МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Острозька академія»
Економічний факультет
Кафедра економіко-математичного моделювання та інформаційних технологій

КВАЛІФІКАЦІЙНА РОБОТА
на здобуття освітнього ступеня бакалавра

на тему: **«СТВОРЕННЯ ШТУЧНОГО ІНТЕЛЕКТУ ДЛЯ ІМІТАЦІЇ ДІЙ ГРАВЦЯ В БАГАТОКОРИСТУВАЦЬКІЙ ГРІ»**

Виконав: студент 4 курсу, групи КН-4
першого (бакалаврського) рівня вищої освіти
спеціальності 122 Комп'ютерні науки
освітньо-професійної програми «Комп'ютерні науки»
Дзюнь Владислав Леонідович

Керівник: кандидат технічних наук, доцент,
Шатний Сергій В'ячеславович

Рецензент: *кандидат технічних наук, доцент доцент кафедри прикладної математики та кібербезпеки Донецького національного університету імені Василя Стуса*
Загоруйко Любов Василівна

РОБОТА ДОПУЩЕНА ДО ЗАХИСТУ
Завідувач кафедри економіко-математичного моделювання та інформаційних технологій _____ (проф., д.е.н. Кривицька О.Р.)
Протокол № 11 від «18 » травня 2024 р.

Острог, 2024

Міністерство освіти і науки України
Національний університет «Острозька академія»

Факультет: економічний
Кафедра: економіко-математичного моделювання та інформаційних технологій
Спеціальність: 122 Комп'ютерні науки
Освітньо-професійна програма: Комп'ютерні науки

ЗАТВЕРДЖУЮ
Завідувач кафедри економіко-математичного моделювання
та інформаційних технологій

_____ Ольга КРИВИЦЬКА
«____» _____20__ р.

ЗАВДАННЯ
на кваліфікаційну роботу студента
Дзюнь Владислава Леонідовича
(прізвище, ім'я, по батькові)

1. Тема роботи: Створення
штучного інтелекту для імітації дій гравця в багатокористувацькій грі
керівник роботи: Шатний Сергій В'ячеславович, кандидат технічних наук, доцент
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Затверджено наказом ректора НаУОА від 03.11.2023 р., No 98.

2. Термін здачі студентом закінченої роботи/проєкту: "31" травня 2024 року.
3. Вихідні дані до роботи: Unity, WebSockets.
4. Перелік завдань, які належить виконати: Визначити жанр та тему гри, створення сцени та об'єктів, реалізувати рух персонажів з використанням Unity, налагодити фізику об'єктів та перешкод, звукове оформлення. Налаштувати під'єднання до гри через мобільний пристрій, керування персонажем. Розробити систему прийняття рішень та навігацій штучного інтелекту
5. Перелік графічного матеріалу: рисунки, таблиці
6. Консультанти розділів роботи:

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв
1	Шатний С. В.	01.12.2024р.	01.12.2024р.
2	Шатний С. В.	01.12.2024р.	01.12.2024р.
3	Шатний С. В.	01.12.2024р.	01.12.2024р.

7. Дата видачі завдання: 01.12.2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів	Примітка
1	Затвердження теми роботи/проєкту.	до 31.10.23	
2	Постанова технічного завдання.	до 01.12.23	
3	Дослідження і збір вихідних даних.Визначення жанру, темита основної історії гри.	до 25.02.24	
4	Створення планів рівнів та локацій.Розробка дизайнувідкритих локацій.	до 14.04.24	
5	Розробка дизайну локацій.	до 01.05.24	
6	Реалізація візуальних ефектів та додаткових можливостейдодатку.Створення штучного інтелекту для неігрових персонажівв	до 08.05.24	
7	Провести тестування гри для виявлення та виправленняпомилок та проблем з геймплеєм.	до 12.05.24	
8	Тестування гри та її оптимізація.	до 13.05.24	
9	Попередній захист кваліфікаційної роботи/проєкту.	до 16.05.24	
10	Здача кваліфікаційної роботи/проєкту на кафедру.	до 31.05.24	

Студент: _____ Владислав ДЗЮНЬ

(підпис)

Керівник кваліфікаційної роботи: _____ Сергій ШАТНИЙ

(підпис)

АНОТАЦІЯ

кваліфікаційної роботи

на здобуття освітнього ступеня бакалавра

Тема: *Створення штучного інтелекту для імітації дій гравця в багатокористувацькій грі*

Автор: *Дзюнь Владислав Леонідович*

Науковий керівник*: Шатний Сергій В'ячеславович*

Захищена «_____»_____ 20__ року.

Пояснювальна записка до кваліфікаційної роботи: *49 с., 20 рис., 0 (кількість таблиць) табл., 0 (кількість додатків)*

додатків, _____ (кількість джерел) джерел.

Ключові слова: *Unity, C#, мобільна гра, багатокористувацька гра, HappyFunTimes, офлайн гра, ігрова механіка, анімація, геймплей, мультиплеєр, штучний інтелект, алгоритми, A**

Короткий зміст праці: *Дипломна робота присвячена розробці багатокористувацької гри для мобільних пристроїв з використанням Unity та C#. Гра є офлайн 2D проектом, де гравці використовують мобільні пристрої як контролери, підключаючись до центрального хоста, який координує гру та транслює її на великий екран.*
Теоретичні основи включають обговорення проблеми глибини та інтерактивності в багатьох існуючих іграх і пропонують вирішення через створення кооперативної 2D офлайн гри. Використання Unity дозволяє створити 2D ігровий світ, анімації та програмувати логіку гри на C#. Плагін HappyFunTimes забезпечує мультиплеєрну архітектуру з одним сервером та численними клієнтами, що дозволяє масштабувати систему для великої кількості гравців.

Прикладна розробка гри описує технічне завдання, основний задум гри, ігрові механіки та структуру проекту. Гра передбачає поділ гравців на команди "Creatures" і "Goblins", де перші ловлять гоблінів, а другі збирають ящики із золотом. Ігрові механіки включають рух гравців, механіку захоплення, систему набору очок, статистику гравців та взаємодію з об'єктами.
Розробка ігрового поля за допомогою Tile Palette, механіка руху гравців, керування станом гри та UI через клас GameManager, а також мультиплеєрна механіка з використанням WebSockets для синхронізації дій гравців детально розглянуті в роботі.

Створення префабів в Unity дозволяє централізовано керувати ігровими об'єктами. Префаби використовуються для створення складних ігрових об'єктів, таких як персонажі, з можливістю повторного використання у різних сценах.

Для підвищення інтелектуальності ігрового процесу використовується алгоритм A для пошуку оптимального шляху. A* базується на комбінації вартості пройденого шляху (G cost) та евристичної оцінки (H cost), що дозволяє ефективно знаходити найкоротший шлях до цілі.*

ANNOTATION

of the qualification work

for obtaining a bachelor's degree

Topic: Creation of artificial intelligence to simulate player actions in a multiplayer game

Author: Dziun Vladyslav Leonidovych

Academic advisor: Shatnyi Sergii Viacheslavovych

Defended on the «_____»_____ 20__ year.

Explanatory note to the qualification work: *49 p., 20 fig., 0 (number of tables) tables, 0 (number of appendices) appendices, _____ (number of sources) sources.*

Keywords: *Unity, C#, mobile game, multiplayer game, HappyFunTimes, offline game, game mechanics, animation, gameplay, multiplayer, artificial intelligence, algorithms, A**

Brief content of the work: *The thesis is dedicated to the development of a multiplayer game for mobile devices using Unity and C#. The game is an offline 2D project where players use mobile devices as controllers, connecting to a central host that coordinates the game and broadcasts it to a large screen.*

The theoretical foundations include a discussion of the issues of depth and interactivity in many existing games and propose a solution through the creation of a cooperative 2D offline game. The use of Unity allows for the creation of a 2D game world, animations, and programming of game logic in C#. The HappyFunTimes plugin provides a multiplayer architecture with a single server (host) and numerous clients, allowing the system to scale for a large number of players.

The practical development of the game describes the technical requirements, the main concept of the game, game mechanics, and the project structure. The game involves dividing players into two teams: "Creatures" and "Goblins", where the former catch goblins, and the latter collect gold boxes. The game mechanics include player movement, capture mechanics, a scoring system, player statistics, and interaction with objects.

The development of the game field using the Tile Palette, player movement mechanics, game state management and UI through the GameManager class, as well as multiplayer mechanics using WebSockets for synchronizing player actions are discussed in detail in the work.

The creation of prefabs in Unity allows centralized management of game objects. Prefabs are used to create complex game objects, such as characters, with the possibility of reuse in different scenes.
To enhance the intelligence of the gameplay, the A algorithm is used for optimal pathfinding. A* is based on a combination of the cost of the path taken (G cost) and a heuristic estimate (H cost), which allows for efficiently finding the shortest path to the goal.*

ЗМІСТ

ВСТУП **7**

РОЗДІЛ 1. ТЕОРЕТИЧНІ ОСНОВИ РОЗРОБКИ БАГАТОКОРИСТУВАЦЬКОЇ ГРИ ТА ШТУЧНОГО ІНТЕЛЕКТУ **5**

1.1. Опис предметного середовища (функціональної моделі, процесу діяльності) 5

1.2. Огляд існуючих ігрових додатків за даною тематикою 7

1.3. Постановка задачі 9

РОЗДІЛ 2. ПРОЕКТУВАННЯ ІГРОВОГО ЗАСТОСУНКУ **13**

2.1. Технічне завдання 13

2.2. Опис основного задуму 13

2.3. Опис ігрових механік: 13

2.4. Опис структури проєкту та функціоналу 14

2.5. Ігрові предмети 16

2.6. Штучний інтелект 18

РОЗДІЛ 3. РЕАЛІЗАЦІЯ ІГРОВОГО ЗАСТОСУНКУ **29**

3.1. Розробка ігрового поля 29

3.2. Створення механіки руху гравців 31

3.3. Керування станом гри та UI 32

3.4. Імплементація мультиплеєрної механіки 33

3.5. Реалізація ігрових об'єктів 35

3.6. Створення анімації 38

3.7. Інтеграція A* в ігровий застосунок 40

ВИСНОВКИ **42**

ДОДАТКИ **47**

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ І ТЕРМІНІВ

AI - Artificial Intelligence.

NPC - Non-Player Character.

UI – User Interfac

ВСТУП

У сучасному світі цифрових розваг, де мобільні ігри відіграють ключову роль, мій курсовий проєкт зосереджується на створенні інноваційної багатокористувацької 2D офлайн гри для мобільних пристроїв, розробленої за допомогою Unity та C#. Цей проєкт має на меті створення захоплюючого ігрового досвіду, де гравці використовують свої мобільні пристрої як контролери, підключаючись до центрального хоста, який координує гру та транслює її на великий екран. Однією з основних тем цього проєкту є розробка та інтеграція штучного інтелекту (ШІ), який забезпечує реалістичну та динамічну поведінку неігрових персонажів (NPC).

Мета роботи - розробити штучний інтелект (ШІ), здатний імітувати дії гравця в багатокористувацькому середовищі, шляхом створення мобільного ігрового застосунку. Гра дозволить грати офлайн, що є ключовим для гнучкості та доступності гри. Основна увага буде зосереджена на створенні глибокого та захоплюючого геймплею, який забезпечує взаємодію між гравцями через мобільні пристрої, а також на інтеграції ШІ для ефективного керування NPC та створення динамічного ігрового процесу

Завдання - розробка концепції гри: створити детальний опис ігрового процесу, сюжету та механік гри. Розробка ігрового рушія: використати Unity для створення основи гри; інтегрувати плагін HappyFunTimes для забезпечення багатокористувацької взаємодії через мобільні пристрої. Програмування ігрових механік: написати код на C# для реалізації ігрових правил та логіки; забезпечити можливість підключення мобільних пристроїв до центрального хоста. Розробка штучного інтелекту для NPC: створити та інтегрувати алгоритми ШІ для динамічної та реалістичної поведінки NPC; налаштувати ШІ для адаптації до різних сценаріїв гри. Тестування та налагодження гри: провести тестування гри на різних мобільних пристроях; виправити виявлені помилки та покращити продуктивність гри.

Об'єктом дослідження є розробка унікальної багатокористувацької офлайн гри, яка забезпечує соціальну взаємодію, інтерактивність та інтелектуальну поведінку

NPC. Гра орієнтована на велику кількість гравців - від 6 до 30 осіб, що робить її ідеальною для зібрань, святкувань та різних івентів. Важливим аспектом є впровадження ШІ, який здатен адаптуватися до різних сценаріїв гри та забезпечувати цікавий геймплей.

Предмет дослідження

Предмет дослідження охоплює вивчення та застосування різних технологій для створення цієї гри. Це включає роботу з Unity як основного ігрового рушія, використання плагіна HappyFunTimes для забезпечення багатокористувацької взаємодії, а також розробку та інтеграцію ШІ для управління поведінкою NPC. Особлива увага приділяється розробці інтерфейсу, ігрової механіки та алгоритмів ШІ, які дозволяють гравцям плавно взаємодіяти в офлайн режимі.

Методи дослідження - застосування принципів ігрової механіки та дизайну для створення захоплюючого геймплею. Програмування: використання середовища розробки Unity та мови програмування C# для створення ігрових механік та інтеграції багатокористувацьких функцій; інтеграція плагіна HappyFunTimes для забезпечення синхронізації мобільних пристроїв з центральним хостом. Розробка штучного інтелекту: використання алгоритмів штучного інтелекту для створення реалістичної поведінки NPC; застосування принципів адаптивних систем для налаштування ШІ до різних ігрових ситуацій. Тестування: проведення функціонального тестування для виявлення та виправлення помилок у коді; Оцінка та оптимізація: аналіз зібраних даних тестування для оптимізації ігрового процесу та продуктивності гри.

РОЗДІЛ 1

ТЕОРЕТИЧНІ ОСНОВИ РОЗРОБКИ БАГАТОКОРИСТУВАЦЬКОЇ ГРИ ТА ШТУЧНОГО ІНТЕЛЕКТУ

1.1. Опис предметного середовища (функціональної моделі, процесу діяльності)

Unity Engine

Unity — це інтегроване середовище для розробки, яке дозволяє створювати 2D та 3D ігри, а також інші інтерактивні додатки, такі як симуляції та візуалізації. Платформа підтримує багато операційних систем та платформ, включаючи Windows, MacOS, Linux, Android, iOS, а також ігрові консолі та VR-системи.

Unity Engine був створений у Данії у 2004 році розробниками Девідом Гельгасоном, Йоакімом Анте та Ніколасом Франціcом. Спочатку Unity створювався як ігровий рушій для власного проекту, але швидко перетворився на універсальний інструмент, що відкрив можливості для інших розробників створювати та вдосконалювати ігри. Перша версія Unity була випущена у 2005 році і була спрямована на створення ігор для Mac OS X.

З того часу Unity зростав і розвивався, зокрема через швидкий розвиток мобільних технологій, що дозволило Unity стати одним з найпопулярніших рушіїв для мобільних ігор. Unity став відомим завдяки своїй мультиплатформенності, підтримуючи не лише мобільні платформи, але й більшість ігрових консолей та комп'ютерних систем.

Переваги Unity Engine:

- Легкість вивчення: Unity має інтуїтивно зрозумілий інтерфейс та велику спільноту користувачів, завдяки чому новачки можуть швидко розпочати роботу.
- Багато ресурсів для навчання: Існує безліч навчальних матеріалів, включаючи офіційні туторіали, відеокурси та книги.

- Велика спільнота: Спільнота розробників Unity надзвичайно активна та підтримує, що сприяє спільній роботі та обміну знаннями.
- Підтримка VR та AR: Unity є однією з провідних платформ для розробки віртуальної та доповненої реальності.

Недоліки Unity Engine:

- Проблеми з оптимізацією: Для дуже великих і складних проектів можуть виникнути проблеми з производительністю та оптимізацією.
- Ліцензування: Хоча Unity пропонує безкоштовну версію, повнофункціональні комерційні опції можуть бути дорогими, особливо для невеликих студій або індивідуальних розробників.
- Залежність від сторонніх асетів: Деякі розробники можуть занадто покладатися на асети з магазину, що може обмежити унікальність і індивідуальність ігор.

HappyFuntimes

HappyFuntimes — це плагін до Unity Engine, який розширює можливості створення ігор, спрямованих на мультиплеєрну взаємодію з використанням мобільних пристроїв як контролерів. Цей плагін дозволяє розробникам швидко і легко інтегрувати в ігри функціональність, при якій гравці можуть підключати свої смартфони або планшети до гри, що запущена на комп'ютері або консолі, через Wi-Fi або інтернет-з'єднання.

Переваги HappyFuntimes:

- Легкість інтеграції: HappyFuntimes розроблений так, щоб його було легко інтегрувати в існуючі проекти Unity без необхідності складного налаштування.
- Підтримка мультиплеєра: Плагін ідеально підходить для створення ігор на кілька гравців, особливо для великих груп, де кожен гравець може використовувати свій мобільний пристрій як унікальний контролер.

- Взаємодія з мобільними пристроями: HappyFuntimes дозволяє використовувати широкий спектр можливостей мобільних пристроїв, таких як сенсорні екрани, акселерометри і гіроскопи, для створення інноваційного ігрового досвіду.
- Інноваційність: Плагін відкриває нові можливості для творчих ідей в ігровому дизайні, дозволяючи розробникам створювати унікальні ігрові механіки.

Недоліки HappyFuntimes:

- Залежність від мережі: Щоб використовувати мобільні пристрої як контролери, необхідна надійна мережа Wi-Fi, що може бути проблематичним в деяких умовах.
- Обмеженість платформи: Поки що HappyFuntimes може мати обмежену підтримку на деяких платформах, і розробники можуть зіткнутися з проблемами сумісності в залежності від версії Unity або мобільної ОС.
- Складність тестування: Тестування ігор, які використовують мультиплеєр на багатьох мобільних пристроях, може бути логістично складним і часозатратним.
- Обмежені ресурси: Інформація та ресурси для розробки з HappyFuntimes можуть бути менш доступними в порівнянні з іншими більш традиційними ігровими розробками в Unity.

1.2. Огляд існуючих ігрових додатків за даною тематикою

“Jackbox Party Packs” - серія ігор для вечірок, де гравці підключаються до центрального екрана через мобільні пристрої. Ігри варіюються від вікторин до творчих завдань, що забезпечує веселий та інтерактивний досвід.

Рис. 1.1. Ігровий додаток “Jackbox Party Packs” [6]

“Keep Talking and Nobody Explodes” - кооперативна гра, де один гравець управляє бомбою через ПК, а інші гравці використовують мобільні пристрої для читання інструкцій з її знешкодження. Ідеальна для групових ігор і соціальної взаємодії.

Рис. 1.2. Ігровий додаток “Keep Talking and Nobody Explodes” *[7]*

1.3. Постановка задачі

Багато з існуючих ігор здаються простими і легкими для розуміння, проте часто вони втрачають свою привабливість через брак глибини та інтерактивності. Моя дипломна робота має на меті вирішити цю проблему, створивши ігровий застосунок, який не тільки легкий у використанні, але й пропонує багатогранний ігровий досвід.

Вирішенням цієї проблеми є реалізація кооперативного 2D офлайн ігрового застосунку, який залучає гравців до спільної гри з використанням їхніх мобільних пристроїв як контролерів. Основна концепція гри полягає в наступному:

1. Гравці об'єднуються в значну групу, розділяючись на дві команди - "Creatures" та "Goblins" - і підключаються до центрального хоста. Цей хост керує грою та транслює її на великий екран, де всі учасники можуть спостерігати за розвитком подій і стратегіями обох команд.
2. Гра має динамічну сюжетну лінію, де команда "Creatures" прагне захистити ключовий елемент гри – кладовище зі скарбами від нападів команди "Goblins". З іншого боку, "Goblins" намагаються досягти своєї мети – пограбувати кладовище, збираючи ящики з золотом.
3. Кожен гравець контролює свого персонажа через мобільний пристрій. Стратегічне планування та командна взаємодія є ключовими для успіху в грі.
4. Гра закінчується, коли одна з команд досягає своєї кінцевої мети: "Creatures" вдається зловити всіх "Goblins" і помістити їх у склеп, або "Goblins" успішно збирають всі ящики з золотом.
5. Штучний інтелект відіграє ключову роль у забезпеченні динамічної та реалістичної поведінки NPC (неігрових персонажів). ШІ дозволяє NPC адаптуватися до дій гравців, забезпечуючи непередбачуваність і виклики, що підтримують інтерес до гри. Завдяки ШІ, NPC можуть виконувати складні

стратегії, реагувати на зміну умов гри та взаємодіяти з гравцями, роблячи ігровий процес більш насиченим і цікавим.

1.3.1. Підхід до вирішення проблеми

Реалізація ігрового застосунку

Unity - це потужний і гнучкий крос-платформений ігровий рушій, який дозволяє розробникам створювати ігрові застосунки та інтерактивні досвіди. Його особливості включають підтримку 2D та 3D графіки, фізичний рушій, вбудовані інструменти для анімації та скриптингу на C#.

- **Розробка Ігрового Світу та Механіки**: Я почну з створення 2D ігрового світу в Unity, розробляючи ігрові рівні, персонажів та інтерактивні об'єкти. Завдяки інструментам, таким як Tilemap, я зможу детально налаштовувати ігрове середовище та створювати унікальні локації.
- **Анімація та Графіка**: Використовуючи Sprite Editor, я займусь створенням та анімацією персонажів і об'єктів гри. Unity дозволить мені імпортувати та оживляти асети, додаючи динамічність до геймплея.
- **Програмування Логіки Гри на C#**: Я реалізую ігрову логіку, управління персонажами та взаємодію об'єктів через скрипти на C#. Unity надає доступ до багатого API та інтеграції з .NET, що надає мені велику гнучкість в реалізації ігрових механік.

Робота з Фізикою: Я використаю фізичний рушій Unity для надання реалістичних фізичних властивостей об'єктам гри, таких як колізії та гравітація.

Реалізація Мультиплеєра з HappyFunTimes

Створення Мережевої Архітектури:

- Для реалізації мультиплеєра у моєму проекті, я обрав використання плагіну HappyFunTimes, оскільки він дозволяє створити гнучку мережеву архітектуру

з одним сервером (хостом) та численними клієнтами. Ця архітектура ідеально підходить для кооперативних ігор, де кожен гравець може підключатися до гри через свій мобільний пристрій як клієнт.

- Основною перевагою такої архітектури є можливість масштабування: система може обслуговувати значну кількість гравців без істотного зниження продуктивності. Також, це спрощує розподіл ресурсів і обробку даних, оскільки всі ігрові розрахунки централізовано обробляються на сервері.

Синхронізація Даних та Взаємодії Гравців:

- Сервер HappyFunTimes використовує WebSockets для забезпечення надійного, низькозатримкового з'єднання між сервером та клієнтами. WebSocket - це протокол, який дозволяє двосторонній обмін даними між клієнтом і сервером по відкритому з'єднанню. Це дозволяє отримувати вводи від гравців в реальному часі.
- Я імплементував систему синхронізації, що дозволяє одночасно обробляти вводи від усіх підключених гравців. Це включає обробку дій гравців, рухів персонажів та інших взаємодій у грі.

Реалізація Мережевої Логіки на Сервері:

- Усі ігрові механіки та правила обробляються на сервері. Це включає логіку стану гри, управління рахунками, а також розрахунок перемоги та поразок. Обробка всієї логіки на сервері знижує ризики несанкціонованого втручання та забезпечує однорідний ігровий процес.

Реалізація Штучного Інтелекту:

- Створення Інтелектуальних NPC: Використання алгоритмів ШІ для керування поведінкою неігрових персонажів (NPC) забезпечить динамічний та

реалістичний ігровий процес. NPC будуть здатні адаптуватися до дій гравців, реагувати на зміни в ігровому середовищі та виконувати складні стратегії.

- Алгоритми Навігації: Реалізація алгоритмів навігації, таких як A*, дозволить NPC ефективно знаходити та обирати оптимальні маршрути в ігровому середовищі, обходячи перешкоди та досягаючи своїх цілей.
- Адаптивна Поведінка: Використання машинного навчання та інших технік ШІ для забезпечення адаптивної поведінки NPC. Це дозволить NPC вчитися на основі взаємодії з гравцями, роблячи їхні дії більш передбачуваними та реалістичними.

РОЗДІЛ 2

ПРОЕКТУВАННЯ ІГРОВОГО ЗАСТОСУНКУ

2.1. Технічне завдання

- Розробка ігрового поля.
- Створення механіки руху для гравців.
- Розробка взаємодії з ігровими об'єктами.
- Впровадження системи балансу та умов програшу.
- Імплементація мультиплеєрної механіки через плагін HappyFunTimes.
- Розробка штучного інтелекту

2.2. Опис основного задуму

Перед початком гри, гравці у випадковому порядку поділяються на дві команди – “Creatures” та “Goblins”. Ціль команди “Creatures” зловити всіх гоблінів та відправити їх до склепу. Ціль команди “Goblins” зібрати всі ящики із золотом до того, як їх зловить команда противника. Гра завершується тоді, коли або всіх гоблінів було зловлено та вони знаходяться у склепі (перемога команди “Creatures), або всі ящики із золотом були зібрані (перемога команди “Goblins”).

2.3. Опис ігрових механік:

- рух гравців: гравці мають здатність переміщатися в усі чотири напрямки у двовимірному ігровому світі. Ця механіка є ключовою для навігації по ігровому полю та виконання завдань;

- захоплення та ліквідація: під час зустрічі гравця з команди “Creatures” з гравцем команди “Goblins” відбувається автоматичне захоплення гобліна і його

переміщення до склепу. Водночас, гравці з команди “Goblins” збирають ящики з золотом, торкаючись їх на ігровому полі;

- набір очок: очки нараховуються за зібрані ящики золота та за кожного гобліна, який не перебуває у склепі. Кількість очок динамічно відображається на ігровому екрані, дозволяючи гравцям відстежувати прогрес та стан гри;

- статистика гравців: після завершення гри, статистика кожного гравця, включаючи кількість зловлених гоблінів чи зібраних ящиків золота, підводиться та відображається. На підставі цієї статистики визначається найкращий гравець переможної команди.

- взаємодія з об’єктами гри: у грі присутні численні інтерактивні об'єкти, кожен з яких має свої унікальні функції та способи взаємодії з ними. Ці об'єкти значно урізноманітнюють геймплей та додають додаткові стратегічні елементи у гру. Детальний опис цих об'єктів та їхніх функцій буде наведено в розділі “Опис структури проекту та функціоналу”.

2.4. Опис структури проєкту та функціоналу

Проект передбачає створення багатокористувацької офлайн гри, яка дозволяє гравцям зануритися в ігровий світ через свої мобільні пристрої. Основою для цього є один центральний пристрій-хост, який запускає і відображає головну сцену гри. Ця сцена може бути трансльована на більший екран, такий як телевізор, щоб всі учасники могли її бачити.

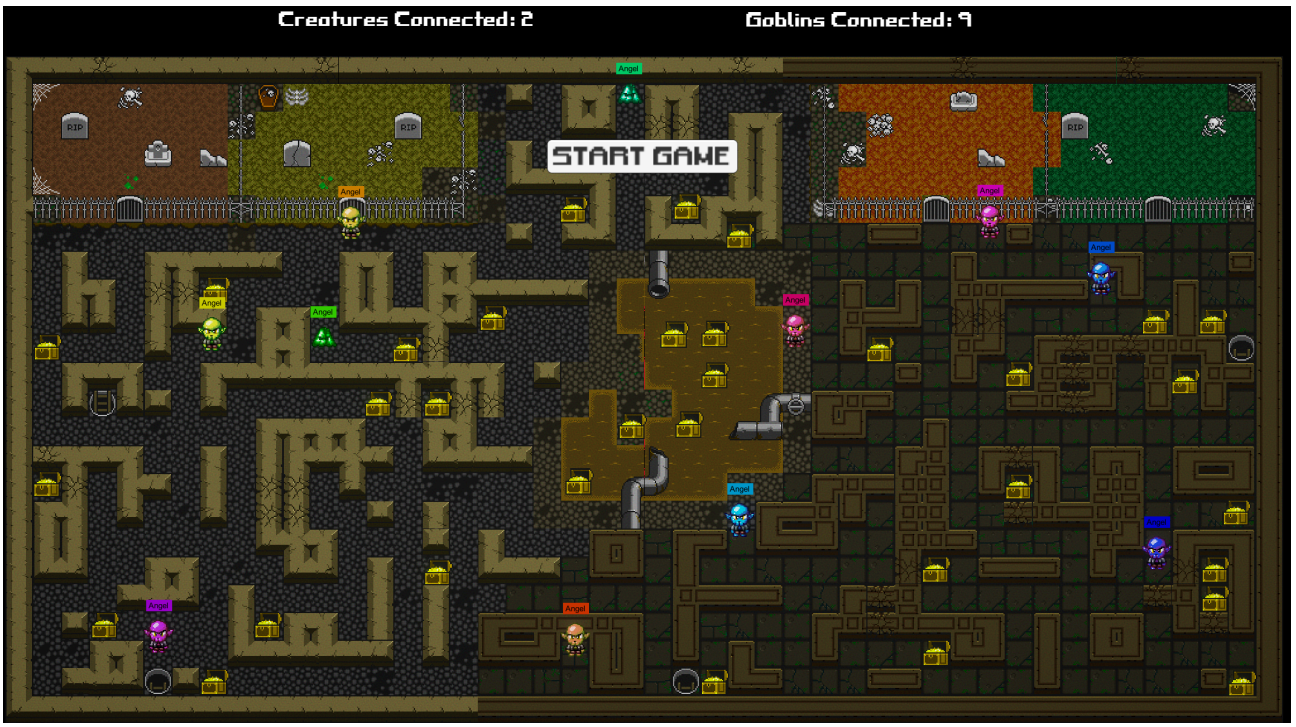

Рис. 2.1. головна сцена створена за допомогою функціоналу Tile Pallete.

Джерело: автор

Щоб підключитися до гри, мобільні пристрої гравців мають бути підключені до тієї ж мережі, до якої підключено хост-пристрій. Гравці входять на localhost або happyfuntimes.net зі своїх пристроїв, де кожен з них може вибрати ім'я для свого персонажа. Після вибору імені на екрані з'являється ігровий контролер, який дозволяє управляти персонажем. На головному екрані кожен персонаж відображається з обраним ім'ям та кольором, який відповідає кольору контролера гравця.

Рис. 2.2. контролер гравця (фунціонал контролеру та приєднання гравців до гри здійснено за допомогою плагіну від [happyfuntimes])

Джерело: автор

Після підключення гравців до гри, вони розподіляються випадковим чином між двома командами - гоблінами та створіннями, з відношенням 1:3 для збалансування геймплею. Мінімальна кількість гравців для початку гри становить чотири. До моменту, коли хост розпочинає гру, вона знаходиться у стані паузи, дозволяючи гравцям ознайомитися зі своїми персонажами, але не дозволяючи виконувати будь-які дії. Гра починається з натисканням хостом кнопки “Start Game”, після чого всі учасники мають можливість активно взаємодіяти в ігровому процесі.

2.5. Ігрові предмети

Предмети підбору:

Ці предмети надають гравцям особливі можливості або переваги у грі. Вони з'являються випадковим чином на ігровому полі на певних локаціях:

- «еліксир швидкості» – при підбиранні цього предмету гравець отримує тимчасове збільшення швидкості руху, що може бути вирішальним фактором під час переслідувань або уникнення зіткнень;
- «лопата» – надає гравцеві можливість прокопувати тунелі крізь тріснуті стіни, тим самим відкриваючи нові шляхи та скорочуючи маршрути;
- «ключ» – дозволяє гравцеві відчиняти чи зачиняти двері до склепу, що може бути використано стратегічно для блокування або відкриття шляху для гоблінів;

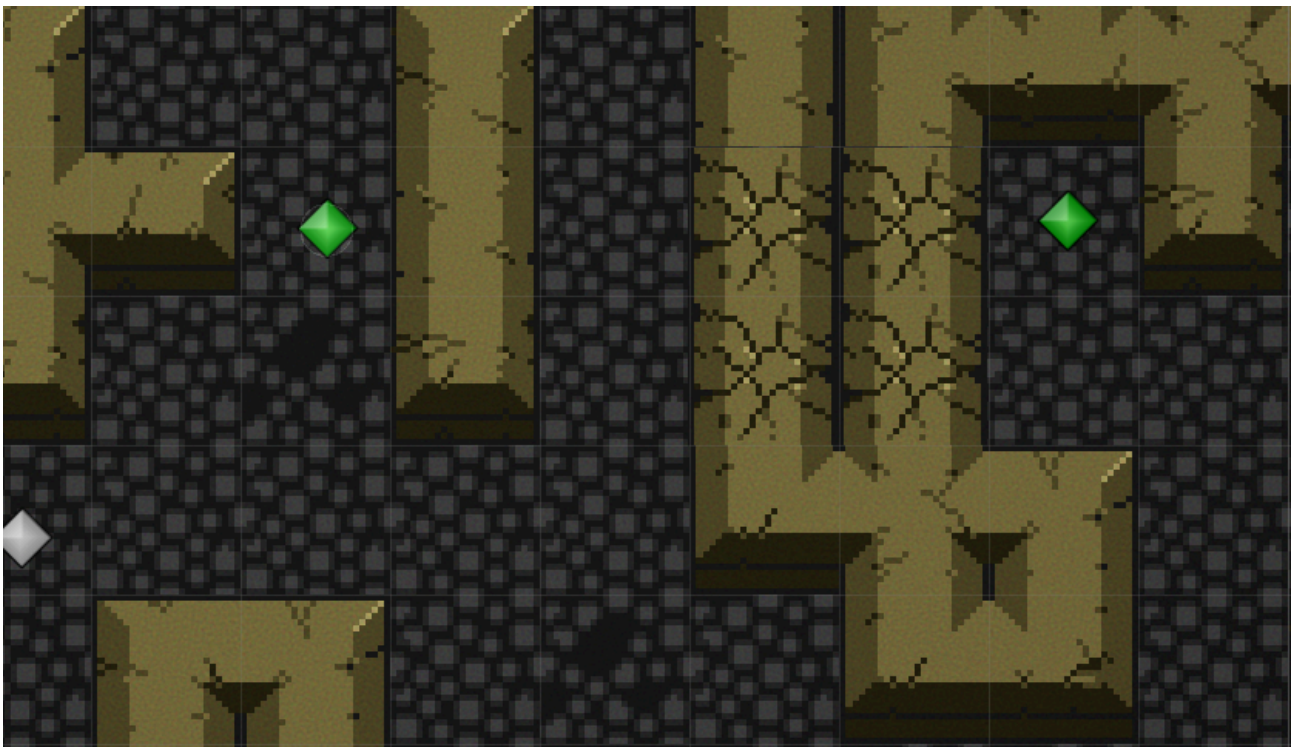

Рис. 2.3. точки для розподілу ігрових предметів збору

Джерело: автор

Предмети навколишнього середовища:

-  «тріснута стіна» – цей об'єкт може бути знищений гравцем, який підібрав «лопату». Це створює нові тактичні можливості, дозволяючи гравцям відкривати нові шляхи або короткі маршрути для швидкого переміщення по мапі;
-  «болото» – розташоване в центрі ігрової мапи, болото сповільнює швидкість пересування гравців, що проходять через нього. Це додає додатковий вимір стратегії та обережності при плануванні маршрутів;
-  «каналізація» – розміщені на мапі, ці труби діють як портали, швидко переміщаючи гравця з одного кінця на інший. Це може бути використано для уникнення противників або досягнення важливих точок на мапі швидше;

-  «двері склепу» – цей елемент дозволяє гравцям з «ключем» відкривати або закривати двері склепу. Це може змінити хід гри, дозволяючи зловленим гоблінам втекти або залишаючи їх заблокованими всередині.

2.6. Штучний інтелект

Штучний інтелект (ШІ) став невід'ємною частиною сучасної ігрової індустрії, особливо в контексті розробки 2D ігор на платформі Unity. Використання передових алгоритмів ШІ не тільки підвищує реалізм ігор, але й значно збагачує ігровий досвід, створюючи непередбачувані та динамічні сценарії, що відповідають на дії гравців. Центральну роль у таких технологіях відіграють алгоритми пошуку шляху, які дозволяють ігровим персонажем ефективно навігувати у складних ігрових світах, оптимізуючі свої маршрути та обходячи перешкоди.

У цій роботі ми також розглянемо популярні алгоритми пошуку шляху, такі як Dijkstra, Breadth-First Search (BFS) та A*, з метою порівняння їх ефективності в різних ігрових сценаріях. Завдяки детальному аналізу цих алгоритмів, ця робота має на меті виявити найбільш оптимальні стратегії для розробки ШІ у 2D іграх, що відкриває нові можливості для подальшого підвищення якості ігрового процесу.

2.6.1. Опис проблеми

Однією з основних задач у розробці штучного інтелекту для ігор є пошук оптимального шляху та уникнення перешкод. Ця задача особливо важлива в динамічних ігрових середовищах, де ігрові персонажі повинні швидко і ефективно адаптуватися до змін у навколишньому світі.

Уявімо сценарій, де об’єкт (ігровий персонаж) знаходиться у нижній частині карти і прагне досягти верхньої. Під час руху об’єкт сканує навколишню територію, яка виділена рожевим кольором на рис. 1.8. На цьому етапі він не виявляє жодних

перешкод, тому продовжує рухатися вгору. Однак, наближаючись до вершини, об’єкт натрапляє на несподівану перешкоду. Виявивши перешкоду, об’єкт змінює напрямок і починає шукати обхідний шлях у формі "U", слідуючи червоному маршруту, який є довшим і менш оптимальним.

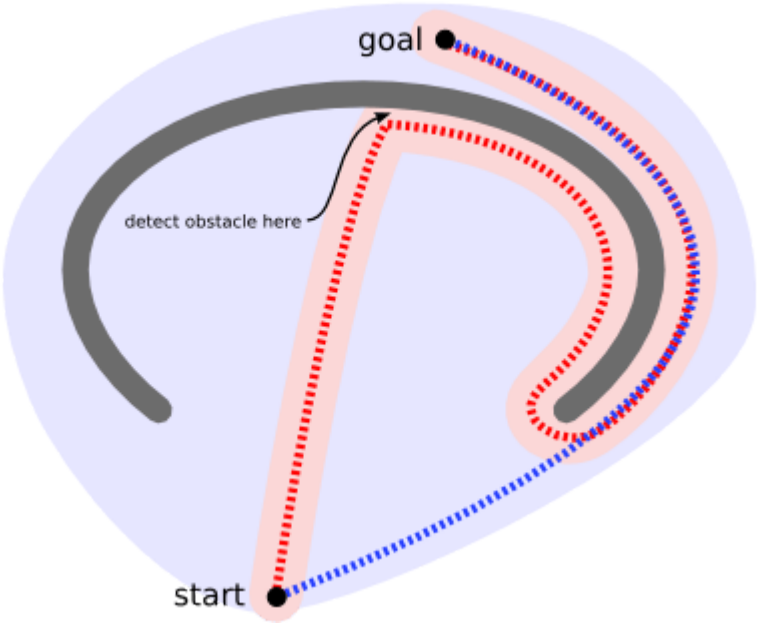

Рис. 2.4. Не оптимальний пошук шляху [8]

Оптимальний алгоритм пошуку шляху діяв би інакше. Замість того, щоб просто реагувати на перешкоди під час руху, він би заздалегідь сканував ширшу територію, показану жовтим кольором, і обрав би коротший та більш ефективний шлях, позначений синім. Такий підхід дозволяє уникнути прямого зіткнення з перешкодами і значно скорочує час досягнення мети.

Основна перевага алгоритмів пошуку шляху полягає в їх здатності планувати маршрут заздалегідь, а не лише реагувати на перешкоди в останній момент. Проте, між плануванням маршруту і адаптивними алгоритмами руху існує компроміс. Планування зазвичай вимагає більше часу на обчислення, але дає кращі результати у вигляді оптимальних маршрутів. Натомість, реактивний рух є швидшим у виконанні, але може призвести до неефективних або тупикових ситуацій.

У світах, які постійно змінюються, важливість планування на велику дистанцію зменшується. В таких випадках доцільно використовувати обидва підходи: алгоритми пошуку шляхів для загального планування, де зміни в розміщенні перешкод відбуваються повільно і на великих дистанціях; і адаптивний рух для швидких змін і коротких відстаней.

Застосування цих стратегій у поєднанні дозволяє створити більш ефективний і гнучкий штучний інтелект, здатний справлятися з широким спектром ігрових ситуацій, забезпечуючи гравцям захоплюючий і реалістичний ігровий досвід.

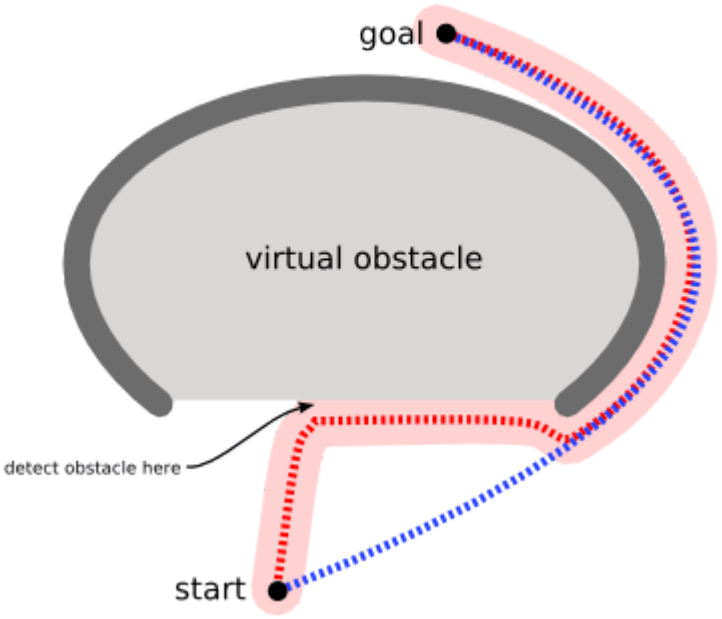

Рис. 2.5. Оптимальний пошук шляху [8]

2.6.2. Порівняння алгоритмів пошуку шляху

Алгоритм Дейкстри

Алгоритм Дейкстри — це класичний метод знаходження найкоротшого шляху в графі, де кожне ребро має визначену вагу, що представляє відстань або вартість пересування між вузлами. Цей алгоритм підходить для ситуацій, коли необхідно знайти оптимальний шлях від однієї точки до іншої.

Уявімо, що ви шукаєте найшвидший шлях додому з роботи. Починаючи з вашого офісу (рожевий квадрат на мапі), алгоритм перевіряє всі близькі місця, куди можна піти далі. На кожному кроці він вибирає місце, яке має найменшу вартість пересування на даний момент. Цей процес триває, доки алгоритм не досягне кінцевої точки — вашого дому (блакитний квадрат).

Кожен раз, коли алгоритм проходить через вузол, він "фарбує" його в блакитний колір на мапі. Світліші відтінки вказують на вузли, які були досліджені пізніше. Це візуально демонструє, як алгоритм поступово поширюється від стартової точки до мети, подібно до того, як вода розливається по поверхні. У кінцевому підсумку алгоритм Дейкстри знаходить найкоротший шлях додому, гарантуючи, що він буде найшвидшим за умови, що всі вулиці відкриті та немає раптових змін у дорожній мережі.

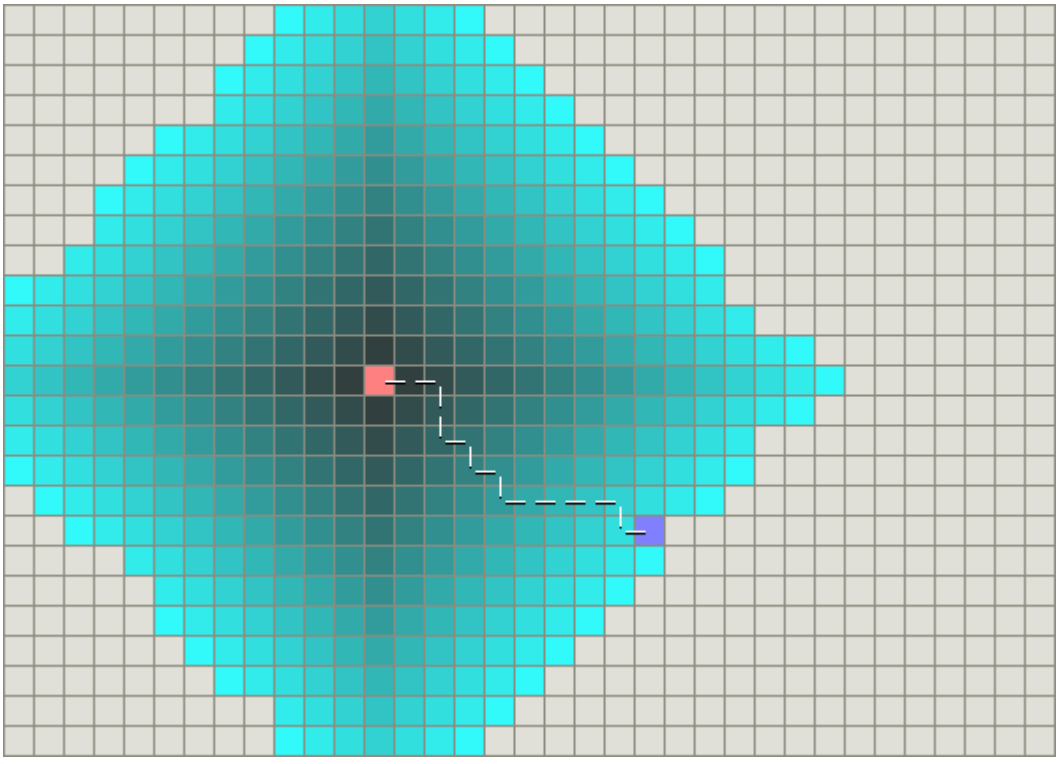

Рис. 2.6. Робота алгоритму Дейкстри [8]

Жадібний алгоритм пошуку шляху (Greedy Best-First Search)

Жадібний алгоритм пошуку шляху (Greedy Best-First Search) — це метод, який намагається знайти найшвидший маршрут до цілі, використовуючи спеціальну оцінку — евристику. Евристика допомагає алгоритму робити "освічені здогадки" щодо того, в якому напрямку рухатися, щоб якнайшвидше наблизитися до мети.

Наприклад, уявімо, що ми шукаємо шлях у лабіринті, і знаємо, що вихід знаходиться на південь. Алгоритм буде в першу чергу вибирати ті шляхи, які ведуть на південь, базуючись на евристичній оцінці відстані до мети. Жовтий колір на діаграмі може представляти точки, які є далеко від цілі за оцінкою алгоритму, тобто, здається, що їх важко досягти. Натомість чорний колір представляє точки, які здаються близькими до цілі, тобто, до них легко дістатися. Жадібний алгоритм пошуку буде рухатися до чорних точок, ігноруючи жовті, оскільки він припускає, що чорні точки приведуть його до цілі швидше.

Основна перевага цього методу полягає у його швидкості. Він може не завжди знаходити найкоротший можливий шлях, але часто знаходить досить хороший шлях за набагато коротший час, ніж більш ретельні методи, такі як алгоритм Дейкстри. Це особливо корисно в ситуаціях, де час обчислення має вирішальне значення, і потрібно швидко отримати прийнятне рішення, навіть якщо воно не є абсолютно оптимальним.

На прикладі лабіринту, якщо ми знаємо, що вихід розташований на південь, жадібний алгоритм буде пріоритетно перевіряти всі можливі південні маршрути. Якщо він натрапить на перешкоду, алгоритм швидко обере наступний найкращий маршрут згідно з евристикою, що наблизить його до мети. Врешті-решт, він досягне виходу набагато швидше, ніж якби використовувався алгоритм Дейкстри, який ретельно оцінює всі можливі шляхи без виключень.

Жадібний алгоритм пошуку є ефективним і швидким інструментом, особливо в тих випадках, де швидкість є важливішою за абсолютну точність маршруту. Він

знаходить прийнятні шляхи, що відповідають заданій евристиці, ідеально підходячи для динамічних ігрових середовищ, де необхідно швидко реагувати на зміни.

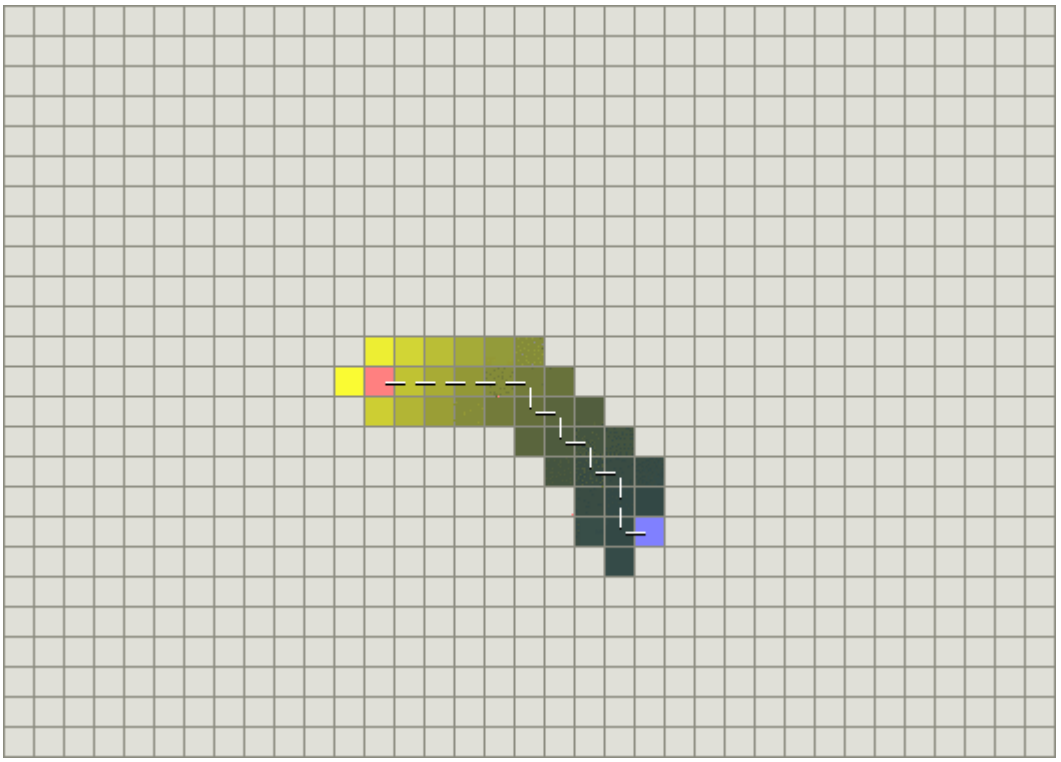

Рис. 2.7. Робота жадібного алгоритму [8]

Обидва ці приклади демонструють найпростіший випадок, коли на карті немає перешкод і найкоротший шлях є прямою лінією. Розглянемо складніший сценарій з увігнутою перешкодою, як описано в попередньому розділі. Алгоритм Дейкстри виконує більше обчислень, але гарантовано знаходить найкоротший шлях.

З іншого боку, Greedy Best-First-Search виконує менше роботи, але його шлях часто є не оптимальним. Проблема полягає в тому, що цей алгоритм "жадібний" і намагається рухатися безпосередньо до мети, навіть якщо це не є правильним шляхом. Оскільки він враховує лише вартість досягнення мети, ігноруючи вартість пройденого шляху до цього моменту, він продовжує рух навіть тоді, коли обраний шлях стає занадто довгим.

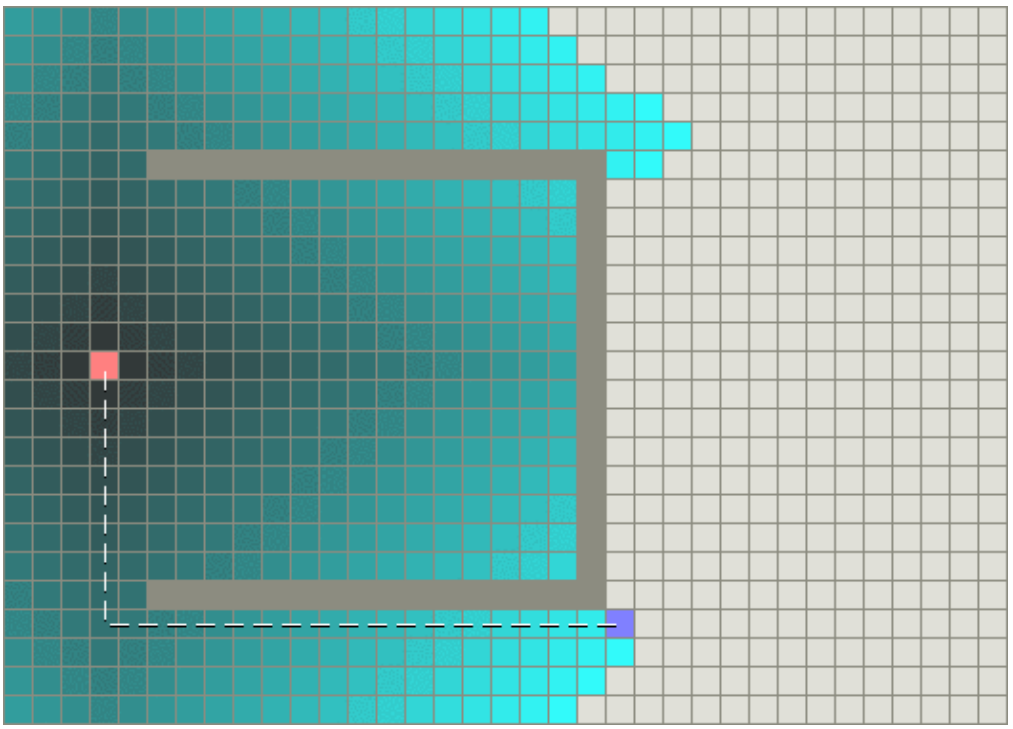

Рис. 2.8 Обминання перешкод алгоритмом Дейкста [8]

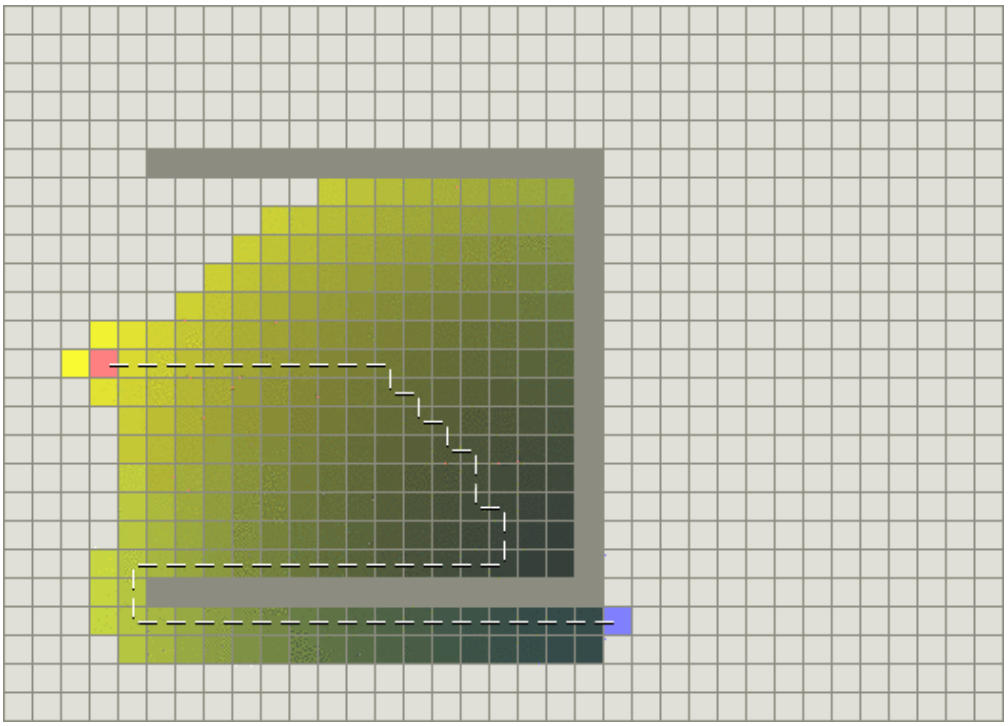

Рис. 2.9. Оминання перешкод алгоритмом Жадібним алгоритмом [8]

Алгоритм A*

Алгоритм A* схожий на алгоритм Дейкстри тим, що його можна використовувати для пошуку найкоротшого шляху. Однак, як і Greedy Best-First-Search, A* може використовувати евристику для направлення пошуку. У простих випадках він може працювати так само швидко, як Greedy Best-First-Search.

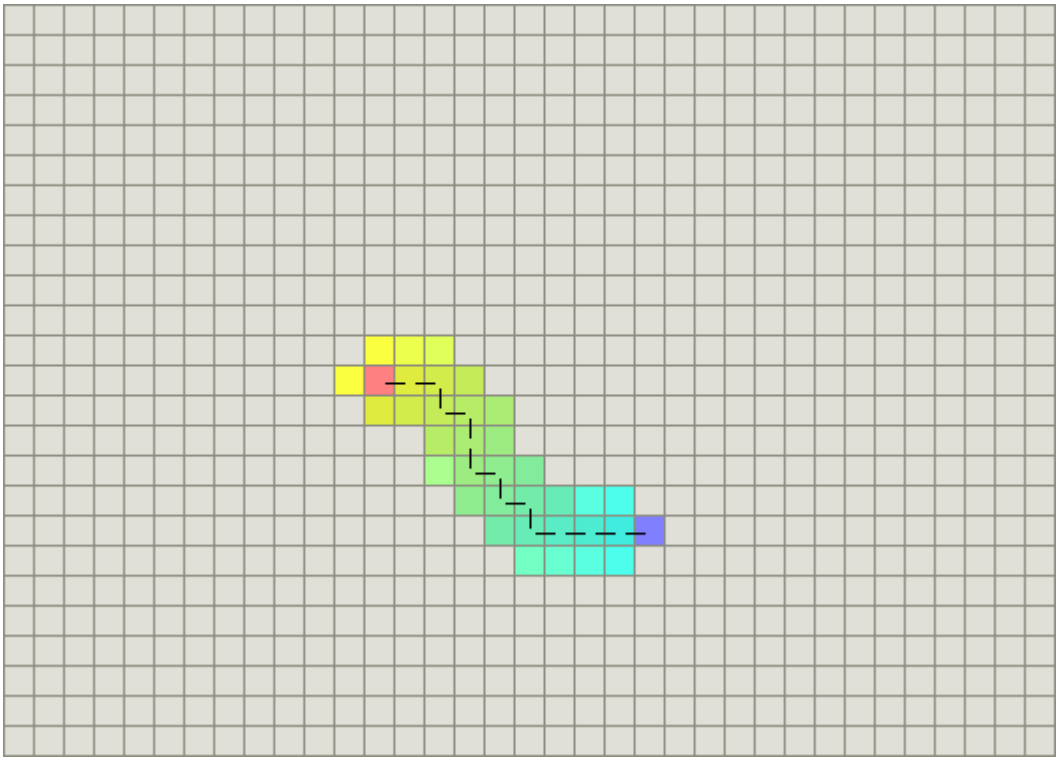

Рис. 2.11. Аналіз вузлів алгоритмом A* [8]

У прикладі з увігнутою перешкодою алгоритм A* знаходить такий же оптимальний шлях, як і алгоритм Дейкстри. Секрет успіху A* полягає в тому, що він поєднує два підходи: інформацію, яку використовує алгоритм Дейкстри (віддаючи перевагу вузлам, близьким до початкової точки), та інформацію, яку використовує Greedy Best-First-Search (віддаючи перевагу вузлам, близьким до мети). Це дозволяє A* ефективно балансувати між пошуком найкоротшого шляху та швидкістю обчислень, роблячи його потужним інструментом для навігації в складних ігрових середовищах.

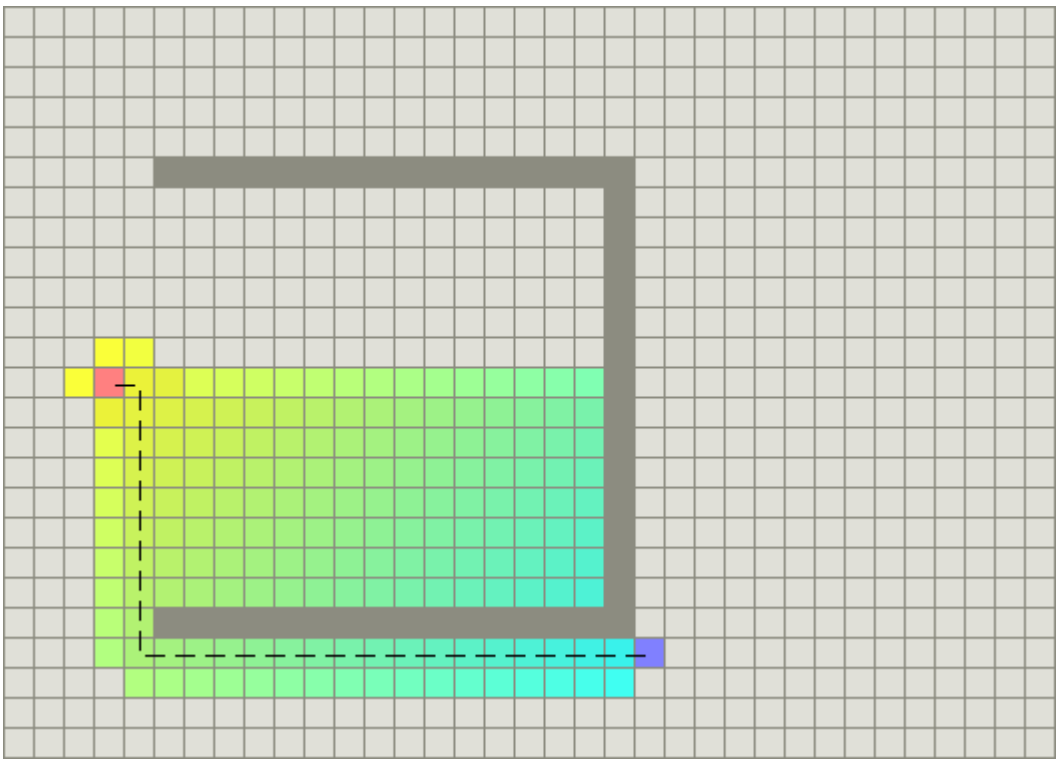

Рис. 2.12. Оминання перешкоди алгоритмом A* [8]

2.6.3. Принцип роботи алгоритму A*

Основні компоненти:

- **Вузли (Nodes)**: Основні елементи, через які проходить шлях. Кожен вузол представляє точку на графі і може мати декілька сусідніх вузлів, до яких він підключений.
- **Вартість шляху від початкової точки до поточного вузла (G cost або Actual cost)**: Це фактична вартість пересування від початкового вузла до поточного вузла. Вона включає всі витрати, пов'язані з проходженням цього шляху.
- **Евристична оцінка вартості шляху від поточного вузла до кінцевої точки (H cost або Heuristic cost)**: Це передбачувана вартість пересування від поточного вузла до кінцевої точки. Евристика допомагає алгоритму "передбачити" відстань до мети, що дозволяє оптимізувати пошук.

- **Загальна оцінка вартості вузла (F cost)**: Це сума G cost і H cost, тобто F = G + H. Ця величина використовується для визначення пріоритету обробки вузлів у черзі пошуку. Алгоритм обирає вузли з найменшим значенням F cost для подальшої обробки, що дозволяє ефективніше досягати кінцевої точки.

Принцип роботи алгоритму

A* алгоритм починає з початкового вузла і визначає для сусідніх вузлів їхні вартості F, які є сумою вартості G до вузла та евристичної вартості H від вузла до цілі. Ці сусідні вузли додаються до списку відкритих вузлів. Алгоритм обирає вузол з найменшою F вартістю з відкритого списку для подальшого розгляду та переміщує його до закритого списку. Цей процес триває доти, доки кінцевий вузол не опиниться в закритому списку, що означає, що шлях знайдений, або поки відкритий список не стане порожнім, що означає відсутність шляху. Після знаходження шляху можна прослідкувати назад від кінцевого вузла до початкового, використовуючи батьківські посилання кожного вузла.

Вибір евристичної функції є ключовим для ефективності алгоритму A* Найпоширеніші евристики включають:

- Евклідова відстань: Це пряма лінія від поточного вузла до цілі, яка використовується в ситуаціях, де можливий рух по діагоналі. Обчислюється як сума квадратів різниць координат.
- Манхеттенська відстань: Це сума абсолютних різниць між координатами поточної точки і цілі по осях X та Y. Використовується, коли рух можливий тільки вертикально та горизонтально. Відстань між двома точками дорівнює сумі модулів різниць їх координат.

Ефективність і швидкість алгоритму A* значною мірою залежать від правильного вибору евристики, оптимізації роботи зі списками відкритих та закритих вузлів, а також від складності карти та розміщення перешкод.

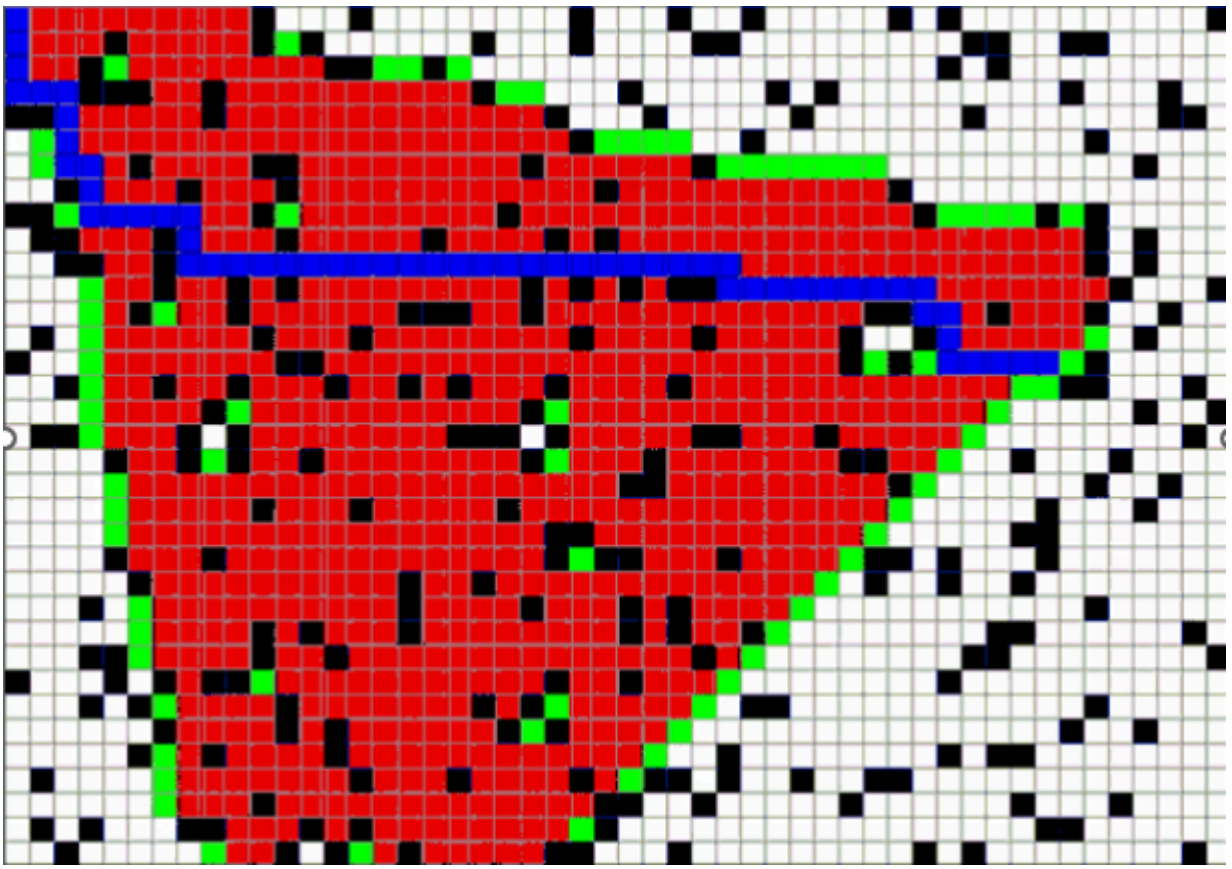

Рис. 2.13. Пошук найкоротшого маршруту алгоритмом A* [9]

РОЗДІЛ 3

РЕАЛІЗАЦІЯ ІГРОВОГО ЗАСТОСУНКУ

3.1. Розробка ігрового поля

У рамках мого курсового проєкту, я розробив 2D мапу в Unity, використовуючи інструмент Tile Palette, що значно спрощує процес створення складних ігрових рівнів.

Процес розробки:

1. Підготовка та Імпортування Тайлів:
    - Спочатку я підготував та імпортував набір тайлів, які я буду використовувати для створення мапи. Ці тайли включають різноманітні текстури та елементи ландшафту, як-от трава, дороги, вода, тощо.
    - Я імпортував ці тайли до Unity та налаштував їх як "Sprite" типу "2D and UI".
2. Створення Tile Palette:
    - У Unity я відкрив вкладку "Window", вибрав "2D" -> "Tile Palette" та створив нову палітру тайлів, давши їй відповідну назву.
    - Після цього я перетягнув імпортовані тайли у Tile Palette.
3. Створення Tilemap:
    - У ієрархії проекту я створив новий GameObject, обравши "2D Object" -> "Tilemap", що створило новий шар Tilemap для моєї мапи.
    - Я створив кілька шарів Tilemap для різних елементів дизайну мапи.
4. Дизайн Мапи:

- Використовуючи інструменти кисті Tile Palette, я почав "малювати" мапу, розміщуючи тайли у потрібних місцях. Це було інтуїтивно та дозволило мені швидко створити основний ландшафт мапи.
- Я також використовував інструменти гумки та вибору для коригування та деталізації мапи.

5. Фіналізація Мапи:

- Після завершення основного дизайну, я додав додаткові елементи, такі як інтерактивні об'єкти та декоративні елементи.
- Оптимізація мапи була ключовим етапом, щоб забезпечити її ефективне виконання та відповідність ігровому процесу.

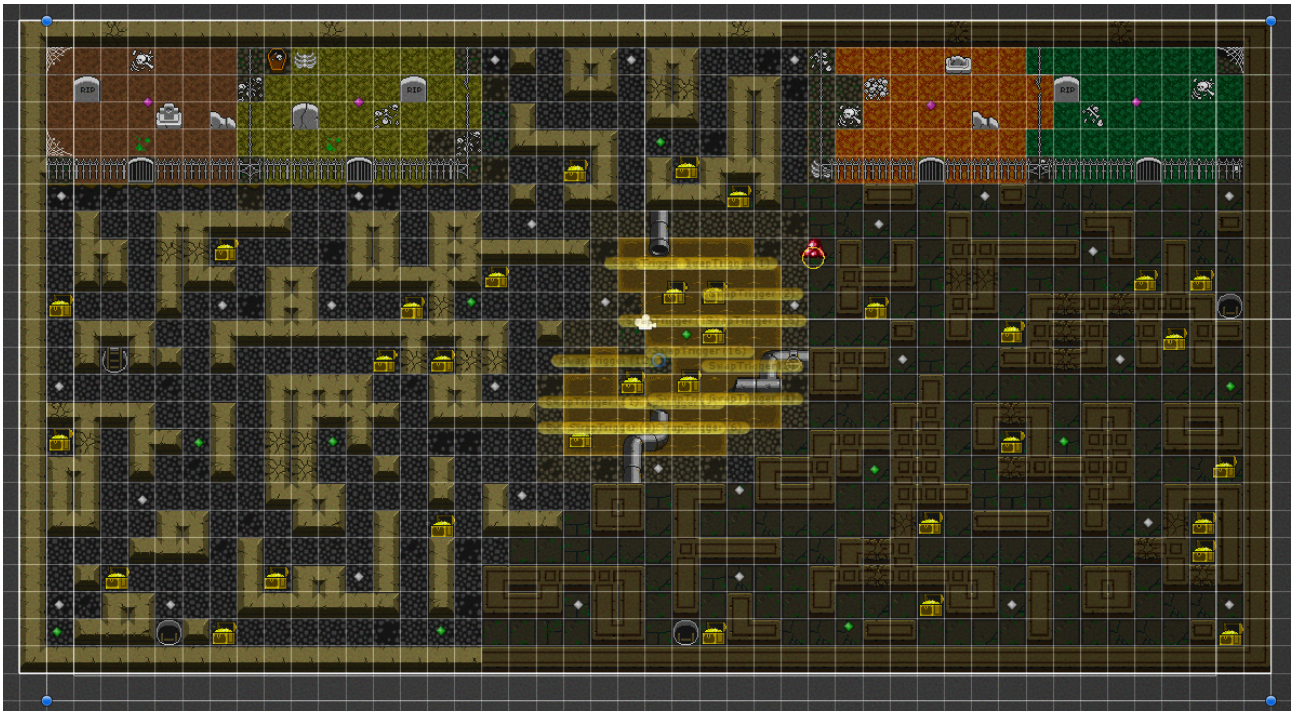

Рис. 3.1. вигляд готового ігрового поля в Scene View

Джерело: автор

3.2. Створення механіки руху гравців

При розробці механіки руху гравців, я зосередився на створенні плавного та інтуїтивно зрозумілого управління персонажами. Ось як я підійшов до цього процесу:

1. **Ініціалізація компонентів**:

   Спочатку я налаштував основні компоненти для мого персонажа, включаючи фізичний компонент (Rigidbody2D для 2D руху) і компонент анімації.

   Також я визначив змінні для керування швидкістю руху та іншими ключовими параметрами руху персонажа.

2. **Обробка вводу для руху**:

   Я розробив систему вводу, яка дозволяє гравцям керувати персонажем за допомогою клавіш на клавіатурі або через мобільний ввід. Ця система зчитує горизонтальний та вертикальний вводи для визначення напрямку руху персонажа.

3. **Застосування руху**:

   На основі отриманих даних вводу, я програмував логіку руху персонажа, використовуючи фізичний компонент для застосування сил та зміни напрямку руху.

4. **Візуальні та інтерактивні елементи**:

   Я також включив візуальні елементи, такі як зміна кольору персонажа та відображення імені гравця, щоб зробити ігровий процес більш динамічним та особистісним.

Використовуючи ці підходи, я зміг створити ефективну та відповідну механіку руху для моїх персонажів у грі, що забезпечує плавний ігровий досвід та зберігає високий рівень інтерактивності та контролю для гравців.

3.3. Керування станом гри та UI

Game Manager - це регулюючий центр гри. Цей клас відповідав за відстеження стану гри, кількості гравців, зберігання рахунків, а також за контроль над візуальними та аудіо елементами гри. Ось детальний опис його функціональності:

1. **Ініціалізація singleton інстансу**:

   Я забезпечив, що в грі існує лише один екземпляр **GameManager**, використовуючи шаблон Singleton. Це дозволяло легко отримувати доступ до цього класу з будь-якої частини гри.

2. **Відстеження стану гри**:

   Я використовував статичні змінні для відстеження основних параметрів гри, таких як кількість зібраних предметів (moneyBags), кількість гравців у кожній команді (goblins та creatures), та статус гри (чи гра розпочалась, чи завершилась).

3. **Управління ігровими полотнами (Canvas)**:

   Я використовував різні ігрові полотна для управління різними стадіями гри - стартове полотно, основне полотно гри та полотно перемоги.

4. **Оновлення ігрових рахунків**:

   У методі **Update()**, я оновлював тексти, які відображають поточний стан гри, наприклад кількість залишених "скарбів" або гравців. Це забезпечувало гравцям постійне оновлення інформації про стан гри.

5. **Перемога та поразка**:

   Я реалізував логіку для визначення перемоги однієї з команд. Якщо умови перемоги для однієї з команд були досягнуті (всі предмети зібрані або всі

супротивники піймані), гра завершувалась, відображалось відповідне повідомлення про перемогу та активувалось відповідне полотно.

6. **Відображення MVP (Most Valuable Player)**:

   Після завершення гри я визначав MVP для кожної команди на основі їхніх досягнень у грі. Це додавало елемент конкуренції та визнання до ігрового досвіду.

7. **Камера та аудіо**:

   Управління камерою та аудіо було також інтегровано в цей клас. Я налаштував камеру для фокусування на MVP після завершення гри та активував аудіо відтворення, коли гра була розпочата.

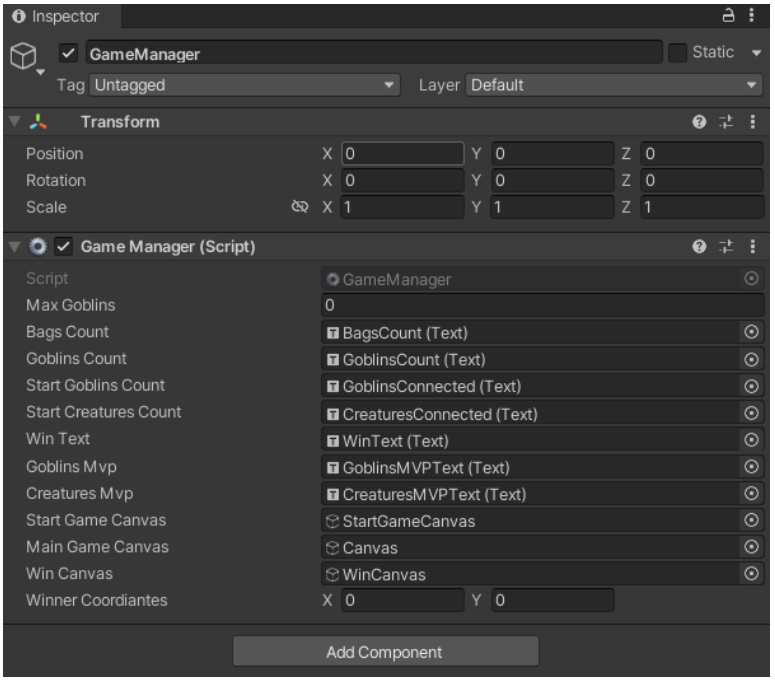

Рис. 3.2. Game Manager в Unity Inspector

Джерело: автор

3.4. Імплементація мультиплеєрної механіки

Для мого кооперативного проєкту я обрав архітектуру "один сервер – багато клієнтів", оскільки вона ідеально відповідала потребам геймплея. Вся ігрова логіка була централізована на сервері, в той час як клієнти, використовуючи свої пристрої, лише надсилали вводи для керування своїми персонажами. Плагін HappyFunTimes виявився ідеальним рішенням для реалізації цієї мультиплеєрної механіки, оскільки він ефективно підтримує велику кількість одночасних користувачів і забезпечує стабільний обмін даними між сервером та клієнтами через WebSockets.

Клас **GameServer** відіграє ключову роль у цьому процесі, управляючи зв'язками між сервером та клієнтами, обробляючи комунікацію через WebSockets.

1. **Ініціалізація сервера**:

   Клас **GameServer** ініціалізується з налаштуваннями, що визначають параметри підключення та обробки подій. Це включає налаштування WebSocket та обробники подій для різних типів повідомлень.

2. **Підключення та протокол WebSockets**:

   GameServer встановлює з'єднання з клієнтами через WebSocket, забезпечуючи двосторонній обмін даними. Це дозволяє серверу отримувати та відправляти повідомлення в реальному часі.

3. **Обробка повідомлень та команд**:

   Сервер обробляє різні типи повідомлень та команд від клієнтів, використовуючи обробники подій. Це включає повідомлення про підключення гравців, оновлення стану та системні команди.

4. **Управління гравцями**:

   GameServer керує підключеннями гравців, стежить за їхнім статусом та обробляє їх дії в грі. Це включає реєстрацію нових гравців та обробку від'єднань.

5. **Відправлення та отримання команд**:

   GameServer використовує різні методи для відправлення команд до клієнтів та отримання відповідей. Це включає механізми для масового розсилання повідомлень всім гравцям або специфічним гравцям.

6. **Обробка специфічних ситуацій**:

   Клас містить логіку для обробки специфічних ситуацій, таких як втрата з'єднання, помилки та логування подій.

3.5. Реалізація ігрових об'єктів

Префаби в Unity - це могутній інструмент, який дозволяє розробникам створювати, налаштовувати та зберігати складні ігрові об'єкти з їх компонентами та властивостями для повторного використання. Вони функціонують як шаблони або блакитні копії, з яких можна створювати нові екземпляри у сцені. Кожен префаб зберігає інформацію про компоненти об'єкта, такі як трансформації, візуалізатори, колайдери, скрипти та будь-які інші налаштування.

Коли існує потреба використовувати однаково налаштовані об'єкти у багатьох місцях сцени або в різних сценах проекту, перетворення їх в префаби є оптимальним рішенням. Це забезпечує централізоване керування цими об'єктами: будь-які зміни, внесені в основний префаб, автоматично відображаються на всіх його екземплярах. Це означає, що замість того, щоб редагувати кожен об'єкт індивідуально, ви можете швидко оновити всі копії, редагуючи лише префаб.

Крім того, префаби можуть містити дочірні об'єкти, що дозволяє створювати складні ієрархії всередині одного префаба. Це особливо корисно для створення складних інтерактивних об'єктів або персонажів, що містять багато різних компонентів та функціональностей. Таким чином, префаби значно спрощують управління ресурсами та підвищують ефективність розробки в Unity.

У проекті використано ряд префабів, серед яких префаб гравця команди гоблінів служить яскравим прикладом. Він містить компоненти, що визначають його функціональність та поведінку

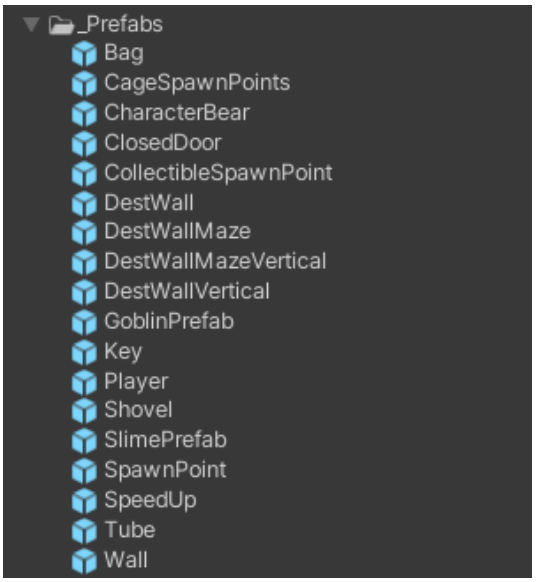

Рис. 3.3. Список префабів використаних у проекті

Джерело: автор

Для прикладу наведено префаб гравця з команди гоблінів.

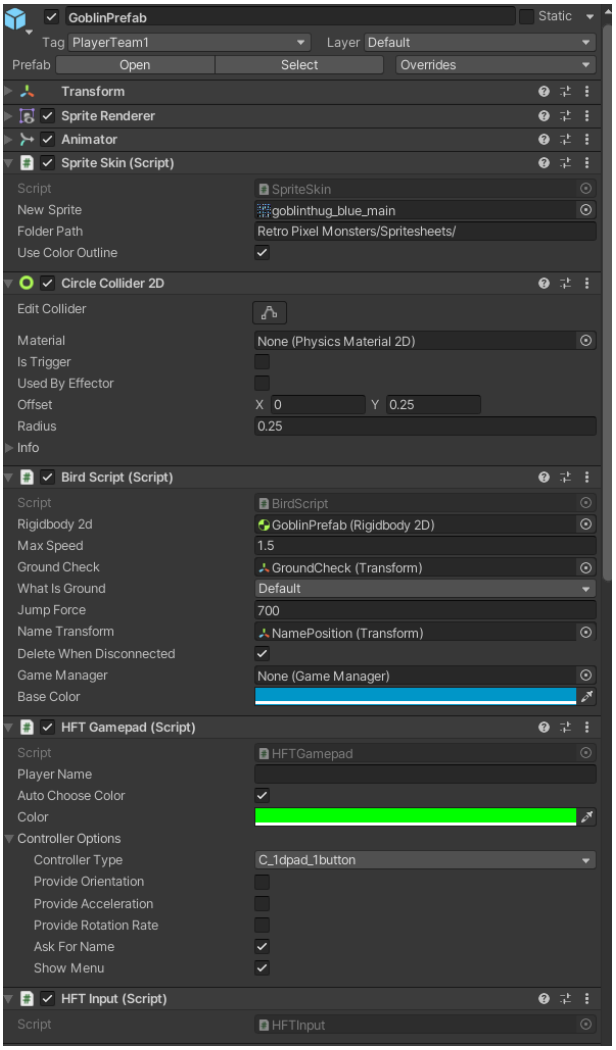

Рис. 3.4. Вид інспектору префабу гравця гоблінів

Джерело: автор

- Transform: Управляє позиціонуванням, обертанням та масштабуванням об'єкта в ігровій сцені.
- Sprite Renderer: Контролює візуальне відображення спрайту персонажа.
- Animator: Керує анімацією, дозволяючи персонажу переходити між різними анімаційними станами.

- Sprite Skin Script: Адаптує колір спрайту, щоб він відповідав кольору контролера гравця.
- Circle Collider 2D: Забезпечує фізичну взаємодію з іншими об'єктами на сцені.
- Bird Script: Дозволяє управління персонажем через контролер.
- HFT Gamepad: Налаштовує контролер для користувача.
- HFT Input: Обробляє введення даних з контролера.
- Rigidbody 2D: Додає фізичні властивості персонажу, такі як маса та тяжіння.
- Player Script: Передає інформацію про взаємодії гравця до менеджера гри.
- Upgrade Manager: Відповідає за логіку підбирання предметів та їх відображення.
- Player Statistic: Збирає та зберігає статистику гравця для подальшого використання, наприклад, при відображенні результатів гри.

3.6. Створення анімації

У процесі розробки ігрового проекту було важливо імплементувати ефективну систему анімацій, що б дозволила персонажам виглядати живими та реагувати на ігрові події. Для досягнення цього було використано Аніматор Контролер Unity, який дозволяє створити комплексну систему анімацій з переходами між різними станами.

В Аніматор Контролері були створені основні стани, такі як "Idle" (стійка спокою), "Walk" (ходьба) з різними орієнтаціями (вгору, вниз, вліво, вправо), та додаткові стани "Attack" (атака) та "Dead" (стан мертвого персонажа). Кожен стан асоційований з відповідним анімаційним кліпом, а логіка переходів між станами базується на визначених умовах, наприклад, натисканні кнопки стрибка.

Машина станів була налаштована так, щоб управління переходами відбувалось візуально та інтуїтивно, що дозволяло з легкістю моделювати поведінку персонажів. Редагування будь-якого стану в Аніматор Контролері автоматично застосовує зміни

до всіх екземплярів префаба, що забезпечує консистентність та високу адаптивність анімаційного процесу в різних сценах та умовах гри.

Використання Аніматор Контролера значно оптимізувало робочий процес, дозволяючи зосередитись на творчих аспектах анімації, водночас забезпечуючи потужні та гнучкі засоби для управління складними анімаційними сценаріями.

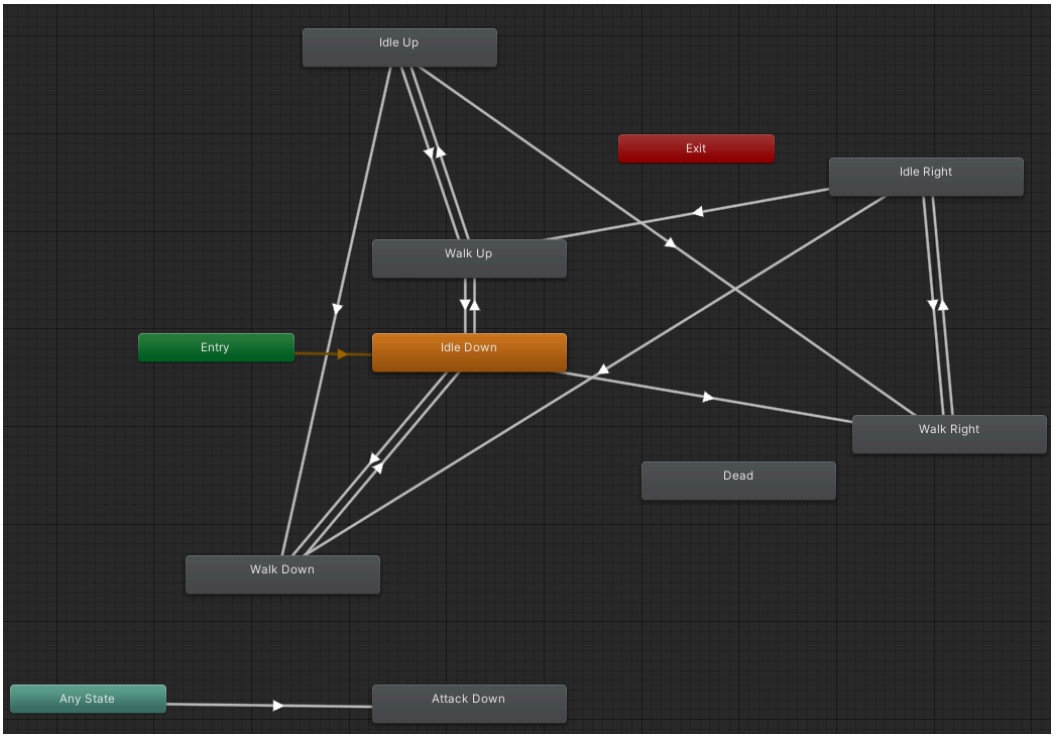

Рис. 3.5. Побудовані зв’язки між різними станами анімації об’єкту

Джерело: автор

3.7. Інтеграція A* в ігровий застосунок

В проєкті для навігації неігрових персонажів (NPC) використовувався готовий пакет алгоритму A* з Unity Asset Store. Використання цього пакету значно спростило процес інтеграції алгоритму A* та зменшило час, необхідний для його реалізації. Нижче наведено покроковий опис інтеграції алгоритму A* у Unity на основі цього пакету.

Налаштування A* Pathfinding Project

Процес налаштування алгоритму A* розпочався зі створення об'єкта для управління алгоритмом. На сцені було створено порожній об'єкт, до якого додано компонент "Astar Path". Цей об'єкт став головним для управління всім процесом навігації на основі A*.

Далі в інспекторі компонента "Astar Path" було додано новий "Grid Graph". В цьому графі визначено розмір сітки, розміри вузлів та інші параметри, щоб вони відповідали ігровій карті проекту "Goblins vs Creatures". Також було налаштовано шари, які використовуються для визначення прохідності вузлів. Це дозволило алгоритму враховувати всі перешкоди на карті під час розрахунку шляху, що забезпечило точну навігацію NPC.

Імплементація навігації NPC

Для кожного NPC на сцені було додано компонент "AIPath". Цей компонент надав можливість NPC використовувати алгоритм A* для навігації та переміщення по карті. Щоб кожен NPC мав свою цільову точку, до якої він буде рухатися, було додано компонент "AIDestinationSetter" та вказано об'єкт цілі, до якого NPC повинні прямувати.

Налаштування поведінки NPC

В інспекторі компонента "AIPath" було налаштовано параметри руху NPC, такі як швидкість, радіус обертання та інші характеристики, що впливають на їх поведінку під час навігації. Для забезпечення динамічної гри NPC змінюють свої

цілі в залежності від ігрової ситуації. Це реалізовано через скрипти, які оновлюють компонент "AIDestinationSetter" відповідно до нових умов гри.

Тестування та оптимізація

Після налаштування всіх компонентів гра була запущена для перевірки, як NPC використовують алгоритм A* для навігації. Під час тестування було переконанося, що NPC коректно реагують на перешкоди та змінюють свій маршрут при необхідності. Для досягнення оптимального балансу між точністю навігації та продуктивністю гри було налаштовано параметри сітки та оновлення шляхів. Це включало зменшення розміру вузлів та оптимізацію обчислювальних ресурсів, щоб забезпечити плавний ігровий процес.

Рис. 3.6. Результат роботи A* в Unity

Джерело: автор

ВИСНОВКИ

В рамках дипломної роботи було успішно реалізовано проект багатокористувацької офлайн гри, створеної з використанням ігрового рушія Unity та плагіна HappyFunTimes. Проект включав розробку інтуїтивно зрозумілого головного меню, яке забезпечувало легкий доступ до всіх функцій гри. Меню було структуроване так, щоб гравці могли швидко розпочати гру, підключитися до хоста, налаштувати свої профілі та переглядати статистику. Важливим аспектом було забезпечення користувацького досвіду, що дозволяло навіть новачкам без проблем освоїти інтерфейс гри.

Ефективна ігрова механіка була одним із ключових елементів проекту. Гравці поділялися на дві команди – "Creatures" та "Goblins", де кожна команда мала свої унікальні цілі та завдання. "Creatures" повинні були ловити "Goblins" та відправляти їх до склепу, в той час як "Goblins" збирали скарби, уникаючи захоплення. Така структура забезпечувала динамічний та захоплюючий геймплей, що тримав гравців у постійній напрузі.

Забезпечення високого рівня синхронізації між різними мобільними пристроями стало одним з основних завдань. Плагін HappyFunTimes дозволив реалізувати стабільне з'єднання та обмін даними між пристроями в режимі реального часу. Це забезпечило плавний ігровий процес без затримок, що є критично важливим для багатокористувацьких ігор. Крім того, налаштування мережевої взаємодії включало тестування та виправлення багів, що виникали під час підключення та гри.

Особлива увага була приділена стабільності коду. Під час розробки виявлені помилки були оперативно виправлені, що дозволило досягти безперебійної роботи гри. Використання префабів та системи анімацій Unity значно спростило процес розробки. Префаби дозволили створювати універсальні ігрові об'єкти, які можна легко змінювати та використовувати в різних частинах гри. Система анімацій Unity

забезпечила плавні переходи між станами персонажів та об'єктів, що додало грі реалістичності та динамічності.

Завдяки адаптивності плагіна HappyFunTimes гра набула можливості підтримки великої кількості гравців одночасно. Це особливо важливо для багатокористувацьких ігор, де від стабільності та швидкості обміну даними залежить якість геймплею. Успішна реалізація цього аспекту дозволила забезпечити інтерактивний та захоплюючий досвід для всіх учасників гри.

Важливою частиною проекту стала реалізація штучного інтелекту (ШІ) для керування неігровими персонажами (NPC). Основою для цього слугував алгоритм A*, інтегрований за допомогою пакету A* Pathfinding Project з Unity Asset Store. Перед вибором алгоритму ми порівняли декілька методів пошуку шляхів, включаючи алгоритми Дейкстри та жадібного найпершого пошуку. Алгоритм Дейкстри забезпечує знаходження найкоротшого шляху, але не враховує евристичні оцінки, що робить його менш ефективним для великих карт. Жадібний найперший пошук використовує евристики, але може не завжди знаходити найкоротший шлях. Алгоритм A* поєднує в собі найкращі характеристики обох методів, використовуючи евристику для прискорення пошуку і гарантуючи при цьому знаходження оптимального шляху. Це робить його ідеальним для нашого проекту, де необхідно було враховувати складні карти з численними перешкодами.

Під час реалізації алгоритму A* особлива увага приділялась налаштуванню сітки (Grid Graph), яка визначає прохідні та непрохідні ділянки на ігровій карті. Цей процес включав налаштування розміру сітки, розміру вузлів та вказання шарів перешкод, що дозволило алгоритму коректно обраховувати шляхи NPC. Важливою частиною роботи було тестування та оптимізація параметрів сітки для досягнення оптимального балансу між точністю навігації та продуктивністю гри.

Імплементація навігації для NPC включала додавання компонентів "AIPath" та "AIDestinationSetter" до кожного NPC. Це дозволило NPC використовувати алгоритм A* для знаходження найкоротших шляхів до своїх цілей, враховуючи всі перешкоди

на карті. Динамічне оновлення цілей NPC в залежності від ігрових ситуацій було реалізовано через скрипти, що значно підвищило адаптивність та реалістичність їх поведінки.

Налаштування поведінки NPC охоплювало такі параметри, як швидкість руху, радіус обертання та інші характеристики, що впливають на їхню взаємодію з оточенням. Завдяки цьому вдалося створити реалістичні умови для NPC, які коректно реагували на зміни в ігровому середовищі та забезпечували цікаву динаміку гри. Особливо корисним виявилось використання префабів, які спростили управління ресурсами та дозволили швидко вносити зміни до поведінки NPC.

Тестування гри показало, що NPC коректно реагували на перешкоди, змінюючи свої маршрути в реальному часі, що додало грі динамічності та непередбачуваності. Це підвищило загальний ігровий досвід та задоволення гравців від взаємодії з NPC.

Таким чином, реалізація штучного інтелекту з використанням алгоритму A* та пакету A* Pathfinding Project з Unity Asset Store стала важливим етапом у розробці проекту "Goblins vs Creatures". Вона дозволила створити ефективну систему навігації для NPC, забезпечивши реалістичну поведінку персонажів та високий рівень динаміки гри. Цей досвід став важливим внеском у моє розуміння розробки багатокористувацьких ігор та роботи з штучним інтелектом у середовищі Unity.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Unity. URL: https://docs.unity3d.com/Manual/index.html (дата звернення 12.02.2024)
2. C#. URL: https://learn.microsoft.com/en-us/dotnet/csharp/ (дата звернення 04.03.2024)
3. HappyFunTimes. URL: https://docs.happyfuntimes.net/docs/ (дата звернення 19.02.2024)
4. Покликання на проект в GitHub «GoblinsVsCreatures». URL: https://github.com/vladdziun/GoblinVsCreatures_OA (дата звернення 25.02.2024)
5. "Jackbox Party Packs". URL: https://store.steampowered.com/app/434170/The_Jackbox_Party_Pack_3 (дата звернення 03.02.2024)
6. "Keep Talking and Nobody Explodes". URL: https://www.nintendo.com/us/store/products/keep-talking-and-nobody-explodes-switch/ (дата звернення 01.03.2024)
7. Pathfinding algorithms: Dijkstra, BFS, A*. URL: https://theory.stanford.edu/~amitp/GameProgramming/AStarComparison.html (дата звернення 28.02.2024)
8. A* Pathfinding Project. URL: https://en.wikipedia.org/wiki/A*_search_algorithm (дата звернення 06.02.2024)
9. Unity Tile Palette. URL: https://docs.unity3d.com/Manual/Tilemap-Palette.html (дата звернення 07.03.2024)
10. Unity Sprite Editor. URL: https://docs.unity3d.com/Manual/SpriteEditor.html (дата звернення 14.02.2024)
11. Unity Animator Controller. URL: https://docs.unity3d.com/Manual/AnimatorController.html (дата звернення 27.02.2024)

12. WebSockets in Unity. URL: https://docs.unity3d.com/Manual/UnityWebRequest-WebSockets.html (дата звернення 08.02.2024)
13. HappyFunTimes API. URL: https://docs.happyfuntimes.net/docs/API (дата звернення 21.02.2024)
14. Unity Prefabs. URL: https://docs.unity3d.com/Manual/Prefabs.html (дата звернення 05.02.2024)
15. Unity GameObject. URL: https://docs.unity3d.com/ScriptReference/GameObject.html (дата звернення 22.03.2024)
16. Unity Rigidbody2D. URL: https://docs.unity3d.com/ScriptReference/Rigidbody2D.html (дата звернення 17.02.2024)
17. Unity CircleCollider2D. URL: https://docs.unity3d.com/ScriptReference/CircleCollider2D.html (дата звернення 03.03.2024)
18. Unity MonoBehaviour. URL: https://docs.unity3d.com/ScriptReference/MonoBehaviour.html (дата звернення 09.02.2024)

ДОДАТКИ

ДОДАТОК А

Скрипт «GameManager»

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

public class GameManager : MonoBehaviour
{
    public static GameManager instance = null;
    public static int moneyBags = 0;
    public static int goblins = 0;
    public int maxGoblins;
    public static int creatures = 0;
    public Text bagsCount;
    public Text goblinsCount;
    public Text startGoblinsCount;
    public Text startCreaturesCount;
    public Text winText;
    public Text goblinsMvp;
    public Text creaturesMvp;
    public static bool isGameStarted = false;
    public static bool isGoblinsWin;
    public static  bool isCreaturesWin;
    public GameObject startGameCanvas;
    public GameObject mainGameCanvas;
    public GameObject winCanvas;
    public Vector2 winnerCoordiantes;

    void Awake()
    {
        //Check if instance already exists
        if (instance == null)
            instance = this;

        //If instance already exists and it's not this:
        else if (instance != this)
            Destroy(gameObject);
    }

    void Start()
    {
        GetGoblinsBags();

        bagsCount.text = "Bags Left: " + moneyBags;
        goblinsCount.text = "Goblins Left: " + goblins;
    }

    // Update is called once per frame
    void Update()
    {
        if (isGameStarted)
            Camera.main.GetComponent<AudioSource>().enabled = true;

        if(moneyBags <= 0 && isGameStarted)
```

```
        {
            winCanvas.SetActive(true);
            isGameStarted = false;
            winText.text = "GOBLINS WIN!";
            isGoblinsWin = true;
        }
        if(goblins <= 0 && isGameStarted)
        {
            winCanvas.SetActive(true);
            isGameStarted = false;
            winText.text = "CREATURES WIN!";
            isCreaturesWin = true;

        }

        if (isGoblinsWin || isCreaturesWin)
        {
            var allCreatureStatistics = GameObject.FindGameObjectsWithTag("PlayerTeam0")
                .ToList();
            var allGoblinStatistic = GameObject.FindGameObjectsWithTag("PlayerTeam1")
                .ToList();

            // finds the best pair
            var bestCreature = allCreatureStatistics.Aggregate((x, y) =>
x.GetComponent<PlayerStatistic>().GoblinsCaptured >
y.GetComponent<PlayerStatistic>().GoblinsCaptured ? x : y);
            var bestGoblin = allGoblinStatistic.Aggregate((x, y) =>
x.GetComponent<PlayerStatistic>().BagsCollected > y.GetComponent<PlayerStatistic>().BagsCollected
? x : y);

            var bestCreatureStatistic = bestCreature.GetComponent<PlayerStatistic>();
            var bestGoblinStatistic = bestGoblin.GetComponent<PlayerStatistic>();

            creaturesMvp.color = bestCreatureStatistic.PlayerColor;
            goblinsMvp.color = bestGoblinStatistic.PlayerColor;
            creaturesMvp.text = $"{bestCreature.GetComponent<HFTGamepad>().playerName} captured
{bestCreatureStatistic.GoblinsCaptured} {(bestCreatureStatistic.GoblinsCaptured > 1 ? "goblins" :
"goblin")}!";
            goblinsMvp.text = $"{bestGoblin.GetComponent<HFTGamepad>().playerName} collected
{bestGoblinStatistic.BagsCollected} {(bestGoblinStatistic.BagsCollected > 1 ? "bags" : "bag")}!";

            Camera.main.transform.position = isGoblinsWin
                ? Vector3.Lerp(Camera.main.transform.position, new
Vector3(bestGoblin.transform.position.x, bestGoblin.transform.position.y,
Camera.main.transform.position.z), Time.deltaTime * 2)
                : Vector3.Lerp(Camera.main.transform.position, new
Vector3(bestCreature.transform.position.x, bestCreature.transform.position.y,
Camera.main.transform.position.z), Time.deltaTime * 2);
            Camera.main.orthographicSize = Mathf.Lerp(Camera.main.orthographicSize, 5,
Time.deltaTime * 2);
        }


        if (!isGameStarted)
            maxGoblins = GameObject.FindGameObjectsWithTag("PlayerTeam1").Length;
        if (isGameStarted)
        {
            int newMaxGoblins = GameObject.FindGameObjectsWithTag("PlayerTeam1").Length;
            if (maxGoblins != newMaxGoblins)
            {
                goblins = goblins - (maxGoblins - newMaxGoblins);
                maxGoblins = newMaxGoblins;
                UpdateCount();
            }
```

```
        }
    }

    public void UpdateCount()
    {
        Debug.Log("Goblins left:" + goblins);
        bagsCount.text = "Chests Left: " + moneyBags;
        goblinsCount.text = "Goblins Left: " + goblins;
    }

    public void GetGoblinsBags()
    {
        moneyBags = GameObject.FindGameObjectsWithTag("Bag").Length;
        goblins = GameObject.FindGameObjectsWithTag("PlayerTeam1").Length;
        creatures = GameObject.FindGameObjectsWithTag("PlayerTeam0").Length;
        startGoblinsCount.text = "Goblins Connected: " + goblins;
        startCreaturesCount.text = "Creatures Connected: " + creatures;

    }

    public void StartGame()
    {
        Debug.Log("Game started!");
        isGameStarted = true;
        startGameCanvas.GetComponent<Canvas>().enabled = false;
        mainGameCanvas.GetComponent<Canvas>().enabled = true;
    }


    public void DecreaseGoblins()
    {
        goblins--;
    }
}
```