застосунку з використанням процедурної генерації та штучного інтелекту на рушії Unity має кілька основних призначень.

Створення повноцінної гри: Основною метою є розробка ігрового застосунку, який буде включати в себе ігрові елементи, що використовують процедурну генерацію контенту та штучний інтелект. Це дозволить створити цікаву та захоплюючу гру, яка варіюється з кожним запуском і надає користувачам нові виклики та враження.

Вивчення переваг обраних технологій: Розробка такого застосунку дозволить детально вивчити та експериментувати з методами процедурної генерації контенту та штучного інтелекту. Це дозволить оцінити переваги цих технологій у сучасному ринку ігрової індустрії та визначити їх потенціал для створення цікавих та унікальних ігрових досвідів.

Вивчення рушія Unity: Розробка застосунку на рушії Unity дозволить глибше вивчити можливості та переваги цього інструменту для створення ігрових середовищ з використанням процедурної генерації та штучного інтелекту. Це включає в себе вивчення API, інструментів, компонентів та ресурсів, які Unity надає для розробки ігор, а також дослідження його можливостей у контексті вищезгаданих технологій.

Таким чином, розробка застосунку з процедурною генерацією контенту та штучним інтелектом на рушії Unity має за мету створення цікавої гри з унікальним ігровим досвідом, вивчення та оцінку переваг обраних технологій та поглиблення знань про рушій Unity та його можливості.

3.2 Вимоги до ігрового середовища

Функціональні вимоги ігрового середовища включають:

1. Користувач повинен мати можливість відкрити ігрове середовище.
2. Користувач повинен мати можливість закрити ігрове середовище.
3. Користувач повинен мати можливість активувати алгоритм процедурної генерації для створення нового рівня.
4. Користувач повинен мати можливість грати в обраному рівні, що включає в себе бої з монстрами.

На рис. 9 зазначена use-case діаграма по користувацькому використанню застосунку.

Нефункціональні вимоги включають:

1. Продуктивність. Оптимізація системних вимог для користувача.
2. Простота у розумінні ігрових систем.
3. Надійність. Забезпечення стабільності та відсутності помилок у ігровому застосунку.

3.3 Проектування ігрового середовища

Розробка дизайну рівнів та ігрових об'єктів є важливим етапом проектування ігрового середовища. Процедурна генерація в цьому контексті дозволяє автоматично створювати різноманітні рівні, об'єкти та контент, що забезпечує більшу варіативність та унікальність геймплею.

Перш за все, важливо визначити параметри генерації, які визначатимуть характеристики та властивості створюваних об'єктів. Наприклад, це можуть бути розміщення об'єктів у просторі, їхні розміри, форма, колір, текстура та інші атрибути. Вибір цих параметрів впливатиме на вигляд та взаємодію об'єктів з гравцем.

Крім того, процедурна генерація може використовуватись для формування топографії рівнів. Це означає, що ігровий світ може автоматично генеруватись з урахуванням різних факторів, таких як рельєф місцевості,

наявність водних об'єктів, ландшафтних елементів тощо. Це дозволяє створювати різноманітність у місцевості та дозволяє гравцеві досліджувати нові території кожного разу, коли він грає.

Структура об'єктів також є важливим аспектом при розробці ігрового середовища. Процедурна генерація може використовуватись для створення різноманітних комбінацій об'єктів та їх взаємодії з гравцем. Наприклад, в залежності від умов гри, об'єкти можуть мати різні властивості, функції та способи взаємодії, що впливають на геймплей та стратегію гравця.

Крім процедурної генерації, важливим аспектом розробки ігрового середовища є інтелектуальні супротивники. Штучний інтелект може бути використаний для створення різних типів ворогів зі своїми власними стратегіями, тактиками та поведінкою. Інтелектуальні супротивники можуть реагувати на дії гравця, приймати рішення та навіть вчитися з кожного поєдинку. Це забезпечує більшу виклик і цікавість в грі, а також може стимулювати розвиток стратегічного мислення гравця.

Усі ці елементи впливають на враження та досвід гравця. Використання процедурної генерації та штучного інтелекту дозволяє створювати більш динамічні, варіативні та захоплюючі ігрові світи, які постійно випробовують навички та вміння гравця. Також було визначено саму систему взаємодій користувача у грі, можливість боїв з супротивниками, переміщення по рівню та можливості запуска процедурної генерації користувачем для створення нового рівня (Див. рис. 10).

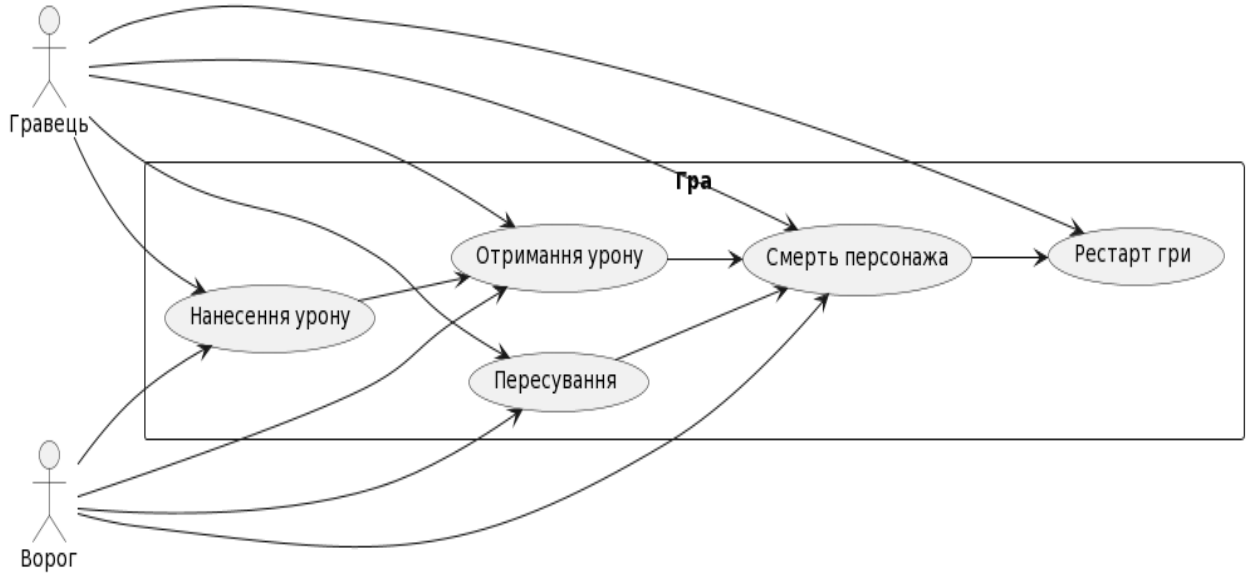

Рис. *10 Use-case діаграма*

У розділі проектування також враховується розробка інтелектуального супротивника. Штучний інтелект відповідає за поведінку супротивників у грі. Були визначені алгоритми прийняття рішень, логіку руху, реакцію на дії гравця та можливість адаптації до змінних умов гри. Були використані скріптовані системи на C# для створення систем штучного інтелекту для супротивників [6]. В супротивника розроблені системи, завдяки яким він може помічати гравця, переслідувати його, оминаючи перешкоди, та атакувати. Для розуміння наведено діаграму станів ШІ (Див. рис. 11).

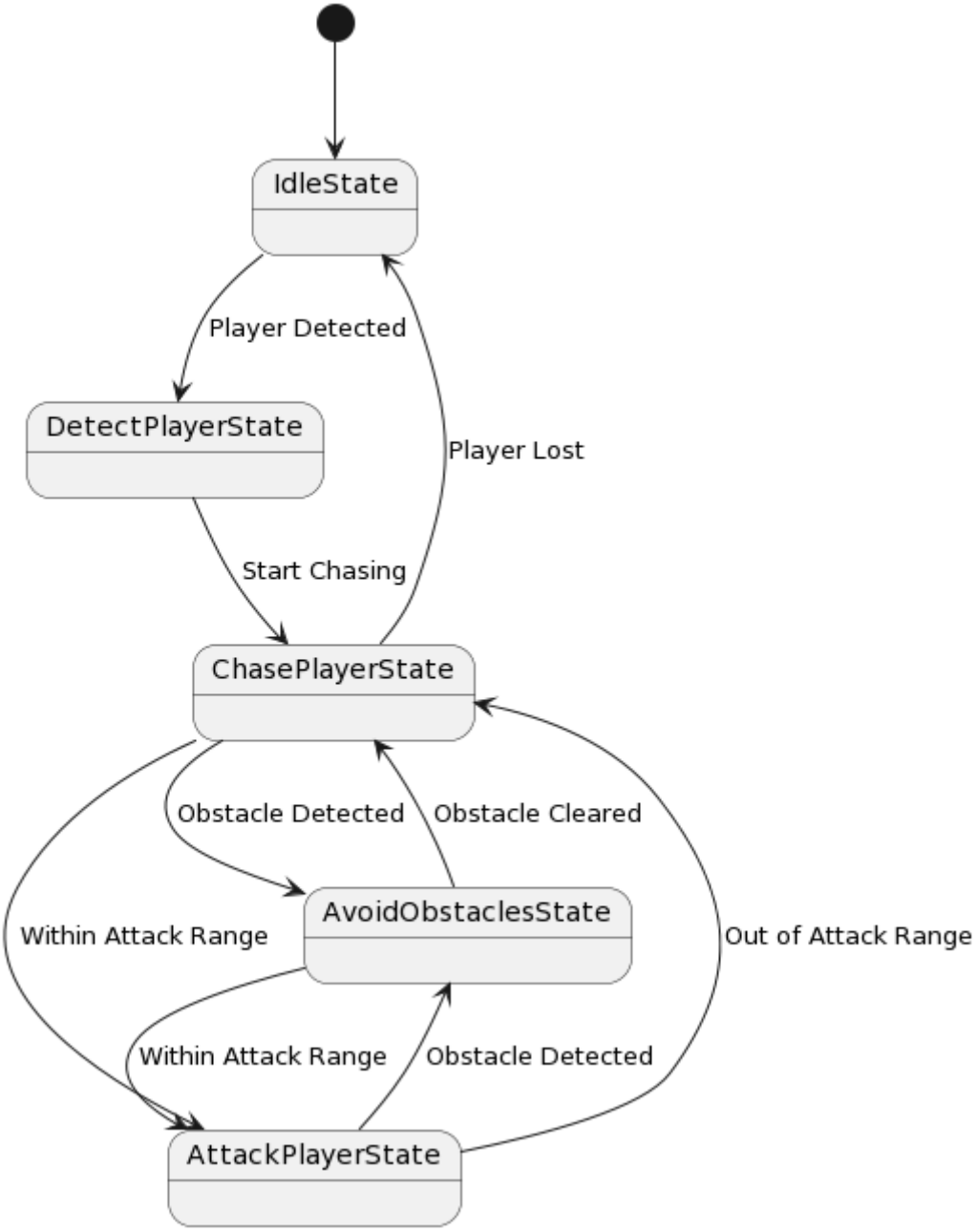

Рис. *11 Діаграма станів ШІ*

3.4 Засоби реалізації

Для створення ігрового застосунку використовувався рушій Unity разом з його вбудованими можливостями для створення процедурно згенерованого оточення. Розробка процедурної генерації, контролю гравця та поведінки ШІ виконувалась за допомогою середовища розробки Visual Studio 2019 та мови програмування C#.

Unity є потужним середовищем розробки, яке дозволяє створювати різноманітні ігрові застосунки. У розробці Rogue-like застосунку з використанням процедурної генерації та ШІ ви можете реалізовати різні класи та методи для реалізації функціональності гри. Ось кілька прикладів:

Клас "EnemyAI" (ШІ ворога):

Методи для реалізації поведінки ШІ ворога, такі як "DetectPlayer" (виявлення гравця), "ChasePlayer" (переслідування гравця) і "AttackPlayer" (атака гравця).

Методи для розпізнавання та оминання перешкод на шляху, наприклад, "ScanObstacles" (розпізнавання перешкод) і "AvoidObstacles" (оминання перешкод).

Клас "PlayerController" (Контролер гравця):

Методи для керування гравцем, такі як "Move" (рух), "Attack" (атака).

Методи по перевірці отримання шкоди, такі як "InitializeHealth" (ініціалізація здоров'я), "GetHit" (отримання шкоди).

Клас "CorridorFirstDungeonGenerator" (генератор рівня)

Методи для проведення процедурної генерації, наприклад "SimpleRandomWalk" (алгоритм процедурної генерації), "CorridorFirstGeneration" (генерація коридорів), "GenerateRooms" (генерація кімнат).

Клас "AgentAnimations" (анімація персонажів)

Методи для відтворення анімацій персонажів, такі як "RotateToPointer" (поглянути у напрямку погляду), "PlayAnimation"(програти анімацію).

Клас "WeaponParent" (Управління зброєю)

Методи управління зброєю, такі як "DetectColliders" (розпізнавання колайдерів), "DelayAttack" (затримка атаки).

Unity також надає широкий набір базових класів та методів для створення скриптів на C#, які ви можете використовувати при розробці ігрових застосунків. Ось кілька базових класів та їх методів в Unity:

Клас "MonoBehaviour":

Метод "Start()": Метод, який викликається під час початку виконання скрипта.

Метод "Update()": Метод, який викликається на кожному кадрі оновлення гри.

Метод "OnTriggerEnter(Collider other)": Метод, який викликається, коли об'єкт зіштовхується з іншим об'єктом.

Клас "Transform":

Метод "Translate(Vector3 translation)": Метод для зміни позиції об'єкта за вказаним вектором пересування.

Метод "Rotate(Vector3 rotation)": Метод для обертання об'єкта за вказаним вектором обертання.

Метод "Scale(Vector3 scale)": Метод для зміни масштабу об'єкта за вказаним вектором масштабування.

Клас "Rigidbody":

Метод "AddForce(Vector3 force)": Метод для додавання сили до фізичного тіла об'єкта.

Метод "MovePosition(Vector3 position)": Метод для зміни позиції фізичного тіла об'єкта.

Клас "Collider":

Метод "OnTriggerEnter(Collider other)": Метод, який викликається, коли об'єкт зіштовхується з іншим коллайдером.

Метод "OnCollisionEnter(Collision collision)": Метод, який викликаєтьс я, коли об'єкт зіштовхується з іншим коллайдером і спричиняє колізію.

Клас "Input":

Метод "GetKey(KeyCode key)": Метод для перевірки, чи натиснута вказана клавіша.

Метод "GetAxis(string axisName)": Метод для отримання значення вказаної вісі управління (наприклад, горизонтальної або вертикальної).

Ці базові класи та методи допомагають вам контролювати об'єкти, їхню фізичну поведінку, взаємодію з колайдерами та обробку вводу в ігрових

застосунках. Використовуючи їх разом з іншими функціями Unity, ви можете створювати різноманітні ігрові механіки та функціональність.

Також для створення ігрового середовища у Unity можуть застосовуватись різноманітні бібліотеки для спрощення та розширення процесу розробки:

2D Animation: надає всі необхідні інструменти та компоненти середовища виконання для скелетної анімації спрайтів.

2D Pixel Perfect: Це пакет в Unity, який допомагає забезпечити точність піксельного відображення для 2D-графіки, надаючи інструменти для ретро-стилю та дотримання роздільної здатності екрану.

Cinemachine: Це модуль в Unity, який надає розширені засоби камери, дозволяючи реалістично керувати кадром та композицією сцен в грі. Він дозволяє створювати складні камерні рухи, переходи між камерами та забезпечує плавну настройку.

Post Processing: Це пакет в Unity, який надає можливості пост-процесингу графіки. Він дозволяє додавати спеціальні ефекти, які покращують візуальний вигляд гри, такі як кольорові корекції, розмите зображення, тіні та багато інших.

Test Framework: Це набір інструментів в Unity, які допомагають розробникам створювати й виконувати автоматизовані тести для перевірки працездатності, стійкості та правильності роботи ігрових застосунків.

Timeline: Це інструмент в Unity, який дозволяє розробникам створювати та керувати складними сценаріями анімації та подій в грі. Він дозволяє розташовувати й налаштовувати ключові кадри, анімацію переходів, зміни параметрів об'єктів та багато іншого.

Unity UI: Це система для створення та керування інтерфейсом користувача в ігрових застосунках Unity. Вона дозволяє створювати кнопки, тексти, полоси прокрутки та інші елементи інтерфейсу, а також керувати їх взаємодією з користувачем.

Universal RP: Це легковага та масштабована система в Unity для рендерингу графіки. Вона дозволяє розробникам налаштовувати рівень деталізації графіки, використовувати різноманітні ефекти освітлення та тіней, а також працювати з різними платформами.

Unity Engine API: Це набір класів та методів, які надають доступ до функціональності самого рушія Unity. Вони дозволяють розробникам маніпулювати об'єктами, керувати фізикою, робити маніпуляції з анімацією, звуком та багато іншого.

UnityEngine.AI: Це модуль в Unity, який надає інструменти для розробки штучного інтелекту та навігації в ігрових застосунках. Він дозволяє створювати штучних ворогів, побудову шляхів, визначення колізій та прийняття рішень ШІ.

System.Collections: Це простір імен в мові програмування C#, який містить базові класи та інтерфейси для роботи з колекціями даних. Використовуючи класи з цього простору імен, розробники можуть працювати зі списками, масивами, хеш-таблицями та іншими структурами даних [10].

3.5 Розробка програми

Системні вимоги до розробленої гри

- Операційна система: Windows XP або новіша, або Mac OS X 10.1 або новіша.
- Процесор: Двоядерний процесор з тактовою частотою 1.4 ГГц або еквівалентний.
- Оперативна пам'ять: 1 ГБ.
- Графічна карта: З підтримкою DirectX 9 із 512 МБ відеопам'яті.
- Вільне місце на жорсткому диску: 50 МБ.
- Звукова карта: Сумісна з DirectX.
- Інші: Клавіатура та миша, доступ до Інтернету для онлайн-ігор або оновлень.

Встановлення необхідних ресурсів

По-перше, необхідно встановити Unity Hub, сам рушій Unity, та усі необхідні бібліотеки для того щоб приступити до розробки гри. Далі будуть приведені два рисунки, які зображають усі необхідні компоненти для розробки застосунку, а саме Unity 2020.3.48f1, середовище розробки Visual Studio 2019 та підтримка платформ. (див. Рис.12, 13).

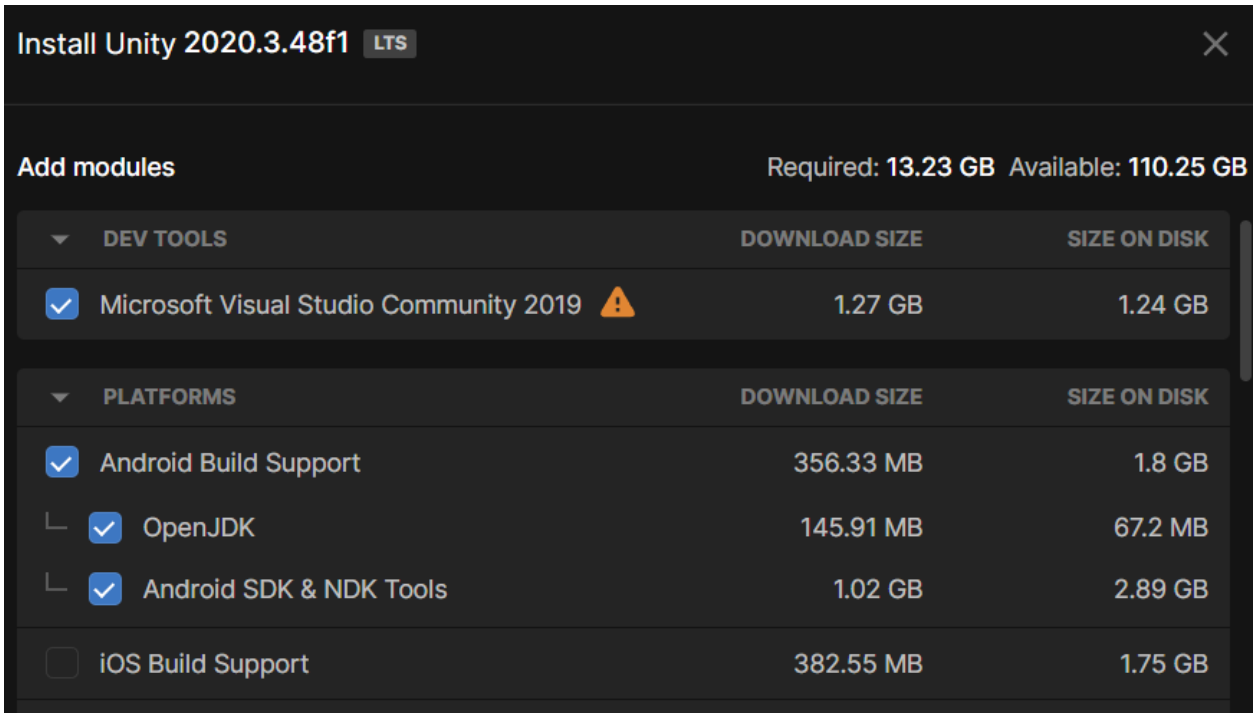

Рис. *12 Необхідні компоненти*

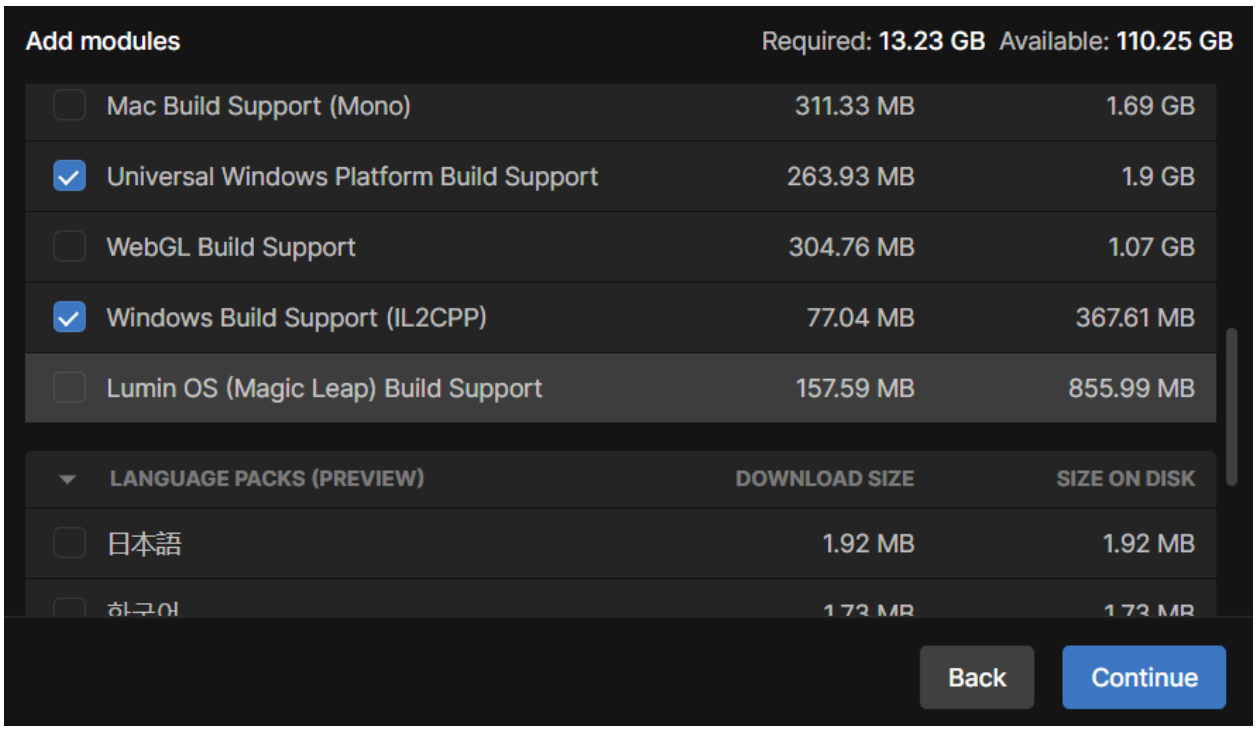

Рис. *13 Необхідні компоненти*

Програмна реалізація застосунку

В лістингу 1 зазначено клас створення рівня, при запуску гри він автоматично запускається, що приводить к процедурній генерації рівня. Числові змінні задають кількість та довжину згенерованих коридорів, а подальші методи створюють відповідні коридори. У кінцях цих коридорів створюються кімнати на алгоритмом Random Walk, за параметрами зазначених в Data-файлах для окремих типів та розмірів. (Див. Рис. 14).

Лістинг 1 *Стартова процедурна генерація*

```
public class CorridorFirstDungeonGenerator :
SimpleRandomWalkDungeonGenerator
{

    [SerializeField]
    private int corridorLength = 14, corridorCount = 5;
    [SerializeField]
    [Range(0.1f,1)]
   private float roomPercent = 0.8f;
```

```
        private Dictionary<Vector2Int, HashSet<Vector2Int>>
roomsDictionary
            = new Dictionary<Vector2Int, HashSet<Vector2Int>>();
        //corridor and room potential position
        private HashSet<Vector2Int> floorPositions,
corridorPositions;

        private List<Color> roomColors = new List<Color>();
        [SerializeField]
        private bool showRoomGizmo = false, showCorridorsGizmo;
        public UnityEvent<DungeonData> OnDungeonFloorReady;
        protected override void RunProceduralGeneration()
        {
            CorridorFirstGeneration();
    //choosen rooms
            DungeonData data = new DungeonData
            {
                roomsDictionary = this.roomsDictionary,
                corridorPositions = this.corridorPositions,
                floorPositions = this.floorPositions
            };
            OnDungeonFloorReady?.Invoke(data);
        }
        private void CorridorFirstGeneration()
        {
            floorPositions = new HashSet<Vector2Int>();
            HashSet<Vector2Int> potentialRoomPositions = new
HashSet<Vector2Int>();
    //create corridors
            CreateCorridors(floorPositions,
potentialRoomPositions);
            GenerateRooms(potentialRoomPositions);

        }
    //room generation
```

```
        private void GenerateRooms(HashSet<Vector2Int>
potentialRoomPositions)
        {
            HashSet<Vector2Int> roomPositions =
CreateRooms(potentialRoomPositions);
    //create missing rooms
            List<Vector2Int> deadEnds =
FindAllDeadEnds(floorPositions);
            CreateRoomsAtDeadEnd(deadEnds, roomPositions);
            floorPositions.UnionWith(roomPositions);
            tilemapVisualizer.PaintFloorTiles(floorPositions);
            WallGenerator.CreateWalls(floorPositions,
tilemapVisualizer);
        }
        private IEnumerator
GenerateRoomsCoroutine(HashSet<Vector2Int>
potentialRoomPositions)
        {
            yield return new WaitForSeconds(2);
            tilemapVisualizer.Clear();
            GenerateRooms(potentialRoomPositions);
            DungeonData data = new DungeonData
            {
                roomsDictionary = this.roomsDictionary,
                corridorPositions = this.corridorPositions,
                floorPositions = this.floorPositions
            };
            OnDungeonFloorReady?.Invoke(data);
        }

    //checkouts
    private void CreateRoomsAtDeadEnd(List<Vector2Int> deadEnds,
HashSet<Vector2Int> roomFloors)
        {
            foreach (var position in deadEnds)
            {
```

```
                if(roomFloors.Contains(position) == false)
                {
                    var room =
RunRandomWalk(randomWalkParameters, position);
                    SaveRoomData(position, room);
                    roomFloors.UnionWith(room);
                }
            }
        }
    //create borders
        private List<Vector2Int>
FindAllDeadEnds(HashSet<Vector2Int> floorPositions)
        {
            List<Vector2Int> deadEnds = new List<Vector2Int>();
            foreach (var position in floorPositions)
            {
                int neighboursCount = 0;
                foreach (var direction in
Direction2D.cardinalDirectionsList)
                {
                    if (floorPositions.Contains(position +
direction))
                        neighboursCount++;

                }
                if (neighboursCount == 1)
                    deadEnds.Add(position);
            }
            return deadEnds;
        }
        private HashSet<Vector2Int>
CreateRooms(HashSet<Vector2Int> potentialRoomPositions)
        {
            HashSet<Vector2Int> roomPositions = new
HashSet<Vector2Int>();
```

```
            int roomToCreateCount =
Mathf.RoundToInt(potentialRoomPositions.Count * roomPercent);
            List<Vector2Int> roomsToCreate =
potentialRoomPositions.OrderBy(x =>
Guid.NewGuid()).Take(roomToCreateCount).ToList();
            ClearRoomData();
            foreach (var roomPosition in roomsToCreate)
            {
                var roomFloor =
RunRandomWalk(randomWalkParameters, roomPosition);

                SaveRoomData(roomPosition, roomFloor);
                roomPositions.UnionWith(roomFloor);
            }
            return roomPositions;
        }
        private void ClearRoomData()
        {
            roomsDictionary.Clear();
            roomColors.Clear();
        }
        private void SaveRoomData(Vector2Int roomPosition,
HashSet<Vector2Int> roomFloor)
        {
            roomsDictionary[roomPosition] = roomFloor;
            roomColors.Add(UnityEngine.Random.ColorHSV());
        }
        private void CreateCorridors(HashSet<Vector2Int>
floorPositions,
            HashSet<Vector2Int> potentialRoomPositions)
        {
            var currentPosition = startPosition;
            potentialRoomPositions.Add(currentPosition);
            for (int i = 0; i < corridorCount; i++)
            {
```

```
                var corridor =
ProceduralGenerationAlgorithms.RandomWalkCorridor(currentPositio
n, corridorLength);
                currentPosition = corridor[corridor.Count - 1];
                potentialRoomPositions.Add(currentPosition);
                floorPositions.UnionWith(corridor);
            }
            corridorPositions = new
HashSet<Vector2Int>(floorPositions);
        }
    //draw borders
        private void OnDrawGizmosSelected()
        {
            if (showRoomGizmo)
            {
                int i = 0;
                foreach (var roomData in roomsDictionary)
                {
                    Color color = roomColors[i];
                    color.a = 0.5f;
                    Gizmos.color = color;
                    Gizmos.DrawSphere((Vector2)roomData.Key,
0.5f);
                    foreach (var position in roomData.Value)
                    {
                        Gizmos.DrawCube((Vector2)position + new
Vector2(0.5f,0.5f), Vector3.one);
                    }
                    i++;
                }
            }
            if (showCorridorsGizmo && corridorPositions != null)
            {
                Gizmos.color = Color.magenta;
                foreach (var corridorTile in corridorPositions)
                {
```

```
                    Gizmos.DrawCube((Vector2)corridorTile + new
Vector2(0.5f, 0.5f), Vector3.one);
                }
            }
        }
    }
```

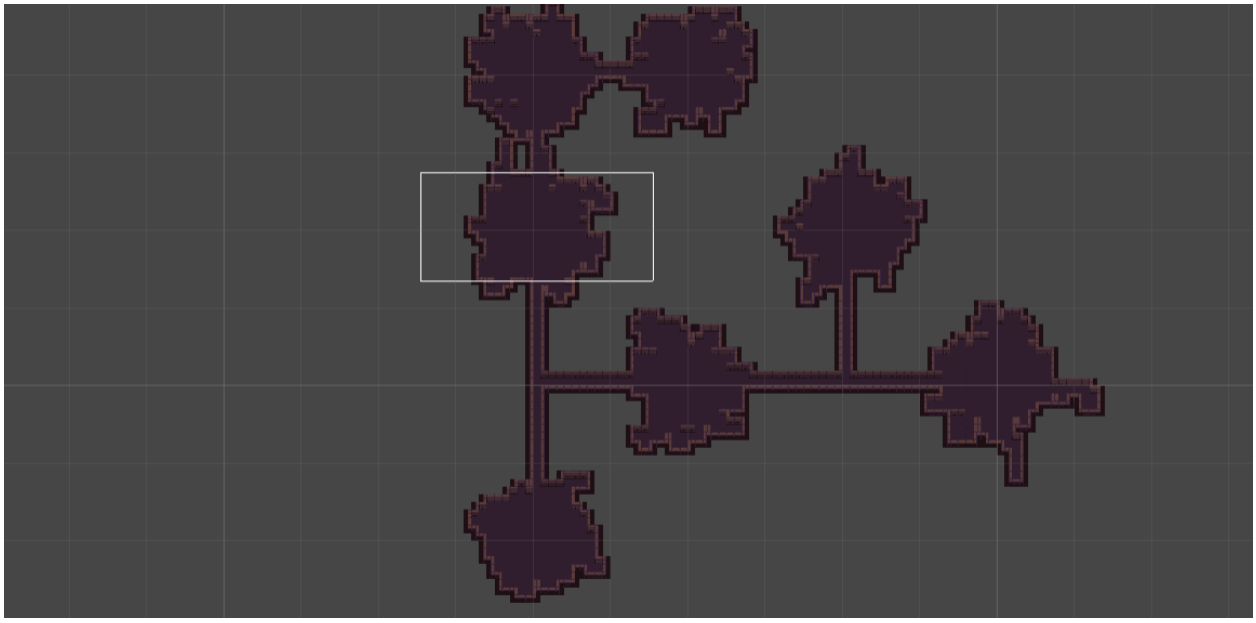

Рис. *14 Процедурна генерація*

У лістингу 2 зазначено принцип роботи ШІ ворогів, завдяки якому вони сприймають гравця, переслідують та атакують його

Лістинг 2 *Робота ШІ ворога відносно гравця*

```
public class EnemyAI : MonoBehaviour
{
    [SerializeField]
    private List<SteeringBehaviour> steeringBehaviours;
    [SerializeField]
    private List<Detector> detectors;
    [SerializeField]
    private AIData aiData;
    [SerializeField]
    private float detectionDelay = 0.05f, aiUpdateDelay =
0.06f, attackDelay = 1f;
    [SerializeField]
```

```
        private float attackDistance = 0.5f;
        //Inputs sent from the Enemy AI to the Enemy controller
        public UnityEvent OnAttackPressed;
        public UnityEvent<Vector2> OnMovementInput,
OnPointerInput;
        [SerializeField]
        private Vector2 movementInput;
        [SerializeField]
        private ContextSolver movementDirectionSolver;
        bool following = false;
        private void Start()
        {
            //Detecting Player and Obstacles around
            InvokeRepeating("PerformDetection", 0,
detectionDelay);
        }
        private void PerformDetection()
        {
            foreach (Detector detector in detectors)
            {
                detector.Detect(aiData);
            }
        }
        private void Update()
        {
            //Enemy AI movement based on Target availability
            if (aiData.currentTarget != null)
            {
                //Looking at the Target

OnPointerInput?.Invoke(aiData.currentTarget.position);
                if (following == false)
                {
                    following = true;
                    StartCoroutine(ChaseAndAttack());
                }
```

```
            }
            else if (aiData.GetTargetsCount() > 0)
            {
                //Target acquisition logic
                aiData.currentTarget = aiData.targets[0];
            }
            //Moving the Agent
            OnMovementInput?.Invoke(movementInput);
        }
        private IEnumerator ChaseAndAttack()
        {
            if (aiData.currentTarget == null)
            {
                //Stopping Logic
                Debug.Log("Stopping");
                movementInput = Vector2.zero;
                following = false;
                yield break;
            }
            else
            {
                float distance =
Vector2.Distance(aiData.currentTarget.position,
transform.position);
                if (distance < attackDistance)
                {
                    //Attack logic
                    movementInput = Vector2.zero;
                    OnAttackPressed?.Invoke();
                    yield return new
WaitForSeconds(attackDelay);
                    StartCoroutine(ChaseAndAttack());
                }
                else
                {
                    //Chase logic
```

```
                    movementInput =
movementDirectionSolver.GetDirectionToMove(steeringBehaviours,
aiData);
                    yield return new
WaitForSeconds(aiUpdateDelay);
                    StartCoroutine(ChaseAndAttack());
                }
            }
        }
    }
```

У лістингу 3 реалізовано сприймання ШІ перешкод та уходу від них, для того щоб він міг ефективно переслідувати гравця, для цього використовується в основі система колайдерів (Див. рис.15).

Лістинг 3 *Логіка уходу від перешкод*

```
public class ObstacleAvoidanceBehaviour : SteeringBehaviour
{
    [SerializeField]
    private float radius = 2f, agentColliderSize = 0.6f;
    [SerializeField]
    private bool showGizmo = true;
    //gizmo parameters
    float[] dangersResultTemp = null;
    public override (float[] danger, float[] interest)
GetSteering(float[] danger, float[] interest, AIData aiData)
    {
        foreach (Collider2D obstacleCollider in
aiData.obstacles)
        {
            Vector2 directionToObstacle
                =
obstacleCollider.ClosestPoint(transform.position) -
(Vector2)transform.position;
            float distanceToObstacle =
directionToObstacle.magnitude;
```

```
                //calculate weight based on the distance Enemy<-
-->Obstacle
                float weight
                    = distanceToObstacle <= agentColliderSize
                    ? 1
                    : (radius - distanceToObstacle) / radius;
                Vector2 directionToObstacleNormalized =
directionToObstacle.normalized;
                //Add obstacle parameters to the danger array
                for (int i = 0; i <
Directions.eightDirections.Count; i++)
                {
                    float result =
Vector2.Dot(directionToObstacleNormalized,
Directions.eightDirections[i]);

                    float valueToPutIn = result * weight;
                    //override value only if it is higher than
the current one stored in the danger array
                    if (valueToPutIn > danger[i])
                    {
                        danger[i] = valueToPutIn;
                    }
                }
            }
            dangersResultTemp = danger;
            return (danger, interest);
        }
        private void OnDrawGizmos()
        {
            if (showGizmo == false)
                return;
            if (Application.isPlaying && dangersResultTemp !=
null)
            {
                if (dangersResultTemp != null)
```

```
                {
                    Gizmos.color = Color.red;
                    for (int i = 0; i <
dangersResultTemp.Length; i++)
                    {
                        Gizmos.DrawRay(
                            transform.position,
                            Directions.eightDirections[i] *
dangersResultTemp[i]
                            );
                    }
                }
            }
            else
            {
                Gizmos.color = Color.cyan;
                Gizmos.DrawWireSphere(transform.position,
radius);
            }
        }
    }
    public static class Directions
    {
        public static List<Vector2> eightDirections = new
List<Vector2>{
                new Vector2(0,1).normalized,
                new Vector2(1,1).normalized,
                new Vector2(1,0).normalized,
                new Vector2(1,-1).normalized,
                new Vector2(0,-1).normalized,
                new Vector2(-1,-1).normalized,
                new Vector2(-1,0).normalized,
                new Vector2(-1,1).normalized
            };
    }
```

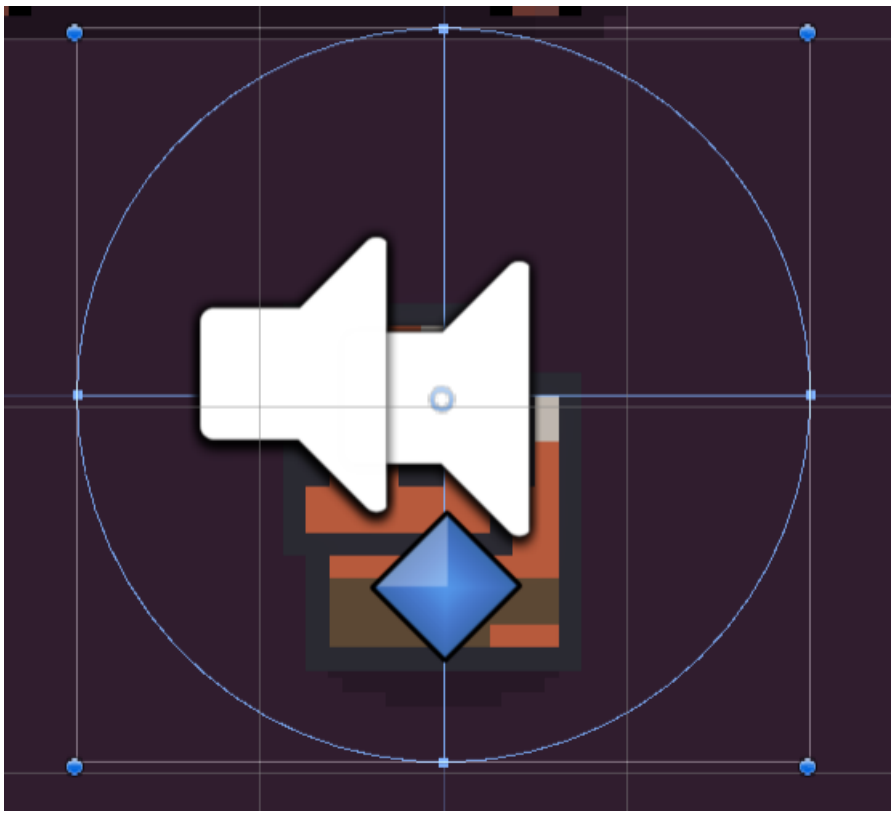

Рис. 15 *Колайдери ворогів*

3.6 Тестування програми

Тестування гри є важливою частиною розробки, оскільки воно допомагає виявляти помилки та переконуватися, що функціональність працює належним чином. Для ігрового застосунку з використанням процедурної генерації та штучного інтелекту (ШІ) у вигляді ізометричного rogue-like, тестування має особливе значення, оскільки ці елементи можуть впливати на геймплей та поведінку ігрових об'єктів.

Нижче наведено кілька важливих перевірок, які було проведено при тестуванні програми:

Тестування процедурної генерації:

Перевірено, чи генерується графіка, рівні та об'єкти згідно з очікуваннями.

Виконано тестування на випадкових комбінаціях параметрів генерації, щоб перевірити стабільність та поведінку системи.

Приклади випадкових генерацій рівнів зазначено у наступних зображеннях (Див. рис. 16, 17):

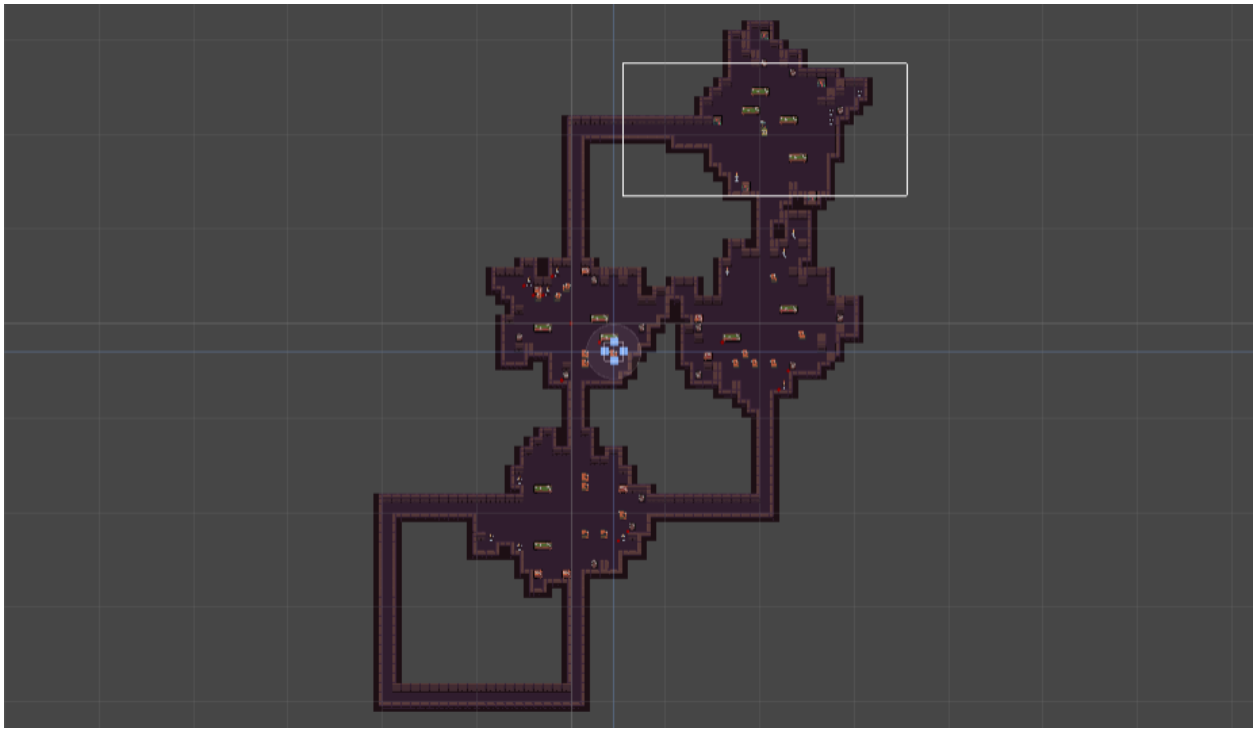

Рис. *16 Тестування генерації*

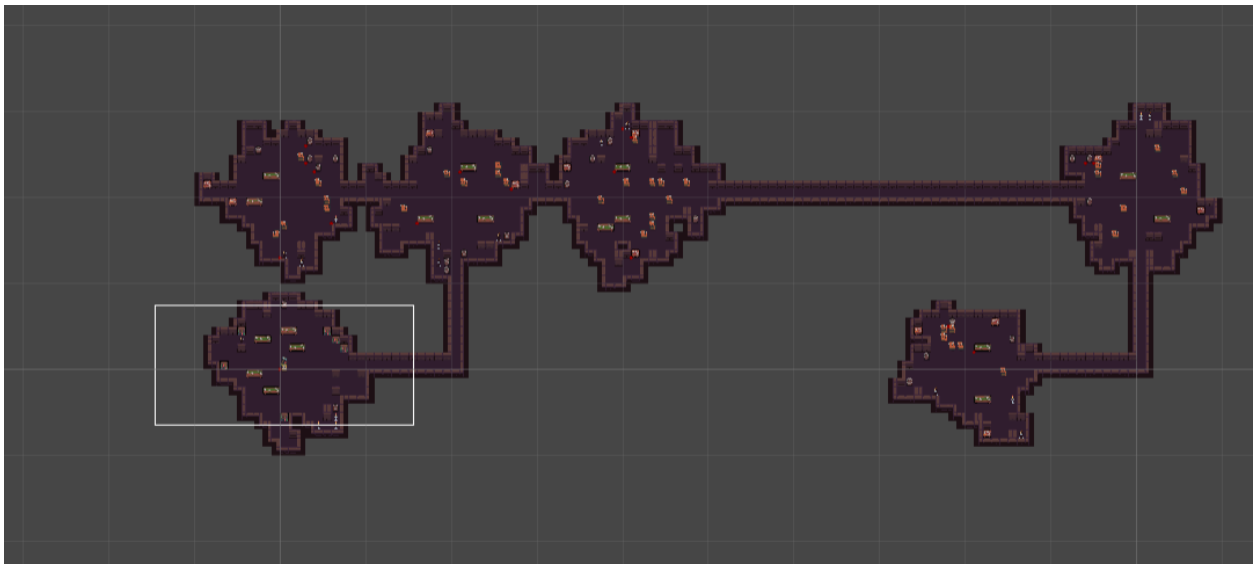

Рис. *17 Тестування генерації*

Як ми бачимо генерація рівней проходить повністю у відповідності до очікувань.

Тестування штучного інтелекту (ШІ):

Перевірено, що ШІ відповідає на дії гравця відповідно до очікуваного поведінки.

Перевірено, чи ШІ правильно розпізнає перешкоди, оминає їх та взаємодіє з ігровим світом (Див. рис. 18).

Рис. *18 Переслідування гравця ворогами*

Тестування геймплею та механік гри:

Перевірено, що механіка гри, така як рух, атака, зіткнення та інші взаємодії, працюють належним чином.

Перевірено, чи гра відповідає на очікувані дії гравця та чи відображаються результати правильно.

Тестування продуктивності та оптимізація:

Перевірено, чи гра працює з достатньою продуктивністю, зокрема з урахуванням процедурної генерації та ШІ.

Виконано тестування на різних пристроях та на різних налаштуваннях графіки, щоб переконатися, що гра працює оптимально та не має затримок.

Тестування на помилки та відладка:

Здійснено тестування на різних сценаріях та в різних умовах, щоб виявити та виправити помилки.

Використано відладчик Unity для виявлення та виправлення проблем з кодом та функціональністю.

3.7 Підсумок

У ході розробки ігрового застосунку на базі рушія Unity з використанням процедурної генерації та штучного інтелекту було виявлено, що ці технології відкривають безмежні можливості для створення захоплюючих ігрових досвідів. Процедурна генерація дозволяє створювати унікальні та різноманітні рівні, об'єкти та контент в грі, що забезпечує більшу повторюваність, варіативність та цікавість для гравців.

Застосування штучного інтелекту у грі дозволяє створити реалістичних інтелектуальних супротивників, які можуть адаптуватися до дій гравця, приймати рішення на основі аналізу ситуації. Це забезпечує глибину геймплею та виклик для гравців, покращує іммерсивність та відчуття взаємодії з ігровим світом.

Процес розробки ігрового застосунку на базі рушія Unity вимагає глибоких знань програмування, розуміння принципів геймдизайну, вміння використовувати інструменти процедурної генерації та штучного інтелекту. Був проведений детальний аналіз вимог до гри та обмежень ресурсів, що дозволило створити оптимальний метод реалізації.

Комбінування різних методів імплементації штучного інтелекту разом з процедурною генерацією дало приріст у реалістичності та складності поведінки супротивників.

Розробка ігрового застосунку на базі рушія Unity з використанням процедурної генерації та штучного інтелекту є складним завданням, але надзвичайно захоплюючим і творчим процесом. Вона вимагає уважного аналізу вимог, планування, розробки та тестування. Правильний вибір інструментів, розуміння їх можливостей та особливостей, а також глибокі знання геймдизайну допомогли досягти високої якості ігрового застосунку та вражень для гравців.

ВИСНОВКИ

У ході виконання кваліфікаційної роботи було отримано наступні результати:

1. Досліджено та вивчено концепцію процедурної генерації в ігровому середовищі з використанням Unity. Були вивчені основні алгоритми та методи генерації об'єктів, рівнів, текстур та інших елементів гри.

2. Вивчено практичні аспекти розробки ШІ для неігрових персонажів в ігровому середовищі з використанням Unity. Були досліджені алгоритми прийняття рішень, навігації, поведінкових агентів та інші техніки, що забезпечують реалістичну поведінку неігрових персонажів.

3. Розроблено ігрове середовище з використанням процедурної генерації, яке включає генерацію рівнів, об'єктів та інших елементів гри. Було реалізовано систему ШІ для неігрових персонажів, що дозволяє їм взаємодіяти з оточуючим світом та приймати рішення на основі заданих правил.

4. Проведено тестування розробленого ігрового середовища. Тести включали перевірку реалістичності та функціональності процедурної генерації, а також ефективність та правильність роботи системи ШІ для неігрових персонажів.

5. Результати тестування підтвердили успішну реалізацію процедурної генерації в ігровому середовищі та функціональність системи ШІ для неігрових персонажів. Було підтверджено, що розроблене середовище забезпечує динамічну геймплей-середу.

Рекомендації для подальшого розвитку включають:

Покращення алгоритмів процедурної генерації для більшої різноманітності згенерованих об'єктів.

Розширення системи ШІ для неігрових персонажів шляхом додавання нових алгоритмів та можливостей, що дозволять більш складні та інтелектуальні реакції неігрових персонажів на зміни в грі.

Подальше тестування та оптимізація ігрового середовища.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Short T., Adams T. Procedural Generation in Game Design. 2017. 313p.
2. Millington I., Funge J. Artificial Intelligence for Games 2009. 896p.
3. Shaker N., Togelius J., Nelson M. Unity Procedural Content Generation in Games: A Textbook and an Overview of Current Research. 2016. 253p.
4. Hocking J. Unity in Action: Multiplatform Game Development in C# 2018. 400p.
5. UNITY MACHINE LEARNING AGENTS URL: https://unity.com/products/machine-learning-agents (дата зверення: 02.06.2023)
6. Rabin S. Game AI Pro: Collected Wisdom of Game AI Professionals. 2013. 628p.
7. Subnautica Review on IGN URL: https://www.ign.com/articles/2018/01/26/subnautica-review (дата зверення: 11.06.2023)
8. No Man’s Sky Review on IGN URL: https://www.ign.com/articles/2019/09/14/no-mans-sky-beyond-review (дата зверення: 10.06.2023)
9. Graphic assets used for project URL: https://pixel-poem.itch.io/dungeon-assetpuck (дата зверення: 21.05.2023)
10. Unity Asset Store URL: https://assetstore.unity.com/ (дата звернення: 04.05.2023)