НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

До захисту допущено:
в.о. завідувача кафедри
_____ Михайло Новотарський
(підпис)

“__” _____2025 р.

Дипломний проєкт

на здобуття ступеня бакалавра

за освітньо-професійною програмою “Комп’ютерні системи та мережі”

спеціальності 123 “Комп’ютерна інженерія”

на тему: Модель 2D-агента з навчанням з підкріпленням для ігрового персонажа

Виконав: студент 4 курсу, групи ____ІО-11_____
(шифр групи)

_____Ткаченко Олександр Андрійович_____ _____
(прізвище, ім’я, по батькові) (підпис)

Керівник _____проф., д.т.н., проф. Новотарський М. А.____ _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант (нормоконтроль)____ас. Гончаренко О.О._____ _____
(назва розділу) (посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Рецензент _____к.т.н., доц. Шимкович В.М._____ _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2025 р.

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

Рівень вищої освіти – перший (бакалавр)

Освітньо-професійна програма

“Комп’ютерні системи та мережі”

спеціальність 123 “Комп’ютерна інженерія”

До захисту допущено
в.о. завідувач кафедри ОТ
_____ Михайло Новотарський.
(підпис)

“__” _____2025 р.

ЗАВДАННЯ
на бакалаврський дипломний проект студента

Ткаченко Олександр Андрійович

1. Тема Модель 2D-агента з навчанням з підкріпленням для ігрового персонажа
керівник проєкту . Новотарський Михайло Анатолійович д.т.н., проф.,
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)
затверджені наказом по університету від 23.05 2025 року №1705.с
2. Термін здачі студентом закінченого проекту *18 травня 2025 р.*
3. Вихідні дані до проекту *технічна документація. теоретичні та статистичні дані, патенти на винахід*
4. Зміст розрахунково-пояснювальної записки (перелік питань, які розробляються)

Розділ 1. Огляд та аналіз існуючих відеоігор жанру стратегії та навчання з підкріпленням в іграх або дослідженні у віртуальних симуляціях.
Розділ 2. Аналіз засобів розробки.
Розділ 3. Реалізація програмного продукту.
Розділ 4. Огляд та тестування розробленого продукту

5. Перелік графічного матеріалу (з точним позначенням обов’язкових креслень) структурна схема системи, Діаграма класів (функціональна схема), алгоритм дій програмного забезпечення (принципова схема).

6. Консультанта проекту, з вказівкою розділів проекту, які до них вносяться

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання _____

Календарний план

№ п/п	Найменування етапів дипломного проєкту	Терміни виконання етапів проєкту	Примітки
1.	*Затвердження теми проєкту*	9.01.2025-29.01.2025	
2.	*Вивчення та аналіз завдання*	29.01.2025-14.04.2025	
3.	*Розробка архітектури та загальної структури системи*	14.04.2025-21.04.2025	
4.	*Розробка структур окремих підсистем*	21.04.2025-28.04.2025	
5.	*Програмна реалізація системи*	29.04.2025-20.05.2025	
6.	*Оформлення пояснювальної записки*	28.04.2025-26.05.2025	
7.	*Захист програмного продукту*	28.05.2025	
8.	*Передзахист*	31.05.2025	
9.	*Захист*	18.06.2025	

Студент-дипломник _____ Олександр Ткаченко
(підпис)

Керівник проєкту _____ Михайло Новотарський
(підпис)

Анотація

В бакалаврському дипломному проекті реалізовано робота моделі 2D-агента для гри у жанрі стратегії який використовує навчанням з підкріпленням для ігрового персонажа задача якого проходити лабіринт з перешкодами і мати змогу адаптуватися до його змін які вносить гравець.

Для реалізації гри було використано рушій Unity, а також мову програмування C#.

З метою поліпшення поведінки ШІ-агентів у процесі розробки застосовано методи машинного навчання, а саме, засоби Unity Machine Learning Agents.

Annotation

The bachelor's thesis project implemented the work of a 2D agent model for a strategy game that uses reinforcement learning for a game character whose task is to pass a maze with obstacles and be able to adapt to changes made by the player.

The Unity engine was used to implement the game, as well as the C# programming language.

In order to improve the behavior of AI agents during the development process, machine learning methods were used, namely, Unity Machine Learning Agents.

справки	Формат	*Значення*	*Найменування*	Кіл. листів	№ екземпля	*Додаток*
			Документація загальна			
			Знову розроблена			
	А4	***ІАЛЦ.467200.002 ТЗ***	Модель 2D-агента з	3		
			навчанням з підкріпленням			
			для ігрового персонажа			
			Технічне завдання			
	А4	***ІАЛЦ.467200.003 ПЗ***	Модель 2D-агента з	56		
			навчанням з підкріпленням			
			для ігрового персонажа			
			Пояснювальна записка			
	А4	***ІАЛЦ.467200.004 Д1***	Модель 2D-агента з	1		
			навчанням з підкріпленням			
			для ігрового персонажа			
			Структурна схема			
			системи			
	А4	***ІАЛЦ.4672008.005 Д2***	Модель 2D-агента з	1		
			навчанням з підкріпленням			
			для ігрового персонажа			
			Функціональна схема			
			(діаграма класів)			
	А4	***ІАЛЦ.467200.006 Д3***	Модель 2D-агента з	1		
			навчанням з підкріпленням			
			для ігрового персонажа			
			Алгоритм дій програмного			
			забезпечення			
	А4	***ІАЛЦ.467200.007 Д4***	Модель 2D-агента з	21		
			навчанням з підкріпленням			
			для ігрового персонажа			
			Текст програмного коду			

Зм	*Лист*	*№ докум.*	*Підп*	*Дата*	

ІАЛЦ.467200.001 ОА

Розроб		Ткаченко О.А.			Модель 2D-агента з навчанням з підкріпленням для ігрового персонажа Опис альбому	Літ.	Аркуш 1 / Аркушів 1
Перев		Новотарський М. А.					
						КПІ ім. Ігоря Сікорського, ФІОТ, ІО-11	

ТЕХНІЧНЕ ЗАВДАННЯ

ДО ДИПЛОМНОГО ПРОЄКТУ

на тему: «Модель 2D-агента з навчанням з підкріпленням для ігрового персонажа»

Київ – 2025

ЗМІСТ

					ІАЛЦ.467200.002 ТЗ		
		№ докум.	Підпис	Дата			
Розробив		Ткаченко О.А.			Модель 2D-агента з навчанням з підкріпленням для ігрового персонажа Технічне завдання	Літ.	Аркуш 1 / Аркушів 3
Перевірив							
						КПІ ім. Ігоря Сікорського, ФІОТ, ІО-11	
Н. Контр.		Новотарський М.А					
Затвердив							

1 НАЙМЕНУВАННЯ ТА ОБЛАСТЬ ЗАСТОСУВАННЯ

Дане технічне завдання поширюється на розробку ігрового застосунку, який для ШІ-ворогів буде використовувати навчання з підкріпленням, а також на подальшу підтримку та вдосконалення розробленого продукту.

Областю застосування є гравці які цікавляться іграми в жанрі стратегії з елементами будівництва і захисту бази де штучний інтелект ворога може підлаштовуватися під дії гравця.

2 ПІДСТАВИ ДЛЯ РОЗРОБКИ

Підставою для розробки даної системи є завдання для виконання роботи кваліфікаційно-освітнього рівня «бакалавр інженерії програмного забезпечення», який був затверджений факультетом “Інформатики та обчислювальної техніки” кафедрою обчислювальної техніки Національного технічного Університету України «Київський Політехнічний інститут ім. Ігоря Сікорського».

3 МЕТА ТА ПРИЗНАЧЕННЯ РОЗРОБКИ

Метою та призначенням даної роботи є розробка ігрового застосунку та застосування методів навчання з підкріпленням для реалізації поведінки персонажів.

4 ДЖЕРЕЛА РОЗРОБКИ

Джерелом розробки даного дипломного проекту є офіційні документації, публікації та статті в мережі Інтернет на дану тему, науково-технічна література.

5 ТЕХНІЧНІ ВИМОГИ

5.1. Вимоги до розробленого продукту

Розроблена система має виконувати такі вимоги:

- В грі повинні бути реалізовані ігрові механіки пересування камери, масштабування камери, редагування карти та перемикання на зір супротивника.
- Поведінка супротивників повинна бути реалізована за допомогою методів навчання з підкріпленням.
- Для навчання інтелектуальних агентів повинні бути реалізовані додаткові середовища тренування
- Повинні бути реалізовані головне меню.

5.2. Вимоги до програмного забезпечення

- ОС Windows

5.3. Вимоги до апаратної частини

- ЦП не менше за Intel® Core (TM) i5-8300H.
- ROM не менше ніж 16 ГБ.
- RAM не менше ніж 8 ГБ.
- GPU Nvidia GeForce GTX 1050Ti або кращий.

6 ЕТАПИ РОЗРОБКИ

Назва етапів виконання	Термін виконання
Затвердження теми роботи	9.01.2025-29.01.2025
Вивчення та аналіз завдання	29.01.2025-14.04.2025
Розробка архітектури та загальної структури системи	14.04.2025-21.04.2025
Розробка структур окремих частин системи	21.04.2025-28.04.2025
Програмна реалізація системи	29.04.2025-20.05.2025
Виправлення помилок	28.04.2025-26.05.2025
Оформлення пояснювальної записки	9.01.2025-29.01.2025

ПОЯСНЮВАЛЬНА ЗАПИСКА
ДО ДИПЛОМНОГО ПРОЄКТУ

на тему: «Модель 2D-агента з навчанням з підкріпленням для ігрового персонажа»

Київ – 2025

ЗМІСТ

					ІАЛЦ.467200.003 ПЗ		
Зм.	Арк.	№ докум.	Підпис	Дата			
Розробив		Ткаченко О.А.			Модель 2D-агента з навчанням з підкріпленням для ігрового персонажа Технічне завдання	Літ.	Аркуш 1 / Аркушів 56
Перевірив							
Реценз.						КПІ ім. Ігоря Сікорського, ФІОТ, ІО-11	
Н. Контр.		Новотарський М.А					
Затвердив							

ПЕРЕЛІК СКОРОЧЕНЬ

RTS – Real-Time Strategy – стратегія в реальному часі

TBS – Turn-Based Strategy – покрокова стратегія

PvP – Player versus Player – гравець проти гравця

RPG – Role-Playing Game – рольова гра

DLC – Downloadable Content – завантажуваний контент

NPC – Non-Player Character – неігровий персонаж

GPU – Graphics Processing Unit – графічний процесор

PNG – Portable Network Graphics – формат зображення

ID – Identifier – ідентифікатор

RL – Reinforcement Learning – навчання з підкріпленням

PPO – Proximal Policy Optimization – алгоритм навчання з підкріпленням

SAC – Soft Actor-Critic – зразковий RL-алгоритм із підвищеною ефективністю

ШІ – Штучний інтелект

ВСТУП

В наш час коли відеоігри стають в один ряд з іншими видами мистецтва, як колись це зробили і фільми. Розробники шукають різні методи, щоб виділити своє творіння на фоні інших ігор, для отримання більшого прибутку та популярності. Оскільки у світі існує велике різноманіття комп'ютерних, консольних та мобільних відеоігор, які зазвичай приваблюють гравців різноманітним сюжетом, геймплеєм, графікою та рівнем складності.

Перед тим, як продовжити давайте коротко розглянемо ці 4 ключові елементи гри[1]:

- Геймплей
- Сюжет
- Дизайн та графіка
- Виклик гравцю

По перше геймплей який включає в себе цікаві ігрові механіки та веселий ігровий процес, який буде захоплюючим та цікавим для гравців. Також геймплей має поступово відкривати нові можливості для гравця. Одне з найважливіших задач при створенні геймплею це досягти балансу між простотою входження та складністю на пізніших стадіях гри.

По друге сюжет який дозволяє гравцям краще зрозуміти світ гри, її персонажів та історію цього світу, що дає змогу гравцю відчувати різноманітні емоції піч час проходження гри від страху перед чимось невідомим або радістю після розкриття якоїсь таємниці, аж до печалі після смерті твого улюбленого персонажу, ці речі додають цікавості в дослідження цього світу. Як приклад хочеться навести гру "Відьмак 3: Дикий гін" від польської студії CD Project Red, яка є частиною трилогії, що продовжують книжкову історію.

Під час гри ви будете зустрічати різних персонажів, як таких які вже зустрічалися в минулих частинах гри так і тих які були описані лише у книжках, а у діалогах і записках які розкидані по всьому відкритому світі нерідко буває таке, що згадують про події минулих частин або книг. Це все дає гравцю відчути настрій гри а тим хто вже знайомий з цим світом дозволяє згадати цікаві моменти які відбулися в його минулому. Це один з прикладів гарного сюжету, який не просто переповідає книжний оригінал, а розповідає історію яка відбулася після цього.

По третє дизайн та графіка які не обов'язково повинні бути реалістичними – головне, щоб вони були унікальними і цілісними. Оскільки хороший дизайн доповнює атмосферу та занурення у світ гри. Здебільшого можна відокремити кілька видів графіки, а саме 2D-графіку для якої не треба мати навички в використанні різних 3D-редакторів, але для анімації яких зазвичай використовують по кадрову анімацію ніж скелетну, що збільшує час в її реалізації, далі іде 3D-графіка для якої треба вміти створювати 3D-моделі в різних редакторах і вже для яких легше використовувати скелетну анімацію, також окремо хочу виділити піксельну графіку яка була основною на початку створення відеоігор.

І останнє це виклик гравцю, що є для багатьох гравців одним з найважливіших елементів гри, без якого в грі, ніби, відсутній сенс, тобто те, заради чого ми граємо. В іграх перед гравцями ставлять різні по складності задачі від незначних або легких, аж до тих де в нього майже немає шансів на помилку. Току кожному гравцю можуть подобатися різні за складністю ігри такі, як "the Sims" або інші пісочниці, які дають гравцю майже безмежні можливості для гри. Також є гравці якім більше до вподоби майже неможливі випробування, як у іграх серії "Dark Souls" від яких навіть через їхню велику складність яка не дає гравцю права на помилку, пішов новий жанр під назвою Souls-like який став маркою екстремально складних ігор.

Також окремо хочу розглянути таки елемент у іграх як повторна іграбельність який є важливим для гравців за для відчуття різноманітних емоцій та можливості додали різноманітні перешкоди які не будуть схожі одна на одну, зазвичай цей елемент залежить від геймплея та виклику гравцю. Зазвичай в іграх з великою повторною іграбельністю графіка і дизайн відходять на другий план, а сюжет не дуже важливий, оскільки гравець зазвичай в цій грі сам створює історію, як приклад можна взяти гру “RimWorld” де кожний новий старт може бути унікальним по складності в залежності від вибору гравця, де в одному ви можете побудувати велетенську базу в непрохідних горах яка буде витримувати всі напади ворогів і яка в результаті паде від якоїсь хвороби, а в іншому опинитися самотнім дослідником в мертвій крижаній пустелі. Тому в такі ігри і хочеться грати бо кожна партія в них виходить унікальною і неповторною. Також повторну іграбельність можуть підтримувати і самі гравці шляхом розробки та розповсюджування різноманітних модифікацій, за приклад можна взяти “Minecraft” для якого користувачі створили вже кілька тисяч модифікацій, від тих що незначно впливають на ігровий процес, а ж до тих які роблять з нього зовсім іншу гру

Отже, як розробник ігор своєю задачею я обираю створення хорошої гри з гарною повторною іграбельністю в якій гравці зможуть кидати собі нові виклики і створювати свої унікальні історії. І для реалізації цього я обрав створити гру у жанрі RTS з елементами Base-building та Tower Defense, з використання навчання з підкріпленням для створення ШІ-ворогів, щоб вони під час гри могли адаптуватися до дій гравця, оскільки проблема багатьох ігор у жанрі стратегій це зазвичай шаблона поведінка ШІ до якого гравці з часом звикають, що призводить до втрати інтересу до гри.

РОЗДІЛ 1. ОГЛЯД ТА АНАЛІЗ ІСНУЮЧІХ ВІДЕОІГОР ЖАНРУ СТРЕТЕГІЇ ТА НАВЧАННЯ З ПІДКРІПЛЕННЯМ В ІГРАХ АБО ДОСЛІДЖЕННІ У ВІРТУАЛЬНИХ СИМУЛЯЦІЯХ

1.1 Визначення понять та огляд предметної області

Сучасні ігри мають велику кількість жанрів та піджанрів, на одній з популярніших площадок Steam в якій нараховується 452[2] теги до яких окрім жанрів та піджанрів відносяться ще і візуальний стиль, кількість гравців, тема гри та різні їх особливості. Також одна гра може належати до кількох жанрів або піджанрів в залежності від її особливостей. Як приклад, можна навести кілька основних жанрів комп'ютерних ігор [3,4,5]:

- Симулятори - це ігри, які відтворюють певну ситуацію з реального або вигаданого світу і надають гравцю контроль над нею. Це може бути симулятор військових дій, пілотування, гонки або спорту, тощо. Як приклад можна навести такі ігри як: The Sims 4, Farming simulator 22, SQUAD.

- Стратегії – це ігри в яких для досягнення перемоги треба використовувати планування та стратегічне мислення для використання внутрішньо ігрових ресурсів. Ігри цього жанру можуть бути на різну тематику: військову, економічну, політичну тощо. Як приклад можна навести такі ігри як: Stellaris, Civilization, ANNO.

- Екшн-ігри - ігри, в яких гравець керує певним персонажем відповідаючи за його пересування та інші дії з метою знайдення шляху до наступної цілі

яка сприяє поступовому проходженні гри. Також до екшн-ігор можуть відноситися ігри, в яких гравець стріляє по певним супротивникам або об’єктам. Як приклад можна навести такі ігри як: SQUAD, Counter-Strike, Battlefield.

- Рольові ігри - дозволяють гравцю увійти у роль персонажа або групи, і взаємодіяти з ігровим навколишнім світом. В даному жанрі часто застосовується механіка розвитку персонажів по мірі надходження досвіду, перемог у боях, виконання завдань тощо. Як приклад можна навести такі ігри як: World of Warcraft, Diablo, Dying Light.

Існують ще багато інших жанрів та піджанрів але серед їх всіх, особливо треба розглянути RTS, Base-building та Tower Defense, оскільки вони є основними жанрами цього проекту[6,7,8]:

- RTS-ігри передбачають керування арміями, ресурсами та розвитком бази у режимі реального часу. Гравець повинен приймати стратегічні рішення швидко, одночасно керуючи будівництвом, економікою та бойовими діями. Особливість жанру — відсутність "ходів": усі дії відбуваються без пауз. Прикладами є такі ігри як: StarCraft, Age of Empires, Command & Conquer.

- Base-building - ігри цього жанру зосереджуються на створенні, розширенні та оптимізації бази чи поселення. Гравець керує розміщенням будівель, управлінням ресурсами, а часто й обороною своєї території. Цей жанр може поєднуватись з іншими, як-от стратегії чи симулятори. Прикладами є: Factorio, They Are Billions, RimWorld.

- У Tower Defense-іграх гравець будує оборонні споруди (вежі) для захисту від хвиль ворожих юнітів, які намагаються пройти через карту. Основна

мета — стратегічно розміщувати вежі, щоб завдати найбільшої шкоди противнику, з урахуванням типів ворогів, ресурсів і лімітів будівництва. Прикладами таких ігор є: Plants vs. Zombies, Kingdom Rush, Bloons TD.

Також перед розробкою гри треба звернути увагу на те наскільки складною вона буде, оскільки гравців можна поділити на кілька категорій за рівнем складності на якому їм подобається грати[9, 10]:

- Казуальні гравці надають перевагу іграм, в які просто та весело грати та які не ставлять перед гравцем складні задачі.
- Core-гравці - мають ширшу область інтересів ніж казуальні гравці та можуть грати в різні види ігор, але не мають стільки ж ентузіазму і не витрачають на гру стільки ж зусиль, як хардкорні гравці.
- Хардкорні гравці надають перевагу складним іграм з високим порогом входження і отримують задоволення від подолання важких випробувань або оволодіння складними навичками.
- Гравці, які полюбляють змагання, беруть участь у турнірах проти інших геймерів, або просто отримують задоволення від PvP-сутичок з іншими гравцями.
- Кооперативні гравці - отримують задоволення від спілкування з іншими у процесі гри, знаходженні нових друзів у онлайн грі.
- Гравці які використовують ігри для навчання, саморозвитку. В іграх можуть займатися конструюванням, вивченням математики, програмування, логіки або навіть певній природній мові, тощо.

Отже, у ігри жанру стратегії можуть грати будь-які з розглянутих типів гравців, в залежності від ігрових механік та складності ШІ ворогів.

Ігрова механіка - певний аспект гри та правила, якими послуговується гравець під час гри [11]. У стратегіях вирізняють механіки будівництва, наймання різних персонажів, пересування камери по полю та інколи до цього можна додати обмеження видимого поля, якщо гравець його не дослідив або не має там спостерігача, яким може буди будівля або персонаж.

Також треба розглянути, що ігри можуть різнитися за кількістю гравці на однокористувацькі де у ролі суперника виступає ШІ так і на багатокористувацькі в яких спільно грає від двох гравців змагаючись між собою або проти ШІ.

Тож давайте розглянемо як на складність гри впливає ШІ, оскільки якщо ШІ написаний просто, то з часом гравець може під нього під лаштуватися, що призведе до значного спрощення гри та можливої втрати до неї інтересу.

Для початку треба зауважити, що для ігрового ШІ не обов'язково використовувати нейронні мережі чи інші методи машинного навчання, оскільки зазвичай використовують алгоритми, набори правил, дерево поведінки тощо. Від обраного методу створення ШІ може залежати те наскільки легко буде передбачити його дії, але треба притримуватися золотої середини складності, щоб ворог і не був занадто простим якого можна буде легко передбачити і який не буде мати змоги вносити зміни в свою поведінку, так і не занадто складним який буде прораховувати дії гравця на перед і вибирати дії які не дадуть змоги користувачу перемогти.

Реалістична та непередбачувана поведінка комп'ютера, яка буде імітувати дії гравця може його зацікавити необхідністю постійної зміни тактики та пошуку інших способів використання різних механік гри, що зробить ігровий процес цікавішим і різноманітнішим та дозволить затримати гравця на більшу кількість часу. Через це в цій роботу буде спроба створити агента який буде адаптуватися під дії користувача для імітації людської поведінки.

Для імітації людиноподібної поведінки ігровим персонажем можна використати технології штучного інтелекту. В такому випадку ігровий персонаж розглядається як інтелектуальний агент.

Інтелектуальний агент — це програма, яка може приймати рішення або виконувати дії на основі даних з навколишього середовища, введених даних користувача або власного досвіду[12].

Популярними методами реалізації поведінки інтелектуального агента є дерева рішень, створення системи зі скінченною кількістю станів, а також застосування нейронних мереж.

Отже, ми щойно розглянули основні поняття, потрібні для розуміння предметної області. Тепер слід розглянути існуючі рішення - ігри жанру стратегії та порівняти їх між собою, знайти їх переваги, недоліки та визначити застосовані методи ігрового ШІ. Також треба розглянути використання навчання з підкріпленням в іграх або інших моделях.

1.2 Огляд відеоігор у жанрі стратегії

В цьому розділі розглянемо різні ігри-представники жанру шутер, його піджанрів – RTS та TBS.

1.2.1 RimWorld

RimWorld — це представник жанру симулятор колонії з елементами стратегії у реальному часі, тактичної бойової системи та оповідання, що генерується гравцем. У грі гравець керує групою колоністів, які виживають на віддаленій планеті після аварії космічного корабля. Головна мета — не просто

вижити, а розвивати поселення, керувати ресурсами, настроями та здоров'ям персонажів, відбивати атаки та намагатись урешті-решт покинути планету.

Рисунок 1.1 – Скріншот з гри RimWorld

Гра була випущена у 2018 році студією Ludeon Studios та відома своєю глибокою симуляцією поведінки персонажів, процедурною генерацією подій і високою варіативністю проходження. Кожен колоніст має унікальні риси характеру, бекграунд, уподобання та психологічні стани, що впливають на його дії.

Ключові ігрові механіки, що впливають на геймплей:

- Симуляція психіки: кожен колоніст має рівень настрою, який змінюється в залежності від харчування, оточення, стосунків та пережитих подій.

- Розширене будівництво: гравець розпоряджається розміщенням стін, кімнат, оборонних споруд, інфраструктури, сільського господарства та виробництва.

- Бойова система: бої відбуваються у реальному часі з можливістю використовувати паузу, та з можливістю тактичного керування бійцями. Враховуються дальність, обстріл, укриття, тип зброї та рівень навичок.

- Події: події формуються за допомогою оповідача, який задає темп гри: напади рейдерів, хвороби, погода, психічні зриви, тощо.

- Мікроменеджмент: потребує уважного керування їжею, лікуванням, одягом, розподілом обов’язків і ресурсів.

Недоліки RimWorld:

- Високий поріг входження — новачкам важко опанувати інтерфейс та систему механік без підказок.

- Відсутність повноцінного навчання всередині гри, що ускладнює старт.

- Графіка спрощена та стилізована, що може не відповідати очікуванням гравців, які шукають візуальну глибину.

- Інколи події здаються надто жорстокими або несправедливими, що викликає фрустрацію.

1.2.2 Warcraft III

Warcraft III — це класична RTS з елементами рольової гри, створена компанією Blizzard Entertainment. Вперше випущена у 2002 році, гра стала однією з найвпливовіших у своєму жанрі завдяки глибокому геймплею, яскравому фентезійному світу та інноваційній механіці героїв.

Основна суть гри полягає в тому, щоб керувати армією, збирати ресурси, будувати базу, створювати війська і вести бойові дії проти інших фракцій. Гравець може обрати одну з чотирьох рас: Люди, Орки, Нежить та Нічні ельфи — кожна з унікальною стилістикою, юнітами та механікою.

Рисунок 1.2 – Скріншот з гри Warcraft III

Ключові ігрові механіки, що впливають на геймплей:

- Ресурси: гравець збирає золото, дерево та керує лімітом юнітів, щоб розвивати базу та створювати армію.
- Герої: кожна сторона має доступ до героїв — потужних юнітів, які набирають досвід, підвищують рівень і мають унікальні вміння. Герої — ключовий елемент тактики.
- Бойова система: стратегічна, з врахуванням розміщення військ, типів юнітів, мікроконтролю і заклинань.

- Кампанія: сюжетна частина гри розгортається у світі Азероту й охоплює масштабну історію, яка лягла в основу всесвіту World of Warcraft.
- Редактор карт: гравці можуть створювати власні сценарії, що призвело до народження жанру MOBA, наприклад, культовий мод DotA.

Переваги Warcraft III:

- Глибокий та збалансований геймплей з унікальними расами.
- Введення RPG-механік в RTS.
- Захопливий сюжет та якісні кат-сцени у кампанії.
- Один з найкращих редакторів карт в історії стратегій.
- Велика спільнота та вплив на розвиток інших жанрів.

Недоліки Warcraft III:

- Висока складність керування — потребує швидкої реакції та мікроконтролю.
- Деякі раси вимагають більше навичок для ефективної гри, що може створювати дисбаланс серед новачків.
- Високі вимоги до багатозадачності — потрібно одночасно керувати базою, армією та героєм.

1.2.3 Sid Meier’s Civilization VI

Sid Meier’s Civilization VI — це TBS , де гравець бере на себе роль лідера однієї з націй і веде її від кам’яної доби до ери космічних подорожей. Гра належить до жанру 4X (eXplore, eXpand, eXploit, eXterminate) і є шостою частиною культової серії Civilization від студії Firaxis Games. Вперше була випущена у 2016 році.

Мета гри — створити найуспішнішу цивілізацію за допомогою одного з кількох шляхів до перемоги: військової, наукової, культурної, релігійної або дипломатичної. Гравець досліджує карту, керує містами, арміями, веде дипломатію, розвиває технології та укріплює внутрішню інфраструктуру.

Рисунок 1.3 — Скріншот з гри Civilization VI

Ключові ігрові механіки, що впливають на геймплей:

- Покрокова система: кожна дія відбувається у свій хід. Це дозволяє гравцю планувати стратегію заздалегідь і не потребує реакції в реальному часі.

- Розміщення районів: нова механіка "районів", які будуються окремо від центру міста, вимагає тактичного планування на карті.
- Дослідження технологій та культури: дві паралельні гілки розвитку — наукова та культурна — відкривають нові будівлі, юніти та форми управління.
- Лідери та цивілізації: кожна нація має унікального лідера з власними бонусами, юнітами, будівлями та стилем гри.
- Дипломатія та релігія: можна укладати союзи, вести шпигунство, розповсюджувати релігію, брати участь у голосуваннях в ООН.
- Політика: система урядів та політичних карт дозволяє налаштовувати стиль правління відповідно до обраної стратегії.

Переваги Civilization VI:

- Глибока стратегічна складова з великою кількістю варіантів для перемоги.
- Яскрава стилізована графіка, зручний інтерфейс та інформативна мапа.
- Великий вибір націй та лідерів з різноманітними стилями гри.
- Можливість грати соло або в мультиплеєрі.
- Постійна підтримка від розробників: DLC, оновлення, нові сценарії та механіки.

Недоліки Civilization VI:

- Тривалий час однієї партії — повноцінна гра може займати десятки годин.
- У пізній грі спостерігається зниження динаміки та "ефект сніжної кулі", коли лідер з перевагою майже гарантовано перемагає.
- AI-супротивники не завжди діють логічно або ефективно.
- Повна версія гри з усіма DLC має високу вартість, що може відштовхнути новачків.

1.3 Огляд навчання з підкріпленням в іграх або дослідженні у віртуальних симуляціях

1.3.1 RL у відеоіграх

Навчання з підкріпленням (RL) активно використовується в розробці ігор для створення інтелектуальних агентів, які можуть адаптуватися до змінних умов, приймати стратегічні рішення та діяти самостійно в складних середовищах. RL дозволяє навчати агента на основі взаємодії з середовищем, де агент отримує винагороди за корисні дії і штрафи за небажані дії.

Ключові приклади:

- OpenAI Five (Dota 2)[13]

  Командна RL-система, навчена грати у гру Dota 2, з використанням Proximal Policy Optimization (PPO). Агент досяг рівня гри вищого за професійних гравців. Використовувалося self-play — тренування

агентів один проти одного для підвищення навичок.

- AlphaStar (StarCraft II)[14]

  Розроблений компанією DeepMind агент, який грав проти найкращих гравців у StarCraft II. Навчання проходило через комбінування RL з імітаційним навчанням і багатоступеневим самонавчанням.

- Unity ML-Agents Toolkit[15]

  Бібліотека для інтеграції RL-алгоритмів у Unity-ігри. Дозволяє створювати навчальні сценарії для NPC, наприклад: пересування, ухилення від ворогів, реагування на дії гравця.

Застосування RL у відеоіграх:

- Адаптивні NPC, які вчаться змінювати поведінку на основі стилю гри користувача.
- Створення суперників у симуляціях.
- Балансування геймплею через симуляцію мільйонів сценаріїв.

1.3.2 RL у віртуальних симуляціях

Віртуальні симуляції надають безпечне, кероване середовище для експериментів з RL. В такому середовищі моделі можуть тренуватися в тисячі або мільйони разів швидше, ніж у реальному світі, що є критично важливим для таких сфер, як робототехніка, автономні системи та поведінкове моделювання.

Приклади платформ і застосувань:

- MuJoCo (Multi-Joint dynamics with Contact):[16]

  Фізичний симулятор для моделювання роботів. RL використовується для навчання агентів рухатись, балансувати, ухилятися.

- Isaac Gym (NVIDIA):[17]

  Високопродуктивна платформа симуляції на GPU для навчання з використанням RL. Застосовується в робототехніці для тренування маніпуляторів, пересування, grasping-операцій.

- CARLA Simulator (автономне водіння):[18]

  RL використовується для навігації в міському середовищі, уникнення аварій, прийняття рішень у складних дорожніх ситуаціях.

- Unity Simulation Pro:

  Платформа для створення великих віртуальних середовищ, де агенти можуть проходити тренування на основі RL. Використовується в освіті, дослідженнях, ігорній індустрії.

Переваги використання віртуальних симуляцій:

- Безпечне тренування в критичних ситуаціях без ризику.
- Прискорене тренування завдяки паралельній симуляції.

- Можливість створення умов, які важко відтворити в реальному світі.
- Легкий контроль над параметрами середовища.

ВИСНОВОК ДО РОЗДІЛУ 1

В даному розділі було виконано короткий огляд предметної області, а після цього розглянено існуючі рішення ігрових застосунків у жанру стратегії та огляд використання алгоритмів RL технології ШІ, щоб покращити ігровий досвід користувача.

Серед розглянутих прикладів ігор у жанрі стратегії, найближчим за геймплеєм є RimWorld з його механіками розширеного будівництва, мікроменеджментом та генерацією випадкових подій.

Для створення ШІ для ворогів буде використано Unity ML-Agents Toolkit, через його доступність та відносну простоту використання.

Також треба зауважити, що через те, що агент як і середовище будуть знаходитися у 2D просторі, це значно підвищує оптимізацію гри та полегшує передачу спостережень агенту.

РОЗДІЛ 2. АНАЛІЗ ЗАСОБІВ РОЗРОБКИ

2.1 Ігрові рушії

Основою розробки багатьох сучасних ігор є ігрові рушії оскільки вони пришвидшують та полегшують розробку.

Ігровий рушій - це програмний засіб, який забезпечує основні функції для створення та запуску відеоігор. Він виконує рендеринг графіки, обробку вводу користувача, керує фізикою, звуком, а також надає інструменти для пришвидшеної розробки різних аспектів гри, що дозволяє розробникам приділити більше уваги створенню контенту та розробці геймплею [19].

Використання готового рушія значно прискорює розробку ігрового застосунку, порівняно з розробкою такого двигуна вручну. Найбільш відомі відкриті приклади ігрових рушіїв - Unreal Engine, Unity та Godot [19].

Також існує багато інших рушіїв які не будуть тут розглядатися, оскільки вони мало популярні або закриті, або мають вузьку спеціалізацію. Є також варіант написання рушію з нуля, але це не є доцільним через велику кількість затраченого часу.

2.1.1 Ігровий рушій Unity

Unity - це один з найбільш популярних безкоштовних рушіїв для розробки кросплатформенних ігор. Рушій працює на понад 17 платформах. 70% найкращих Android-ігор виконуються саме на Unity [20].

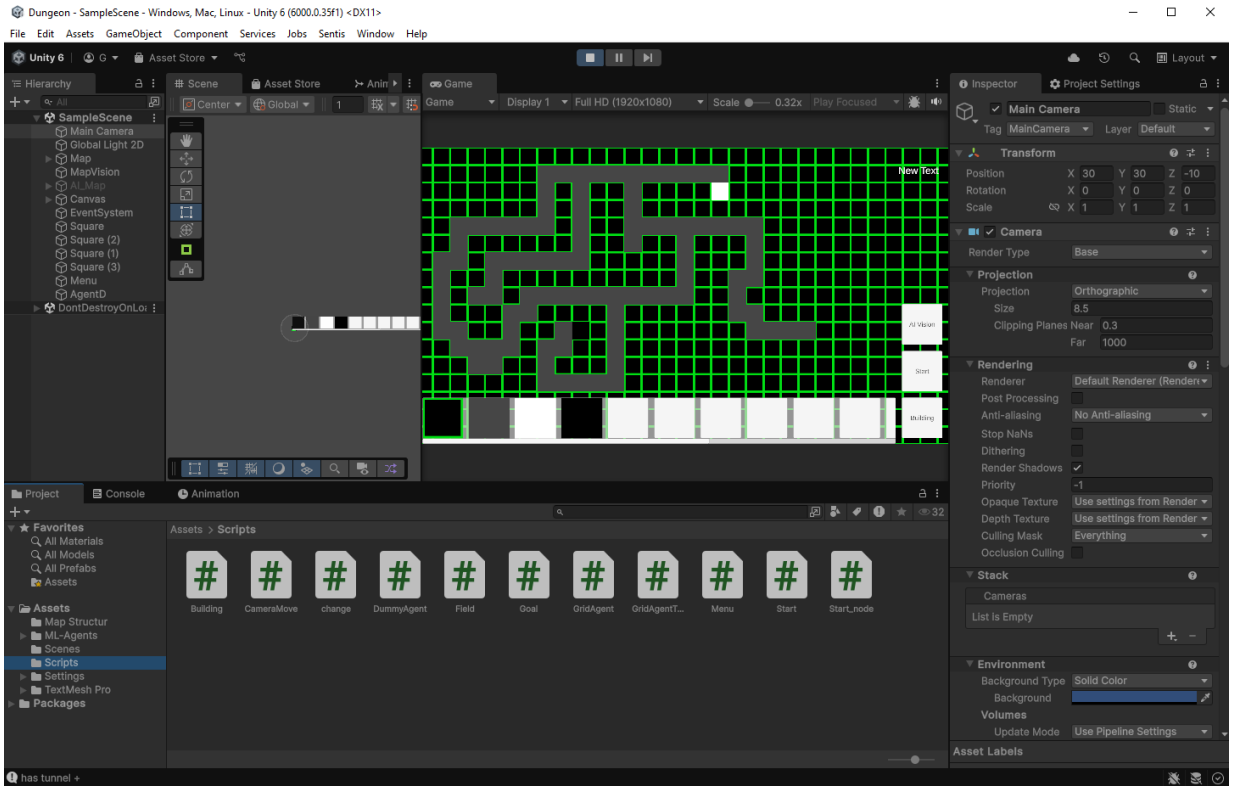


Рисунок 2.1 - Інтерфейс рушія Unity

Unity відомий перш за все великою навчальною базою, широким ком'юніті, великою кількістю готового ігрового контенту та простим інтерфейсом який піддається змінам. Недоліками вважається оптимізація у великих проектах та графіка, але насправді обидві проблеми легко вирішуються доповненнями.

Проте, через деякі зміни в договорі користувача у розділі пов'язаному з прибутком, деякі розробники перешли на інші рушії здебільшого на Unreal Engine та Godot, які конкурентними по відношенню до Unity.

2.1.3 Unreal Engine

Unreal Engine відомий своїми проривними технологіями в графіці, і особливо корисним він є для великих 3D-проектів з фотореалістичною

графікою. Оптимізація також добре себе показує у проектах із великими сценами, великою кількістю об’єктів, тощо. В плані графіки Unreal Engine є найкращим рушієм [21].

Рисунок 2.2 - Unreal Engine 5 та гра Unrecord [22]

На наведеному рисунку можемо побачити, що фотореалізм гри Unrecord, зробленої на Unreal Engine 5, вражає, бо коли був опублікований геймплей цієї гри без виду редактора деякі люди думали що це справжній запис з нагрудної камери. Ще більше вражають анімації та геймплей цієї гри, демонстрацію яких можна побачити на офіційному сайті.

Але Unreal Engine має великий поріг входження та використовує не саму просту мову програмування C++, також можна підмітити чималі кількість необхідних системних ресурсів. Хоча в цьому рушії і є засіб візуального програмування, але він не дуже читабельний та оптимізований для великих проектів

2.1.4 Godot Engine

Godot є безкоштовним кросплатформним рушієм, який орієнтований на простоту у використанні [23]. Він дозволяє обирати зручну для розробника мову програмування – GDScript - спеціально розроблена для двигуна об’єктно-орієнтована мова яка схожа на Python [24], C#, C++, або візуальне програмування.

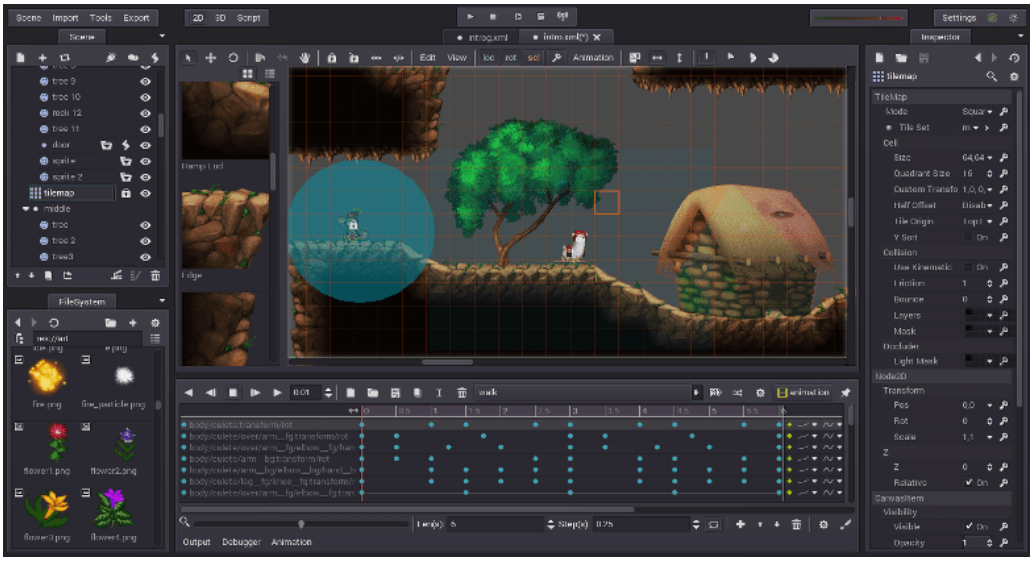


Рисунок 2.3 - Інтерфейс рушія Godot

Через невелику спільноту Godot має невелику кількість навчальних матеріалів, плагінів та інструментів у порівнянні з Unity та Unreal Engine.

2.1.5 Вибір рушія та його додаткових технологій

Після огляду кількох рушіїв та їх додаткових інструментів, було вирішено обрати рушій Unity. Причиною обрання цього рушія були:

- По перше через те, що за темою проекту треба створити 2D-агент, не доцільно використовувати Unreal Engine через його ресурсо затратність у порівнянні з розробкою 2D ігор на Unity або Godot

- По друге, через велику кількість доповнень, плагінів, інструментів та готового контенту для ігор в Unity, а особливо через доповнення Unity ML-Agents, що є open-source проектом, який дозволяє використовувати гру як віртуальне середовище для навчання інтелектуального агента. Він дає змогу використовувати імплементовані ШІ алгоритми на базі PyTorch, що дозволяє легко тренувати агентів для різних задач у 2D чи 3D середовищі.

- По третє, в інтернеті можна знайти багато матеріалів з інформацією як налаштовувати і використовувати ML-Agents в Unity, оскільки для створення ШІ-агента треба використовувати навчання з підкріпленням.

- По четверте, маю певний досвід в роботі з цим рушієм.

Через всі ці причини я і обрав Unity для виконання поставленого завдання.

2.1.6 Вибір технологій машинного навчання

Як вже було згадано, відповідною технологією машинного навчання агентів в обраному рушії є Unity ML-Agents.

Існує велика кількість методів навчання інтелектуального агента. Вони поділяються на навчання під наглядом (supervised learning), без нагляду (unsupervised learning) та навчання з підкріпленням [25]. В роботі буде використано навчання з підкріпленням.

За замовчуванням, ML-Agents використовує алгоритм PPO (Proximal Policy Optimization). PPO - це метод навчання з підкріпленням загального призначення, перевагою якого є стабільність у порівнянні з багатьма іншими алгоритмами навчання з підкріпленням [26]. PPO намагається встановити баланс між простотою імплементації, складністю зразків та простотою

відладки. Стабільність забезпечується тим, що під час оновлення моделі та мінімізації функції втрат, алгоритм забезпечує політику поведінки яка не сильно відрізняється від минулої [27].

Альтернативою до PPO є SAC, який є ефективнішим при використанні меншої кількісті зразків вибірки, тому краще підходить для складних проектів, де крок симуляції займає довше часу. В порівнянні з PPO, часто потребує в 5-10 разів менше зразків [26].

Під час розробки буде використовуватися алгоритм PPO через його стабільність. Також агент буде навчатися поступово з підвищенням рівня складності шляхом додавання різноманітних перешкод та зміни будови лабіринта.

2.2 Вибір джерел контенту гри

Оскільки я створюю 2D гру, що є набагато простішою задачею ніж 3D проекти, оскільки саме створення 3D-моделей вимагає більшу кількість часу і знань, то більшу частину текстур я планую створювати самостійно за допомогою графічних редакторів. Також такий контент можна імпортувати з різних інтернет ресурсів. Те саме можна сказати і на рахунок іншого контенту, такого як: звуки, матеріали, анімації, персонажі, ефекти тощо.

Асет - це термін в розробці ігор, що означає будь-який контент, який може бути використаний для створення ігрового світу та його ігрового процесу.

В Unity для імпортування вже готових асетів можна використовувати Unity Asset Store, де можна придбати або завантажити безкоштовно вже готові асети, але зазвичай такі асети використовуються як навчальні матеріали, а не як матеріали для розробки бо зазвичай не відповідають бажанням розробника.

В малих командах, якщо немає людей з навичками дизайнерів і художників зазвичай використовують вже готові асети або з недавнього часу стали використовувати генерацію контенту за допомогою нейронних мереж.

Оскільки я хочу щоб моя гра була стилізована в піксельному стилі, а сам я маю деякий досвід в створенні такого контенту. Тож створення персонажів, текстур та анімацій буде проходити в ручну з використанням відповідних інструментів, а решта контенту такого, як звуки та музика буде взята з відкритих джерел.

2.3 Вибір засобів створення контенту

Оскільки я планую використовувати піксельний стиль давайте розглянемо кілька інструментів які можна використовувати:

Aseprite вважається одним із найкращих інструментів для створення піксель-арту та анімації. Програма спеціально розроблена для художників, що працюють у ретро-стилі, і надає зручний інтерфейс, підтримку таймлайну, верств і ефектів, а також можливість експортувати спрайт-листи для інтеграції з ігровими рушіями, зокрема Unity[28].

Переваги:

• Інтерфейс оптимізований під піксель-арт

• Повна підтримка анімації

• Експорт у формати PNG, sprite sheets, GIF

• Плагіни для автоматичного імпорту в Unity

Недоліки:

• Платний (одноразова покупка ~$20)

• Безкоштовна версія — лише через компіляцію з GitHub

Piskel — це безкоштовний онлайн-інструмент для створення піксельної графіки та анімації. Його простий інтерфейс робить його доступним навіть для новачків. Програма дозволяє експортувати спрайти у форматах PNG та sprite sheets, що дає змогу легко використовувати її результати в Unity[29].

Переваги:

• Повністю безкоштовний

• Онлайн-доступ з будь-якого браузера

• Простий інтерфейс для анімації

• Можливість збереження локально та в хмарі

Недоліки:

• Відсутність складніших функцій редагування

• Обмежена підтримка гарячих клавіш та шарів

Pixilart — це онлайн-редактор для створення піксель-арту з елементами соціальної платформи. Він дозволяє створювати зображення в браузері, ділитися ними та зберігати у форматі PNG, що робить його зручним для початкової графіки, яку можна використовувати в Unity[30].

Переваги:

• Безкоштовний і доступний з браузера

• Соціальна платформа з галереєю

• Проста екосистема для створення статичних спрайтів

Недоліки:

• Відсутність підтримки анімації на рівні sprite sheets

• Мінімальні можливості для складного редагування

GraphicsGale — це легка програма для Windows, орієнтована на створення піксель-арту та анімації. Підтримує шари, кадрування, прозорість і експортування в анімаційні формати, що робить її зручною для роботи зі спрайтами для ігор[31].

Переваги:

• Проста у використанні для 2D-анімації

• Підтримка таймлайну і попереднього перегляду

• Працює на слабких комп’ютерах

Недоліки:

• Тільки для Windows

• Застарілий інтерфейс

• Менше оновлень та нових функцій у порівнянні з Aseprite

Photoshop / GIMP — це універсальні графічні редактори, які, хоч і не спеціалізуються на піксель-арті, все ж можуть бути налаштовані для роботи у цьому стилі за допомогою сітки, твердих пензлів і режиму масштабування без згладжування. Вони дозволяють створювати спрайти високої точності та експортувати їх у PNG-формат для Unity[32, 33].

Переваги:

• Потужний функціонал редагування зображень

• Підтримка шарів і масок

• Можливість роботи з великими проектами

Недоліки:

• Не спеціалізовані на піксель-арті

• Photoshop — платний, GIMP — складніший в освоєнні

• Відсутня вбудована підтримка sprite sheets

З усіх розглянутих інструментів я обрав Photoshop оскільки маю певний досвід в створенні за допомогою нього піксель-арті. А також різноманітних уроків по ньому в інтернеті.

ВИСНОВОК ДО РОЗДІЛУ 2

В даному розділі було розглянено різні інструменти та технології, які буде використано під час розробки гри. Ігровий рушієм було обрано Unity через його простоту, наявний досвід роботи з ним та наявну технологію ML-Agents, що допоможе інтегрувати ШІ-агентів у гру. Як мову програмування, буде використано C#, який інтегрований з рушієм Unity.

Як інструмент для створення піксель-арті, було обрано Adobe Photoshop через наявний в мене досвід та легкість в налаштуванні його для більш зручного використання під цю задачу.

РОЗДІЛ 3. РЕАЛІЗАЦІЯ ПРОГРАМНОГО ПРОДУКТУ

3.1 Програмна реалізація ігрових механік

У стратегіях гравець зазвичай спостерігає за ігровим полем з верху ніби літаючи над ним. Через інколи великі розміри карт та необхідність наближення камери для більш точного управління. Тож в першу чергу треба додати можливість гравцю пересувами та масштабувати камеру і зробити так щоб вона не могла вийти за межі ігрового поля.

У рамках мого проекту реалізовано систему керування камерою, яка забезпечує плавне переміщення по ігровій сцені, а також змінює масштаб зображення у відповідь на прокручування колеса миші. Основою алгоритму є обробка вхідних даних з клавіатури та миші, які трансформуються у вектор швидкості переміщення. Масштабування реалізовано шляхом зміни параметра zoom, яке безпосередньо впливає на розміри камери та колайдера задача якого в цьому випадку обмежувати камеру для запобігання виходу за межі ігрової зони. Це реалізовано шляхом зіткнення з колайдерами розташованими на межах цієї зони.

```
void Update()
{
    Vector2 InpumMove = new Vector2(Input.GetAxisRaw("Horizontal"), Input.GetAxisRaw("Vertical"));
    MoveVelocity = InpumMove * speed;
    float scroll = Input.GetAxis("Mouse ScrollWheel");
    if (zoom + scroll > 1 && zoom + scroll < 3)
    {
        zoom += scroll;
        Debug.Log("scroll");
    }
    if (zoom < 1)
    {
        zoom = 1;
    }
    if (zoom > 3)
    {
        zoom = 3;
    }
}
Сообщение Unity | Ссылок: 0
private void FixedUpdate()
{
    body.MovePosition(body.position + MoveVelocity * Time.deltaTime);
    m_BoxCollider.size = new Vector2(19.2f / zoom, 10.8f / zoom);
    m_Camera.orthographicSize = 8.5f / zoom;
}
```

Рисунок 3.1 - Фрагмент коду для керування камерою

Нижче наведені приклади виду камери при максимальному та мінімальному наближенні в різних частинах карти.

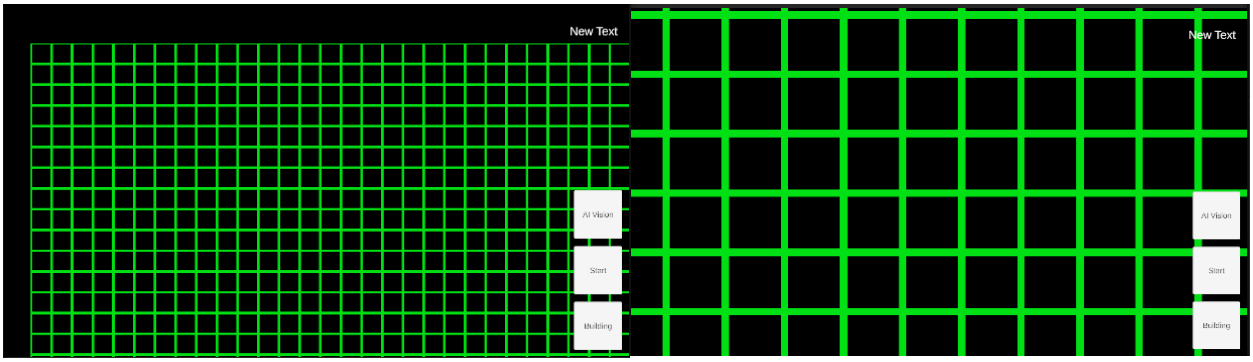


Рисунок 3.2 – Вид камери при мінімальному та максимальному наближенні

Наступна механіка яку я реалізував це будівництво. При запуску нової гри створюється порожня карта заповнена стінами, або при завантаженні збереження відтворюється карта яку була збережена. У кожної клітинки на цій карті є такі характеристики як ID об'єкту який там знаходиться, посилання на цей об’єкт в редакторі, щоб з ним можна було легше взаємодіяти, чи дослідив ШІ цю клітинку та чи відвідував її. Саме будівництво відбувається шляхом обрання якогось об’єкту з палітри структур, перевірки наявності необхідних матеріалів для будівництва та зміною параметрів клітинки під обрану.

```
for (int i = 0; i < 50; i++)
{
    for (int j = 0; j < 50; j++)
    {
        Map[i][j] = new Field();
        Map[i][j].x = i;
        Map[i][j].y = j;
        Map[i][j].id = 1;
        Map[i][j].structures = Instantiate(structuresBase[0], new Vector3(i, j, 0), Quaternion.identity);
        Map[i][j].structures.name = structuresBase[0].name;
        Map[i][j].structures.transform.SetParent(map_list.transform);
        Map[i][j].HaveChild = false;
        Map[i][j].visit = false;
        Map[i][j].know = false;

        AI_Map[i][j] = Instantiate(structuresBase[3], new Vector3(i, j, -1), Quaternion.identity);
        AI_Map[i][j].name = structuresBase[3].name;
        AI_Map[i][j].transform.SetParent(AI_list.transform);

        mapAI_Map[i][j] = 0;
    }
}
```

Рисунок 3.3 – Фрагмент коду який відповідає за заповнення порожньої карти

```
private void Update()
{
    if (Input.GetKey(KeyCode.Mouse0) && Building_activ)
    {
        Vector3 screen_position = Camera.main.ScreenToViewportPoint(Input.mousePosition);
        Vector3 world = Camera.main.ScreenToWorldPoint(Input.mousePosition);
        Vector3 WorldFix = new Vector3(Mathf.RoundToInt(world.x), Mathf.RoundToInt(world.y), 0);
        if ((WorldFix.x > -0.5f && WorldFix.x < 49.5f) && (WorldFix.y > -0.5f && WorldFix.y < 49.5f) && screen_position.y > 0.2f)
        {
            int x = (int)WorldFix.x;
            int y = (int)WorldFix.y;
            if (Map[x][y].id != build_id + 1)
            {
                if (build_id == 0)
                {
                    Debug.Log("-");
                    Destroy(Map[x][y].structures);
                    Map[x][y].structures = Instantiate(structuresBase[0], new Vector3(x, y, 0), Quaternion.identity);
                    Map[x][y].structures.name = structuresBase[0].name;
                    Map[x][y].structures.transform.SetParent(map_list.transform);
                    Map[x][y].id = 1;
                    Map[x][y].HaveChild = false;
                }
                else if (build_id == 1)
                {
                    Debug.Log("+");
                    Destroy(Map[x][y].structures);
                    Map[x][y].structures = Instantiate(structuresBase[1], new Vector3(x, y, 0), Quaternion.identity);
                    Map[x][y].structures.name = structuresBase[1].name;
                    Map[x][y].structures.transform.SetParent(map_list.transform);
                    Map[x][y].id = 2;
                    Map[x][y].HaveChild = false;
                }
                else
                {
                    if (Map[x][y].id == 2)
                    {
                        if (Map[x][y].HaveChild)
                        {
                            if (Map[x][y].id != build_id + 1)
                            {
                                Debug.Log("has tunnel -");
                                Destroy(Map[x][y].structuresChild);
                                Map[x][y].structuresChild = Instantiate(structuresBase[build_id], new Vector3(x, y, -0.1f), Quaternion.identity);
                                Map[x][y].structuresChild.name = structuresBase[build_id].name;
                                Map[x][y].structuresChild.transform.SetParent(Map[x][y].structures.transform);
                                Map[x][y].HaveChild = true;
                                Map[x][y].id = build_id + 1;
                            }
                        }
                        else
                        {
                            Debug.Log("has tunnel +");
                            Map[x][y].structuresChild = Instantiate(structuresBase[build_id], new Vector3(x, y, -0.1f), Quaternion.identity);
                            Map[x][y].structuresChild.name = structuresBase[build_id].name;
                            Map[x][y].structuresChild.transform.SetParent(Map[x][y].structures.transform);
                            Map[x][y].HaveChild = true;
                            Map[x][y].id = build_id + 1;
                        }
                    }
                }
            }
        }
    }
```

Рисунок 3.4 – Фрагмент коду який відповідає за перевірку та заміну структур

Також треба сказати що для визначення з якою клітинкою взаємодіяти код перевіряє позицію курсора.

Одна з найважливіших речей у будь якій грі це можливість зберігати та завантажувати свій прогрес проходження. Для збереження або завантаження гри користувачу треба поставити гру на паузу, за допомогою кнопки перейти в відповідне меню де при збереженні треба вказати назву під якою файл буде

збережено, а при завантаженні обрати необхідний файл. Також було реалізовано оновлення списку збережених файлів.

```
public static class SaveManager
{
    Ссылок: 1
    public static void Save<T>(string fileName, T saveData)
    {
        string saveFolder = Application.persistentDataPath + "/Saves/" + fileName;
        if (!Directory.Exists(saveFolder))
            Directory.CreateDirectory(saveFolder);

        string fullPath = Path.Combine(saveFolder, fileName + ".json");

        string jsonDataString = JsonUtility.ToJson(saveData, true);
        File.WriteAllText(fullPath, jsonDataString);

        Debug.Log("Saved to " + fullPath);
    }

    Ссылок: 1
    public static T Load<T>(string fileName) where T : new()
    {
        string saveFolder = Application.persistentDataPath + "/Saves/" + fileName;
        string fullPath = Path.Combine(saveFolder, fileName + ".json");

        if (!File.Exists(fullPath))
        {
            Debug.LogError("Save file not found: " + fullPath);
            return new T();
        }
        string jsonDataString = File.ReadAllText(fullPath);
        T Data = JsonUtility.FromJson<T>(jsonDataString);
        return Data;
    }
}
```

Рисунок 3.5 – Фрагмент коду який відповідає за збереження та завантаження

```
public void GetSaveFiles(int id)
{
    string saveFolder = Application.persistentDataPath + "/Saves";

    if (Directory.Exists(saveFolder))
    {
        string[] files = Directory.GetDirectories(saveFolder);

        for (int i = 0; i < files.Length; i++)
        {
            files[i] = Path.GetFileNameWithoutExtension(files[i]);
        }
        Debug.Log(saveFolder);
        Debug.Log(files.Length);
        SaveList[id].GetComponent<RectTransform>().SetSizeWithCurrentAnchors(RectTransform.Axis.Vertical, 110 * files.Length);

        for (int j = 0; j < files.Length; j++)
        {
            GameObject button = Instantiate(ButtonSave);
            button.name = files[j];
            button.transform.SetParent(SaveList[id].transform);
            TextMeshProUGUI text = button.GetComponentInChildren<TextMeshProUGUI>();
            text.text = files[j];
            int index = j;
            button.GetComponent<UnityEngine.UI.Button>().onClick.AddListener(() => Loaded(files[index]));
        }
    }
}
```

Рисунок 3.6 – Фрагмент коду який відповідає за оновлення списку збережень

3.2 Реалізація ігрового ШІ

Як вже було зазначено у попередніх розділах для реалізації поведінки ворога буде використовуватися навчання з підкріпленням і для цього буде використана технологія Unity ML Agents.

Гіперпараметри моделі та методу навчання слід налаштувати у файлі конфігурації поведінки, яку використовуватиме алгоритм навчання ML Agents. Приклад такого файлу я наводжу далі:

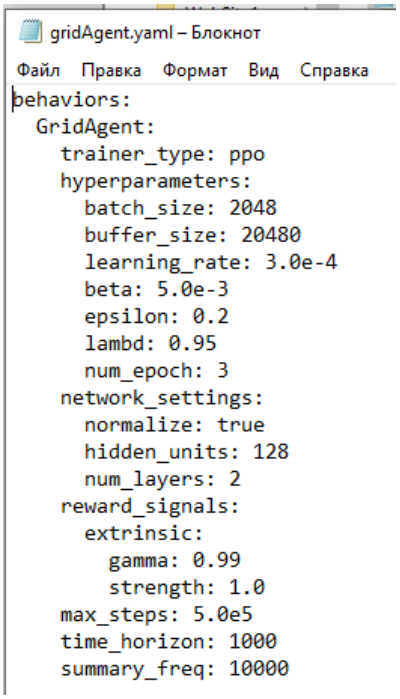


Рисунок 3.7 - Конфігурація поведінки агента

З попереднього рисунку видно, що розмір моделі поведінки агента задана як 2 слої та 128 нейронів у слої. ML Agents також неявно використовує для нейронів функцію активації Swish, тобто аргумент, помножений на синус від нього ж.

Агент діє, базуючись на своїх знаннях та спостереженні з навколишнього світу. Знання агента містяться в параметрах моделі та пам’яті. Розмір пам’яті та моделі визначаються конфігурацією поведінки.

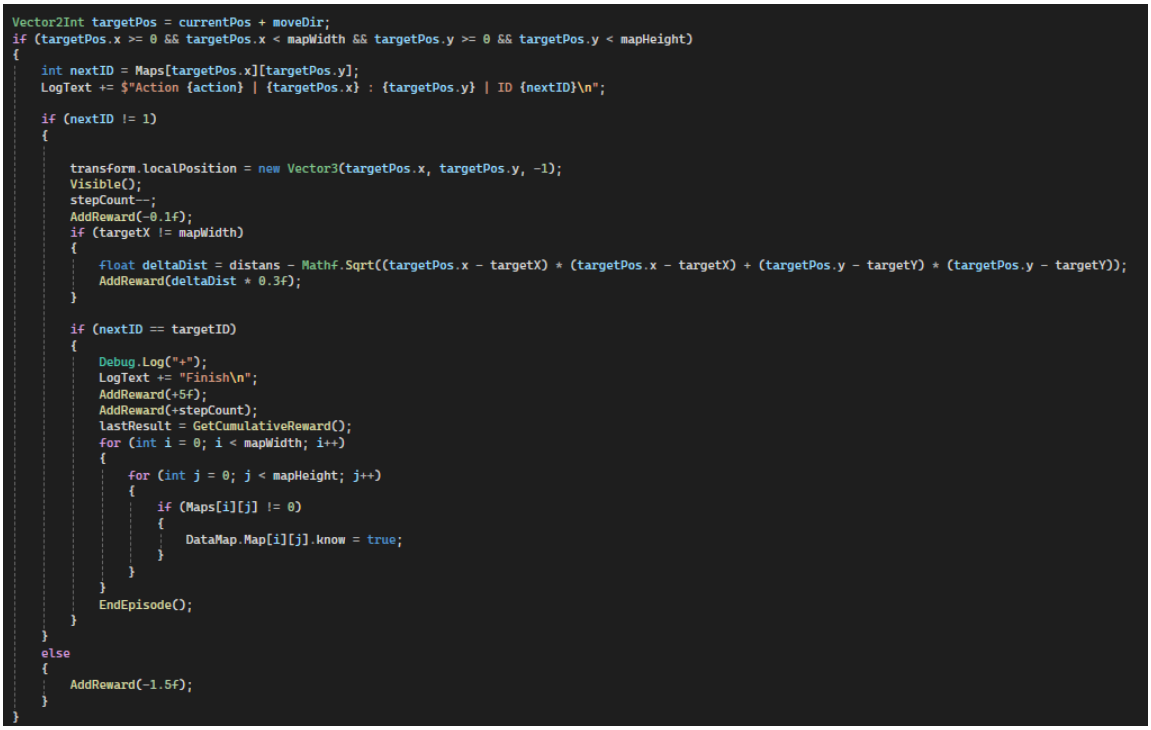
```
public override void CollectObservations(VectorSensor sensor)
{
    Visible();
    float agentPosX = transform.localPosition.x;
    float agentPosY = transform.localPosition.y;
    for (int i = 0; i < 99; i += 3)
    {
        int id = View[i];
        float[] embedding = embeddingLookup.ContainsKey(id) ? embeddingLookup[id] : embeddingLookup[0];
        sensor.AddObservation(embedding[0]);
        sensor.AddObservation(embedding[1]);
        sensor.AddObservation((float)View[i + 1] / mapWidth);
        sensor.AddObservation((float)View[i + 2] / mapWidth);
    }
    sensor.AddObservation(agentPosX / mapWidth);
    sensor.AddObservation(agentPosY / mapHeight);
    float[] targetEmbedding = embeddingLookup.ContainsKey(targetID) ? embeddingLookup[targetID] : embeddingLookup[0];
    sensor.AddObservation(targetEmbedding[0]);
    sensor.AddObservation(targetEmbedding[1]);
    distans = float.MaxValue;
    float targetPosX = 1;
    float targetPosY = 1;
    targetX = mapWidth;
    targetY = mapHeight;
    for (int i = 0; i < mapWidth ; i++)
    {
        for (int j = 0; j < mapHeight ; j++)
        {

            if (Maps[i][j] == targetID && distans > Mathf.Sqrt((agentPosX - i) * (agentPosX - i) + (agentPosY - j) * (agentPosY - j)))
            {
                targetPosX = (float)i / mapWidth;
                targetPosY = (float)j / mapHeight;
                distans = Mathf.Sqrt((agentPosX - i) * (agentPosX - i) + (agentPosY - j) * (agentPosY - j));
                targetX = i;
                targetY = j;
            }
        }
    }
    if (distans == float.MaxValue)distans = 0;
    sensor.AddObservation(targetPosX);
    sensor.AddObservation(targetPosY);
}
```

Рисунок 3.8 - Фрагмент коду обчислення значення сенсорів агента

В моєму випадку агент отримує данні про клітинки в його полі зору, яке є у формі хреста з агентом у центру і променями довжиною в три клітинки і з боковими клітинками до цих, у всі чотири сторони. Також він отримує дані про свою позицію, ID його цілі та її позицію про знаходженні.

В процесі тренування агента, алгоритм навчання намагається оптимізувати метрику нагороди. Агент виконує дії відповідно до своєї попередньої поведінки з деяким випадковим відхиленням, щоб дослідити результат різних варіантів поведінки та визначити, яка поведінка є ефективнішою. Таким чином, якщо правильно призначати нагороди за результат дій агента, він почне вчитися повторювати та покращувати найліпший отриманий результат. Після навчання готова модель використовується в ігровому процесі.



```
Vector2Int targetPos = currentPos + moveDir;
if (targetPos.x >= 0 && targetPos.x < mapWidth && targetPos.y >= 0 && targetPos.y < mapHeight)
{
    int nextID = Maps[targetPos.x][targetPos.y];
    LogText += $"Action {action} | {targetPos.x} : {targetPos.y} | ID {nextID}\n";

    if (nextID != 1)
    {

        transform.localPosition = new Vector3(targetPos.x, targetPos.y, -1);
        Visible();
        stepCount--;
        AddReward(-0.1f);
        if (targetX != mapWidth)
        {
            float deltaDist = distans - Mathf.Sqrt((targetPos.x - targetX) * (targetPos.x - targetX) + (targetPos.y - targetY) * (targetPos.y - targetY));
            AddReward(deltaDist * 0.3f);
        }

        if (nextID == targetID)
        {
            Debug.Log("+");
            LogText += "Finish\n";
            AddReward(+5f);
            AddReward(+stepCount);
            lastResult = GetCumulativeReward();
            for (int i = 0; i < mapWidth; i++)
            {
                for (int j = 0; j < mapHeight; j++)
                {
                    if (Maps[i][j] != 0)
                    {
                        DataMap.Map[i][j].know = true;
                    }
                }
            }
            EndEpisode();
        }
    }
    else
    {
        AddReward(-1.5f);
    }
}
```

Рисунок 3.9 - Фрагмент коду видачі нагород агенту

У моєму випадку агент отримує винагороди за дослідження невідомих клітинок, наближення до цілі, кількість кроків які залишилися та досягнення цілі. Покарання агент отримує за вичерпання допустимої кількості кроків або зіткнення зі стінами або межами карти.

Також були додані невеликі покарання за кожний зроблений агентом крок для прискорення навчання та оптимізації кількості його дій.

Важливою частиною при написанні логіки дії агента, видачі нагород та покарань, є надання агенту простору для самостійного розвитку та знаходження ліпшого шляху вирішення задачі. Через це надмірний контроль над поведінкою агенте не буде нічим відрізнятися від вручну прописаної поведінки, через, що сенсу використовувати ШІ не буде.

Для тренування агентів я використовув декілька середовищ різного рівня складності які були згенеровані випадково.

```
public void Ease()
{
    CleanMap();
    for (int i = 0; i < mapWidth; i++)
    {
        for (int j = 0; j < mapHeight; j++)
        {
            Vector3 LocalPos = new Vector3(startPoint.transform.position.x + i, startPoint.transform.position.y
            Map[i][j].x = i;
            Map[i][j].y = j;
            Map[i][j].id = 2;
            Map[i][j].structures = Instantiate(structuresBase[1], LocalPos, Quaternion.identity);
            Map[i][j].structures.name = structuresBase[1].name;
            Map[i][j].structures.transform.SetParent(startPoint.transform);
            Map[i][j].HaveChild = false;
            Map[i][j].visit = false;
            Map[i][j].know = false;
        }
    }

    Vector2Int start, goal;
    do
    {
        start = new Vector2Int(Random.Range(0, mapWidth), Random.Range(0, mapHeight));
        goal = new Vector2Int(Random.Range(0, mapWidth), Random.Range(0, mapHeight));
    } while (start == goal);

    Destroy(Map[start.x][start.y].structures);
    Vector3 Pos = new Vector3(startPoint.transform.position.x + start.x, startPoint.transform.position.y + start
    Map[start.x][start.y].id = 3;
    StartX = start.x;
    StartY = start.y;
    Map[start.x][start.y].structures = Instantiate(structuresBase[2], Pos, Quaternion.identity);
    Map[start.x][start.y].structures.name = structuresBase[2].name;
    Map[start.x][start.y].structures.transform.SetParent(startPoint.transform);
    Destroy(Map[goal.x][goal.y].structures);
    Pos = new Vector3(startPoint.transform.position.x + goal.x, startPoint.transform.position.y + goal.y, 0);
    Map[goal.x][goal.y].id = 4;
    Map[goal.x][goal.y].structures = Instantiate(structuresBase[3], Pos, Quaternion.identity);
    Map[goal.x][goal.y].structures.name = structuresBase[3].name;
    Map[goal.x][goal.y].structures.transform.SetParent(startPoint.transform);
}
```

Рисунок 3.10 - Фрагмент коду випадкової генерації поля тренування

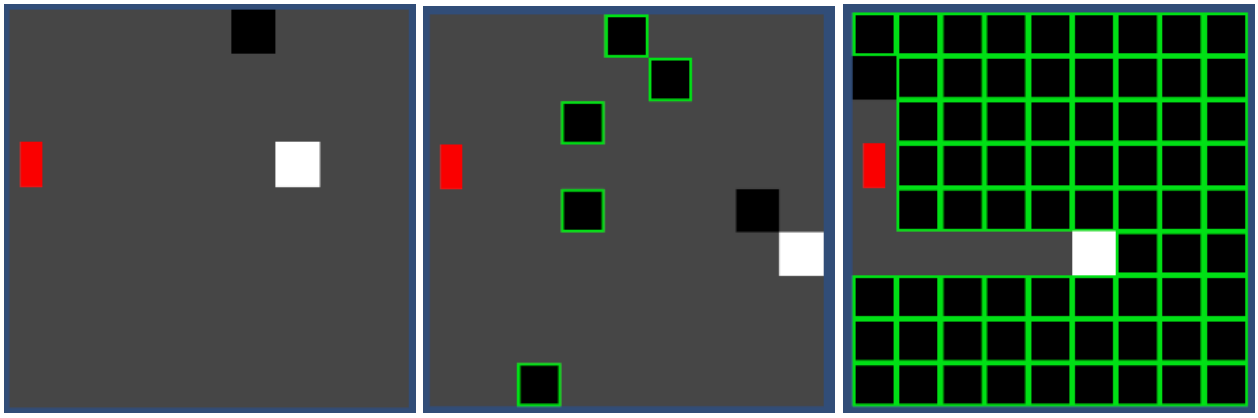


Рисунок 3.11 – Приклади випадково згенерованих рівнів

На першому рівні складность ігент вчився, що за досягнення цілі він отримує нагороду, на другомо рівні складності агент вчився уникати контакту з перешкодами, а не третьому рувні ціль агента було закріапити знання і навчитися пересуватися по тонелях.

З метою навчання моделей Unity спілкується з тренером ML Agents, що використовує Pytorch. Прийняття рішень агента під час та за межами тренування обчислюється за допомогою механізму прийняття рішень – Barracuda.

Рисунок 3.12 – Вигляд консольної програми навчання агентів

3.3 Робота над оптимізацією гри

Оскільки в моєму проекті не використовується складна фізика, а стиль самого проекту 2D, йому не треба багато ресурсів на опрацювання цих данних.

Тож головною проблемою оптимізації гри стає оптимізація споживання ресурсів ШІ. Оскільки у мене в проекті одгочасно може бути активно велика кількість агентів.

Під час розробки агентів я вже виконав певну модифікацію для оптимізації та універсальності агентів. А саме змінив спосів і кількість

спостережень агентом. Оскільки на початку у мене агент отримував інформацію о всіх клітинках карти і через це кількість входів була більше 5000, а оскільки агент має фіксовану структуру спостережень, зміна розмірів карти зробить агент недієвим. Тож я змінив спостереження агента на дільнку навколо нього, що призвело до зменшення кіль спостережень до 138 і можливість працювати не залежно від розмірів мапи.

ВИСНОВОК ДО РОЗДІЛУ 3

В цьому розділі було описано хід роботи над проектом - реалізацію ігрових механік, особливості застосування машинного навчання та роботу над оптимізацією гри.

За допомогою рушія Unity та мови програмування C# було реалізовано ігрові механіки будівництва, пересування та маштабування камери, збереження та завантаження гри.

За допомогою технології ML Agents було створено модель поведінки агента та навчено модель за допомогою методу навчання з підкріпленням.

РОЗДІЛ 4. ОГЛЯД ТА ТЕСТУВАННЯ РОЗРОБЛЕНОГО ПРОДУКТУ

4.1 Огляд розробленої гри

При запусканні гри гравця зустрічає головне меню, де він може перейти в меню налаштування для зміни гучності звуків, перейти до списку збереженнь та обрати бажане, також він може розпочати нову гру або вийти з гри.

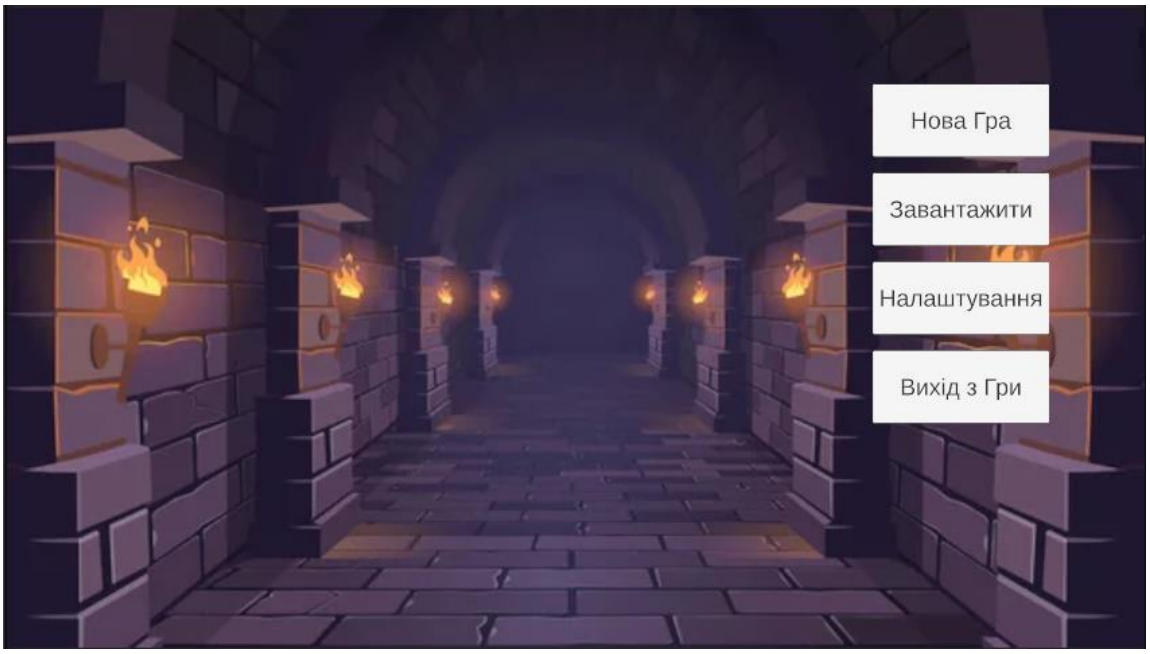


Рисунок 4.1 - Головне меню гри

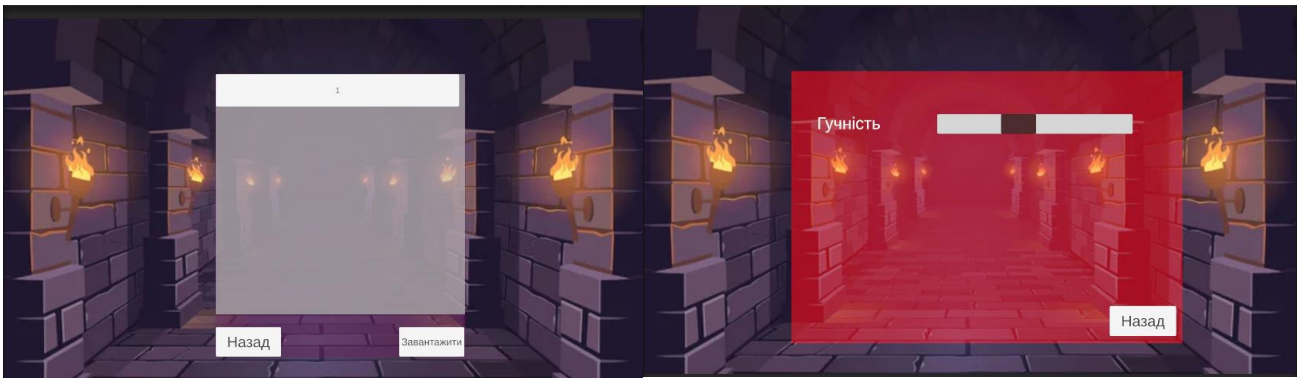


Рисунок 4.2 –список збережень та меню налаштувань

Ціль гри полягає у побудуванні лабіринта(підземелля) з різними перешкодами які буде проходити агент під керуванням ШІ. На данний момент в грі реалізована невелика кількість механік, оскіки це бета версія гри мета якої оцінити і перевірити роботу ключових механік та можливостей ШІ з навчанням з підкріпленням.

При запуске гри гравець побаче карту та інтерфей в якому вказано кількість балів для будівництва, кнопка відкриття меню будівництва та кнопка створення агента при наявності лабіринта.

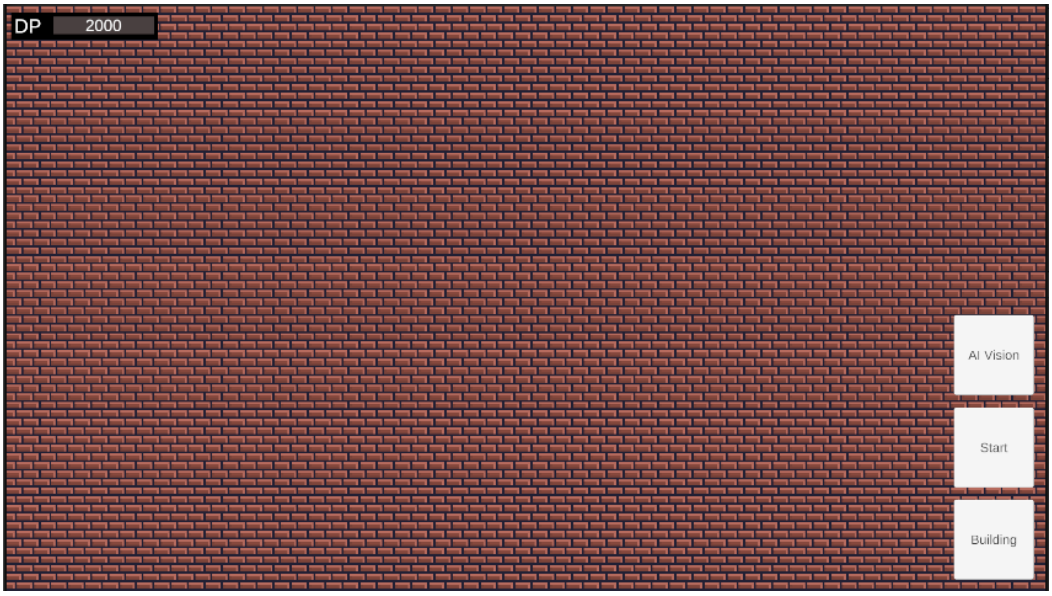


Рисунок 4.3 – вид гри при початку нової гри

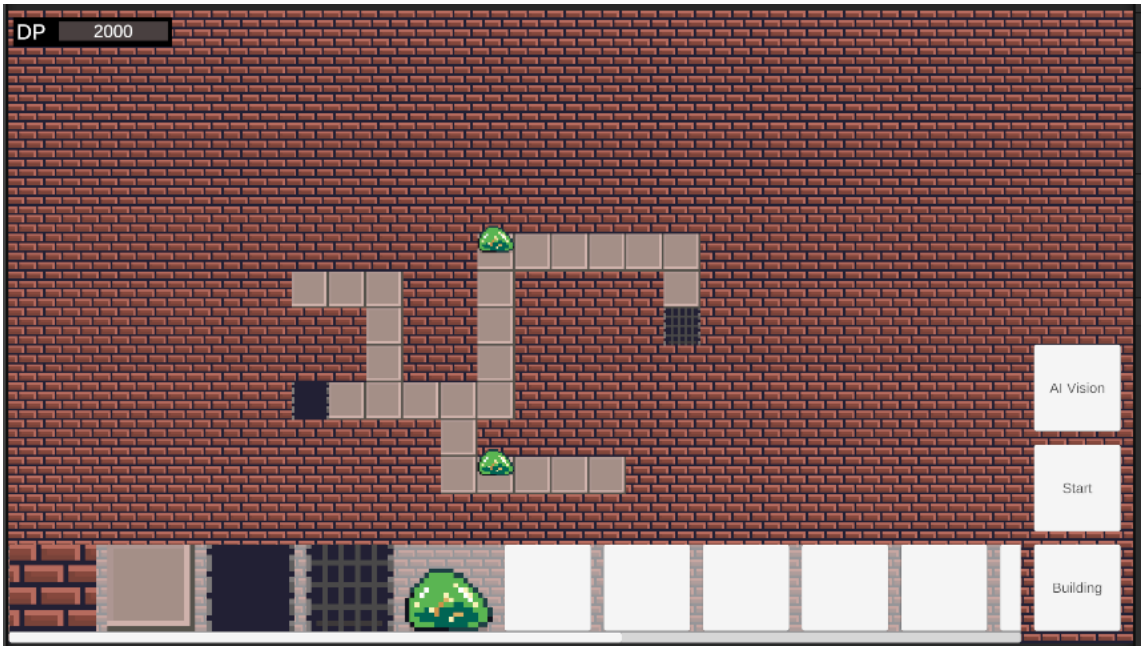


Рисунок 4.4 – меню будівництва та приклад лабіринта

Після створення лабіринта гравець може натиснути кнопку старт для створення агента який буде намагатися пройти лабіринт, досліджуючи його у процесі.

Також гравець може відкрити меню паузи натиснувши кнопку “Esc” де будуть кнопки для продовження гри, відкриття меню налаштувань, завантаження і збереження гри, та виходу до головного меню.

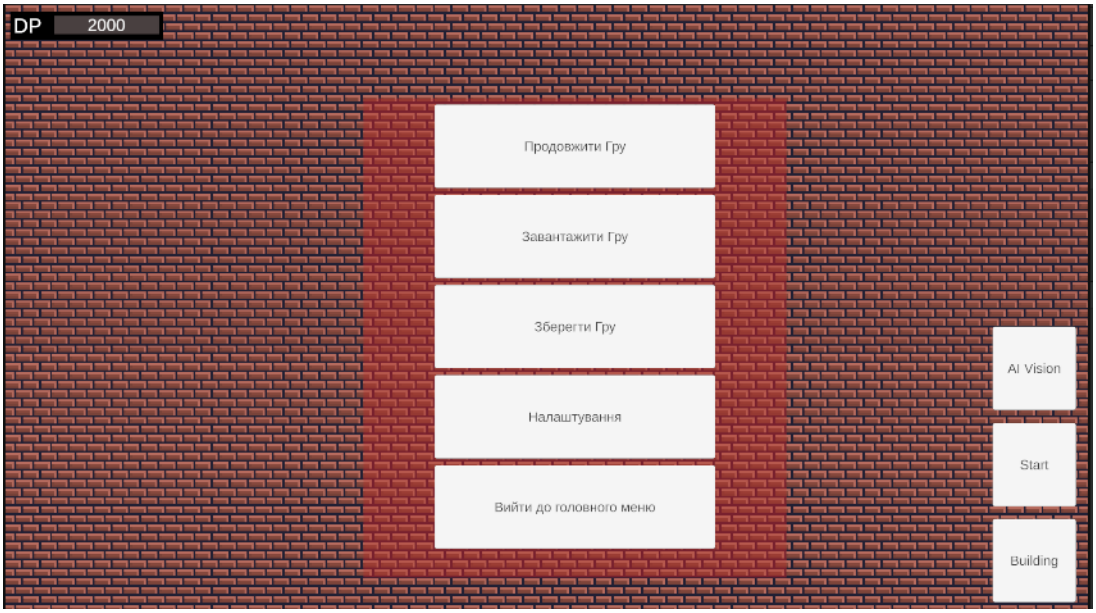


Рисунок 4.5 – меню паузи

При натисканні кнопки зберегти гру перед гравцем відкриється вікно в якому буде список наявних збережень, поле для воду назви під якою буде збережено гру та кнопки для повернення назад і збереження гри.

Рисунок 4.6 – меню збереження гри

Меню для завантаження виглядає майже так само, зокрема воно не має поля для вводу назви для збереження.

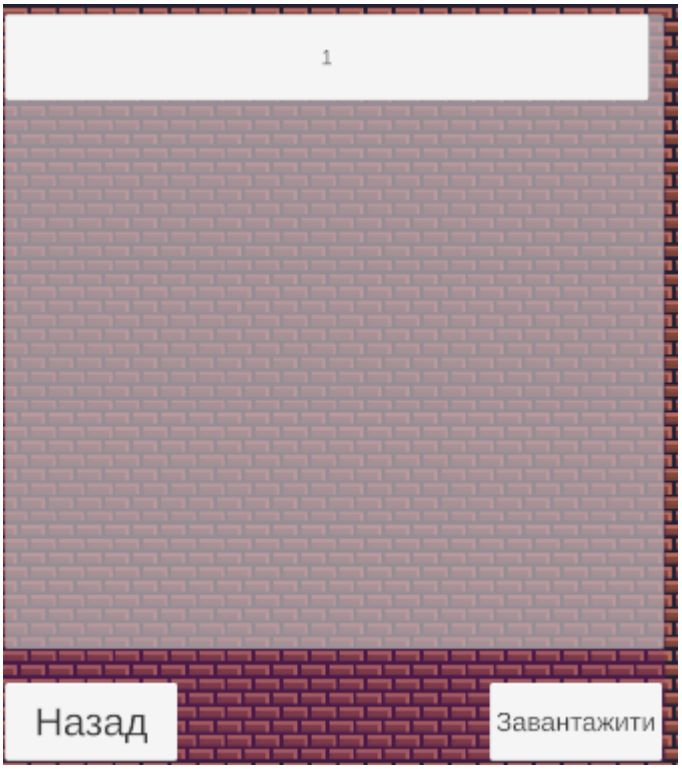


Рисунок 4.6 – меню завантаження гри

Якщо гравець захоче змінити гучність гри то йому треба в меню налаштувань перемістити повзунець гучності та натиснути кнопку застосувати.

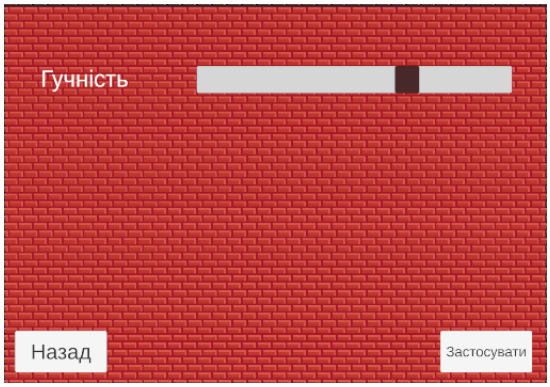


Рисунок 4.7 – меню налаштувань

ВИСНОВОК ДО РОЗДІЛУ 4

В цьому розділі було коротко розглянуто основні механіки гри, перевірено роботу інтерефейса користувача. В даній бета версії гри не було виявлена значних відхилень у роботі коду, але не правильно буде стверджувати про його ідеальну роботу оскільки це тестова версія, в якій досліджується взаємодія ШІ агента з середовищем гри, зручність користування інтерфейсом та кількість споживаємих ресурсів.

Тож створений продукт ще не можна повністю назвати готовою грою оскільки в ній ще за мало контенту та повністю на закінченна оптимізація. І при подальшій розробці майбутня версія гри може сильно відрізнятися від поточної не тільки в дизайнерському плані, але і у поведінці ШІ, оскільки він буде взаємодіяти з більшою кількістю різноманітних даних, або використовувати додаткові алгоритми.

У майбутньому розвидку планується внести зміни у вигляд інтерфейса, щоб він був стилізований під тему гри, також прланується додати різні розділи у меню будівництва такі, як структури та істоти, для легшої навігації. Будуть створені різні версії ШІ-агентів для обрання кращої логіки поведінки та оптимізації.

ВИСНОВОК

В процесі написання дипломної роботи було створено комп’ютерну гру у жанрі стратегії. Де поведінка суперника реалізовано ШІ з навчанням з підкріпленням.

Перед початком розробки, було розглянуто приклади ігор у жанрі стратегії та приклади використання навчання з підкріпленням. Також було розглянуто їх переваги та недоліки, після чого було обрано алгоритм PPO, як найбільш універсальний та стабільний. Його недолік в потребі великої кількості зразків компенсувався створенням великої кількості паралельних тренувальних середовищ.

Частина ігрового контента – текстури та моделі були створені власноруч у Adobe Photoshop, решта ж було завантажено з джерела Unity Asset Store.

Ігровим рушієм було обрано Unity, насамперед як двигун, що є найбільш зручним для реалізації ігрового ШІ, бо має технології ML Agents та Unity Barracuda. Та зручий онлайн-магазин Unity Asset Store де користувачі можуть ділитися між собою своїми ресурсами.

Використовуючі обрані методи я навчив ШІ агента проходити невідомий лабіринт у пошуках необхідною цілі при цьому уникати або знищувати ворогів.

Також гра має голоне меню, меню паузи, меню налаштувань та вікна збереження та завантаження гри.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. What Makes a Good Video Game? 4 Key Elements [Електронний ресурс] // Pluralsight – 2020. – Режим доступу до ресурсу: https://www.pluralsight.com/resources/blog/software-development/what-makes-a-great-game-the-key-elements-of-successful-games
2. All Steam Game Tags [Електронний ресурс]// SteamDBDATABASE – Game tags – Режим доступу до ресурсу: https://steamdb.info/tags/
3. Computer Games [Електронний ресурс] // Nanyang Technological University – Режим доступу до ресурсу: https://www3.ntu.edu.sg/home/asschui/Computer%20Games.PDF
4. Genre and game studies: Toward a critical approach to video game genres [Електронний ресурс] // University of Melbourne – 2006. – Режим доступу до ресурсу: https://journals.sagepub.com/doi/pdf/10.1177/1046878105282278
5. Action games [Електронний ресурс] // AllGame – Режим доступу до ресурсу: https://web.archive.org/web/20141209041709/http://allgame.com/genre.php?id=20
6. Driffle. What Are Real-Time Strategy Games? Explore RTS Genre [Електронний ресурс] // Driffle – Режим доступу до ресурсу: https://driffle.com/blog/what-are-real-time-strategy-games-explore-rts-genre/
7. Gamerant. Games With The Best Base Building Mechanics, Ranked [Електронний ресурс] // Gamerant – Режим доступу до ресурсу: https://gamerant.com/games-best-base-building-mechanics/

8. MasterClass. Tower Defense Game Genre: 6 Characteristics of TD Games [Електронний ресурс] // MasterClass – Режим доступу до ресурсу: https://www.masterclass.com/articles/tower-defense-game-video-game-guide

9. Uncovering the Top 5 Types of Gamers in Today's Gaming Community [Електронний ресурс] // Ganknow – Режим доступу до ресурсу: https://ganknow.com/blog/types-of-gamers/

10. Deeper and Wider [Електронний ресурс] // Nintendo E3 – 2011. – Режим доступу до ресурсу: https://web.archive.org/web/20110608122917/http://iwataasks.nintendo.com/interviews/index.html?disableNav=true/#/e32011/newhw/0/6

11. GameDesigning - Video Game Mechanics for Beginners [Електронний ресурс] // Gamedesigning – 2023. – Режим доступу до ресурсу: https://www.gamedesigning.org/learn/basic-game-mechanics/

12. Intelligent agent definition [Електронний ресурс] // TechTarget – 2019. – Режим доступу до ресурсу: https://www.techtarget.com/searchenterpriseai/definition/agent-intelligent-agent

13. OpenAI. OpenAI Five [Електронний ресурс] // OpenAI – Режим доступу до ресурсу: https://openai.com/research/openai-five

14. DeepMind. AlphaStar [Електронний ресурс] // DeepMind – Режим доступу до ресурсу: https://deepmind.com/research/highlighted-research/alphastar

15. Unity Technologies. ML-Agents Toolkit [Електронний ресурс] // GitHub – Режим доступу до ресурсу:
https://github.com/Unity-Technologies/ml-agents
16. DeepMind. MuJoCo [Електронний ресурс] // GitHub – Режим доступу до ресурсу:
https://github.com/deepmind/mujoco
17. NVIDIA. Isaac Gym [Електронний ресурс] // NVIDIA Developer – Режим доступу до ресурсу:
https://developer.nvidia.com/isaac-gym
18. CARLA. Open-source simulator for autonomous driving research [Електронний ресурс] // CARLA – Режим доступу до ресурсу:
https://carla.org
19. What Is a Game Engine? [Електронний ресурс] // How-To Geek – 2023. – Режим доступу до ресурсу:
https://www.howtogeek.com/888619/what-is-a-game-engine/
20. Welcome to unity [Електронний ресурс] // Unity – Режим доступу до ресурсу:
https://unity.com/our-company
21. Unreal Engine 5. Офіційний сайт. [Електронний ресурс] // Unreal Engine – Режим доступу до ресурсу:
https://www.unrealengine.com/en-US/unreal-engine-5
22. Unrecord. Сторінка у соц. мережах. [Електронний ресурс] // Twitter – Режим доступу до ресурсу:
https://twitter.com/unrecordgame

23. Godot Engine. Офіційна документація. [Електронний ресурс] // Godot Engine – Режим доступу до ресурсу: https://docs.godotengine.org/en/stable/about/introduction.html#about-godot-engine
24. GDScript reference. Офіційна документація. [Електронний ресурс] // Godot Engine – Режим доступу до ресурсу: https://docs.godotengine.org/en/stable/tutorials/scripting/gdscript/gdscript_basics.html
25. Total Environment Research Themes - Assessing the potential of machine learning methods to study the removal of pharmaceuticals from wastewater using biochar or activated carbon [Електронний ресурс] // ScienceDirect – 2022. – Режим доступу до ресурсу: https://www.sciencedirect.com/science/article/pii/S2772809922000016?via%3Dihub
26. ML Agents overview [Електронний ресурс] // GitHub – Режим доступу до ресурсу: https://github.com/Unity-Technologies/ml-agents/blob/develop/docs/ML-Agents-Overview.md#deep-reinforcement-learning
27. Proximal Policy Optimization [Електронний ресурс] // OpenAI – Режим доступу до ресурсу: https://web.archive.org/web/20230401063534/https://openai.com/research/openai-baselines-pp
28. Aseprite [Електронний ресурс] // Aseprite – 2024. – Режим доступу до ресурсу: https://www.aseprite.org
29. Piskel [Електронний ресурс] // PiskelApp – 2024. – Режим доступу до ресурсу: https://www.piskelapp.com

30. Pixilart [Електронний ресурс] // Pixilart – 2024. – Режим доступу до ресурсу: https://www.pixilart.com/draw
31. GraphicsGale [Електронний ресурс] // Human Balance – 2024. – Режим доступу до ресурсу: http://www.humanbalance.net/gale/us/
32. GIMP [Електронний ресурс] // GIMP – 2024. – Режим доступу до ресурсу: https://www.gimp.org
33. Adobe Photoshop [Електронний ресурс] // Adobe – 2024. – Режим доступу до ресурсу: https://www.adobe.com/products/photoshop.html

ДОДАТОК А

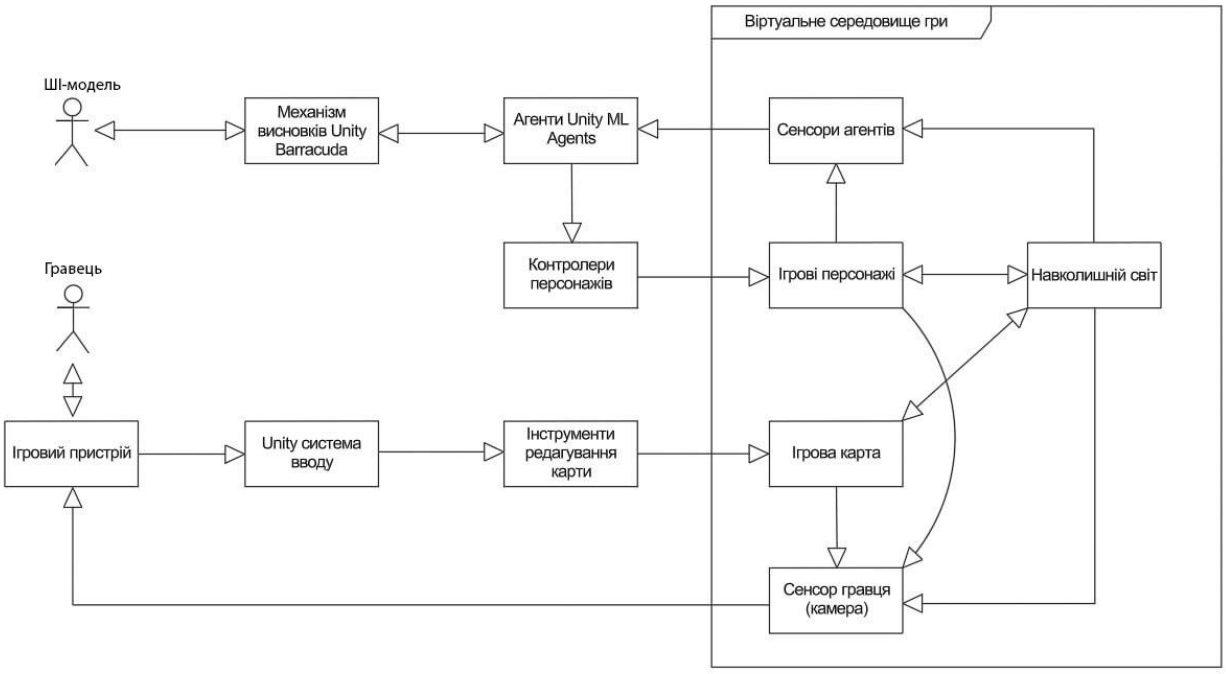
Модель 2D-агента з навчанням з підкріпленням для ігрового персонажа

Діаграма класів (функціональна схема)

ІАЛЦ.467200.004 Д1

Аркушів 1

Київ 2025 р



					ІАЛЦ.467200.004 Д1		
		№ докум.	Підпис	Дата			
Розробив		Ткаченко О.А.			Модель 2D-агента з навчанням з підкріпленням для ігрового персонажа Структурна схема системи	Літ.	Аркуш 1 / Аркушів 1
Перевірив							
						КПІ ім. Ігоря Сікорського, ФІОТ, ІО-11	
Н. Контр.		Новотарський М.А					
Затвердив							

ДОДАТОК Б

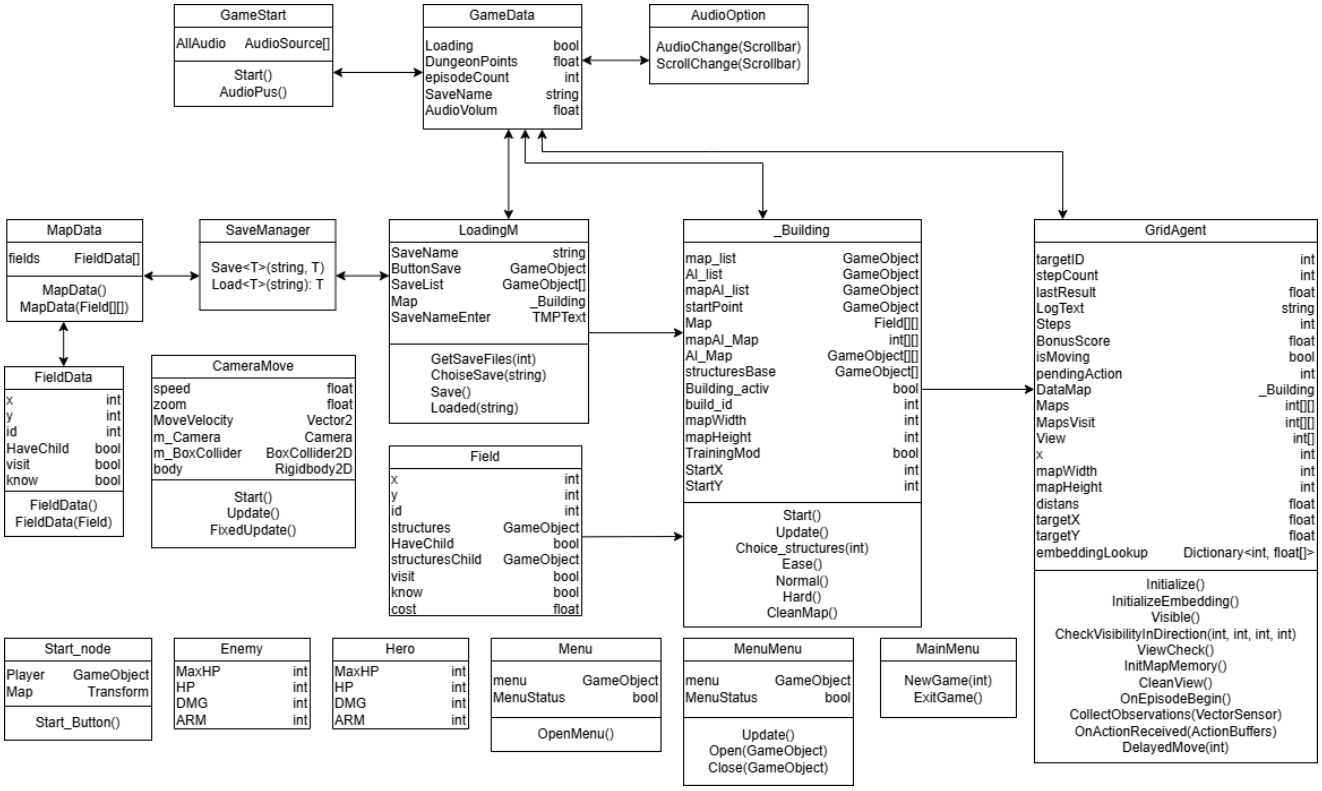
Модель 2D-агента з навчанням з підкріпленням для ігрового персонажа

Діаграма класів (функціональна схема)

ІАЛЦ.467200.005 Д2

Аркушів 1

Київ 2025 р



					ІАЛЦ.467200.005 Д2		
		№ докум.	Підпис	Дата			
Розробив		Ткаченко О.А.			Модель 2D-агента з навчанням з підкріпленням для ігрового персонажа Діаграма класів (функціональна схема)	Літ.	Аркуш 1 / Аркушів 1
Перевірив							
Н. Контр.		Новотарський М.А				КПІ ім. Ігоря Сікорського, ФІОТ, ІО-11	
Затвердив							

ДОДАТОК В

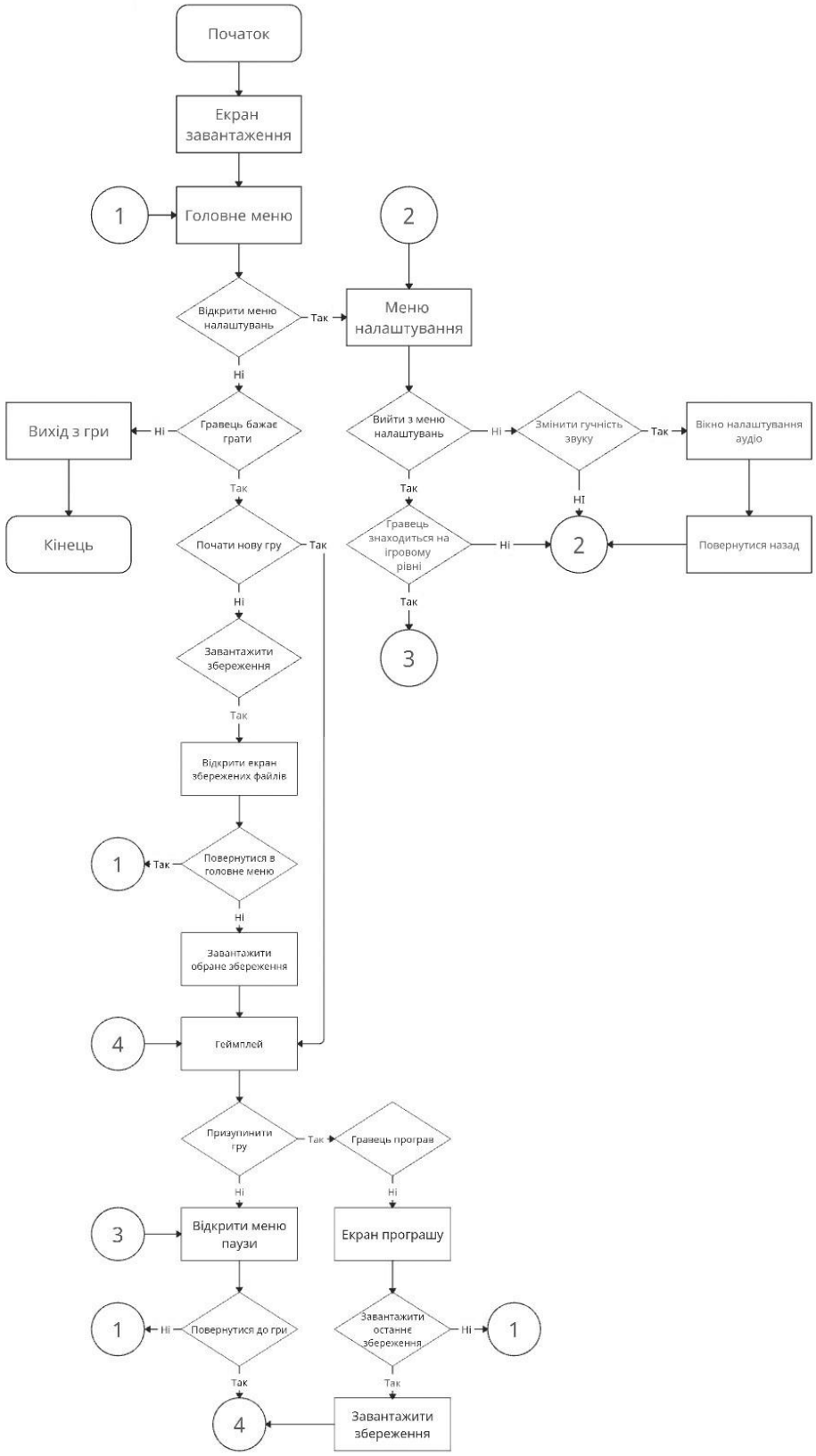
Модель 2D-агента з навчанням з підкріпленням для ігрового персонажа

Алгоритм дій програмного забезпечення (принципова схема)

ІАЛЦ.467200.006 Д3

Аркушів 1

Київ 2025 р



					ІАЛЦ.467200.006 ДЗ		
		№ докум.	Підпис	Дата			
Розробив		Ткаченко О.А.			Модель 2D-агента з навчанням з підкріпленням для ігрового персонажа Алгоритм дій програмного забезпечення (принципова схема)	Літ.	Аркуш 1 / Аркушів 1
Перевірив							
Н. Контр.		Новотарський М.А				КПІ ім. Ігоря Сікорського, ФІОТ, ІО-11	
Затвердив							

ДОДАТОК Г

Модель 2D-агента з навчанням з підкріпленням для ігрового персонажа

Текст програмного коду

ІАЛЦ.467200.007 Д4

Аркушів 21

Київ 2025 р

```
\\CameraMove.cs
using UnityEngine;
public class CameraMove : MonoBehaviour
{
    public float speed;
    private float zoom = 1;
    private Vector2 MoveVelocity;
    private Camera m_Camera;
    private BoxCollider2D m_BoxCollider;
    private Rigidbody2D body;
    void Start()
    {
        m_Camera = GetComponent<Camera>();
        m_BoxCollider = GetComponent<BoxCollider2D>();
        body = GetComponent<Rigidbody2D>();
    }
    void Update()
    {
        Vector2 InpumMove = new Vector2(Input.GetAxisRaw("Horizontal"), Input.GetAxisRaw("Vertical"));
        MoveVelocity = InpumMove * speed;
        float scroll = Input.GetAxis("Mouse ScrollWheel");
        if (zoom + scroll > 1 && zoom + scroll < 3)
        {
            zoom += scroll;
            Debug.Log("scroll");
        }
        if (zoom < 1)
        {
            zoom = 1;
        }
        if (zoom > 3)
        {
            zoom = 3;
        }
    }
    private void FixedUpdate()
    {
        body.MovePosition(body.position + MoveVelocity * Time.deltaTime);
        m_BoxCollider.size = new Vector2(19.2f / zoom, 10.8f / zoom);
        m_Camera.orthographicSize = 8.5f / zoom;
    }
}
\\Building.cs
using System.Collections;
using System.Collections.Generic;
using System.Xml.Serialization;
using Unity.VisualScripting;
using UnityEngine;
using UnityEngine.Tilemaps;
using UnityEngine.UIElements;
using static UnityEngine.Rendering.DebugUI;

public class _Building : MonoBehaviour
{
    public GameObject map_list;
    public GameObject AI_list;
```

					ІАЛЦ.467200.007 Д4		
		№ докум.	Підпис	Дата			
Розробив		Ткаченко О.А.			Модель 2D-агента з навчанням з підкріпленням для ігрового персонажа Текст програмного коду	Літ.	Аркуш 1 / Аркушів 21
Перевірив							
Н. Контр.		Новотарський М.А				КПІ ім. Ігоря Сікорського, ФІОТ, ІО-11	
Затвердив							

```
public GameObject mapAI_list;
public GameObject startPoint;
public Field[][] Map;
public int[][] mapAI_Map;
public GameObject[][] AI_Map;
public GameObject[] structuresBase;
public bool Building_activ = false;
public int build_id = -1;
[SerializeField] public int mapWidth;
[SerializeField] public int mapHeight;
[SerializeField] public bool TrainingMod = true;
public int StartX;
public int StartY;

private void Start()
{
    if (!TrainingMod)
    {
        if (!GameData.Loading)
        {
            Map = new Field[50][];
            for (int i = 0; i < 50; i++)
            {
                Map[i] = new Field[50];
            }
            for (int i = 0; i < 50; i++)
            {
                for (int j = 0; j < 50; j++)
                {
                    Map[i][j] = new Field();
                    Map[i][j].x = i;
                    Map[i][j].y = j;
                    Map[i][j].id = 1;
                    Map[i][j].structures = Instantiate(structuresBase[0], new Vector3(i, j, 0), Quaternion.identity);
                    Map[i][j].structures.name = structuresBase[0].name;
                    Map[i][j].structures.transform.SetParent(map_list.transform);
                    Map[i][j].HaveChild = false;
                    Map[i][j].visit = false;
                    Map[i][j].know = false;
                }
            }
        }
    }
    else
    {
        Map = new Field[mapWidth][];
        for (int i = 0; i < mapWidth; i++)
        {
            Map[i] = new Field[mapHeight];
            for (int j = 0; j < mapHeight; j++)
            {
                Map[i][j] = new Field();
            }
        }
    }
}
private void Update()
{
    if (Input.GetKey(KeyCode.Mouse0) && Building_activ)
```

```
    {
        Vector3 screen_position = Camera.main.ScreenToViewportPoint(Input.mousePosition);
        Vector3 world = Camera.main.ScreenToWorldPoint(Input.mousePosition);
        Vector3 WorldFix = new Vector3(Mathf.RoundToInt(world.x), Mathf.RoundToInt(world.y), 0);
        if ((WorldFix.x > -0.5f && WorldFix.x < 49.5f) && (WorldFix.y > -0.5f && WorldFix.y < 49.5f) && screen_position.y
> 0.2f)
        {
            int x = (int)WorldFix.x;
            int y = (int)WorldFix.y;
            if (Map[x][y].id != build_id + 1)
            {
                if (build_id == 0)
                {
                    Debug.Log("-");
                    Destroy(Map[x][y].structures);
                    Map[x][y].structures = Instantiate(structuresBase[0], new Vector3(x, y, 0), Quaternion.identity);
                    Map[x][y].structures.name = structuresBase[0].name;
                    Map[x][y].structures.transform.SetParent(map_list.transform);
                    Map[x][y].id = 1;
                    Map[x][y].HaveChild = false;
                }
                else if (build_id == 1)
                {
                    Debug.Log("+");
                    Destroy(Map[x][y].structures);
                    Map[x][y].structures = Instantiate(structuresBase[1], new Vector3(x, y, 0), Quaternion.identity);
                    Map[x][y].structures.name = structuresBase[1].name;
                    Map[x][y].structures.transform.SetParent(map_list.transform);
                    Map[x][y].id = 2;
                    Map[x][y].HaveChild = false;
                }
                else
                {
                    if (Map[x][y].id == 2)
                    {
                        if (Map[x][y].HaveChild)
                        {
                            if (Map[x][y].id != build_id + 1)
                            {
                                Debug.Log("has tunnel -");
                                Destroy(Map[x][y].structuresChild);
                                Map[x][y].structuresChild = Instantiate(structuresBase[build_id], new Vector3(x, y, -0.1f),
Quaternion.identity);
                                Map[x][y].structuresChild.name = structuresBase[build_id].name;
                                Map[x][y].structuresChild.transform.SetParent(Map[x][y].structures.transform);
                                Map[x][y].HaveChild = true;
                                Map[x][y].id = build_id + 1;
                            }
                        }
                        else
                        {
                            Debug.Log("has tunnel +");
                            Map[x][y].structuresChild = Instantiate(structuresBase[build_id], new Vector3(x, y, -0.1f),
Quaternion.identity);
```

```
                    Map[x][y].structuresChild.name = structuresBase[build_id].name;
                    Map[x][y].structuresChild.transform.SetParent(Map[x][y].structures.transform);
                    Map[x][y].HaveChild = true;
                    Map[x][y].id = build_id + 1;
                }
            }
        }
    }
}
if (Input.GetKey(KeyCode.Mouse1))
{
    Vector3 screen_position = Camera.main.ScreenToViewportPoint(Input.mousePosition);
    Vector3 world = Camera.main.ScreenToWorldPoint(Input.mousePosition);
    Vector3 WorldFix = new Vector3(Mathf.RoundToInt(world.x), Mathf.RoundToInt(world.y), 0);
    if ((WorldFix.x > -0.5f && WorldFix.x < 49.5f) && (WorldFix.y < 49.5f && WorldFix.y > -0.5f) && screen_position.y
> 0.2f)
    {
        int x = (int)WorldFix.x;
        int y = (int)WorldFix.y;


        Debug.Log($"Position ({Map[x][y].x}, {Map[x][y].y})\n " +
            $"ID: {Map[x][y].id}\n" +
            $"HaveChild: {Map[x][y].HaveChild}\n" +
            $"Visit: {Map[x][y].visit}\n" +
            $"Know: {Map[x][y].know}\n" +
            $"Cost: {Map[x][y].cost}\n");

    }
}
}
public void Choice_structures(int id)
{
    if (build_id == id && Building_activ)
    {
        Building_activ = false;
    }
    else
    {
        Building_activ = true;
        build_id = id;
    }
}

public void Ease()
{
    CleanMap();
    for (int i = 0; i < mapWidth; i++)
    {
        for (int j = 0; j < mapHeight; j++)
        {
            Vector3 LocalPos = new Vector3(startPoint.transform.position.x + i, startPoint.transform.position.y + j, 0);
            Map[i][j].x = i;
            Map[i][j].y = j;
            Map[i][j].id = 2;
            Map[i][j].structures = Instantiate(structuresBase[1], LocalPos, Quaternion.identity);
```

```
            Map[i][j].structures.name = structuresBase[1].name;
            Map[i][j].structures.transform.SetParent(startPoint.transform);
            Map[i][j].HaveChild = false;
            Map[i][j].visit = false;
            Map[i][j].know = false;
        }
    }

    Vector2Int start, goal;
    do
    {
        start = new Vector2Int(Random.Range(0, mapWidth), Random.Range(0, mapHeight));
        goal = new Vector2Int(Random.Range(0, mapWidth), Random.Range(0, mapHeight));
    } while (start == goal);

    Destroy(Map[start.x][start.y].structures);
    Vector3 Pos = new Vector3(startPoint.transform.position.x + start.x, startPoint.transform.position.y + start.y, 0);
    Map[start.x][start.y].id = 3;
    StartX = start.x;
    StartY = start.y;
    Map[start.x][start.y].structures = Instantiate(structuresBase[2], Pos, Quaternion.identity);
    Map[start.x][start.y].structures.name = structuresBase[2].name;
    Map[start.x][start.y].structures.transform.SetParent(startPoint.transform);
    Destroy(Map[goal.x][goal.y].structures);
    Pos = new Vector3(startPoint.transform.position.x + goal.x, startPoint.transform.position.y + goal.y, 0);
    Map[goal.x][goal.y].id = 4;
    Map[goal.x][goal.y].structures = Instantiate(structuresBase[3], Pos, Quaternion.identity);
    Map[goal.x][goal.y].structures.name = structuresBase[3].name;
    Map[goal.x][goal.y].structures.transform.SetParent(startPoint.transform);
}

public void Normal()
{
    CleanMap();
    for (int i = 0; i < mapWidth; i++)
    {
        for (int j = 0; j < mapHeight; j++)
        {
            Vector3 LocalPos = new Vector3(startPoint.transform.position.x + i, startPoint.transform.position.y + j, 0);
            Map[i][j].x = i;
            Map[i][j].y = j;
            Map[i][j].id = 2;
            Map[i][j].structures = Instantiate(structuresBase[1], LocalPos, Quaternion.identity);
            Map[i][j].structures.name = structuresBase[1].name;
            Map[i][j].structures.transform.SetParent(startPoint.transform);
            Map[i][j].HaveChild = false;
            Map[i][j].visit = false;
            Map[i][j].know = false;
        }
    }

    Vector2Int start, goal;
    do
    {
```

```
    start = new Vector2Int(Random.Range(0, mapWidth), Random.Range(0, mapHeight));
    goal = new Vector2Int(Random.Range(0, mapWidth), Random.Range(0, mapHeight));
} while (start == goal);

List<Vector2Int> path = new List<Vector2Int>();
int x = start.x, y = start.y;

while (x != goal.x)
{
    path.Add(new Vector2Int(x, y));
    x += (goal.x > x) ? 1 : -1;
}
while (y != goal.y)
{
    path.Add(new Vector2Int(x, y));
    y += (goal.y > y) ? 1 : -1;
}
path.Add(goal);

void PlaceStructure(Vector2Int pos, int id, int structureIndex)
{
    Vector3 pos3D = new Vector3(startPoint.transform.position.x + pos.x, startPoint.transform.position.y + pos.y, 0);
    Destroy(Map[pos.x][pos.y].structures);
    Map[pos.x][pos.y].id = id;
    if (id == 3)
    {
        StartX = pos.x;
        StartY = pos.y;
    }
    Map[pos.x][pos.y].structures = Instantiate(structuresBase[structureIndex], pos3D, Quaternion.identity);
    Map[pos.x][pos.y].structures.name = structuresBase[structureIndex].name;
    Map[pos.x][pos.y].structures.transform.SetParent(startPoint.transform);
}

PlaceStructure(start, 3, 2);
PlaceStructure(goal, 4, 3);

int wallCount = 5;
int placed = 0;
while (placed < wallCount)
{
    int xWall = Random.Range(0, mapWidth);
    int yWall = Random.Range(0, mapHeight);
    Vector2Int wallPos = new Vector2Int(xWall, yWall);

    if (path.Contains(wallPos) || wallPos == start || wallPos == goal || Map[xWall][yWall].id == 1)
        continue;

    Vector3 pos3D = new Vector3(startPoint.transform.position.x + xWall, startPoint.transform.position.y + yWall, 0);
    Destroy(Map[xWall][yWall].structures);
    Map[xWall][yWall].id = 1;
    Map[xWall][yWall].structures = Instantiate(structuresBase[0], pos3D, Quaternion.identity);
    Map[xWall][yWall].structures.name = structuresBase[0].name;
    Map[xWall][yWall].structures.transform.SetParent(startPoint.transform);
```

```
            placed++;
        }
    }

    public void Hard()
    {
        CleanMap();
        for (int i = 0; i < mapWidth; i++)
        {
            for (int j = 0; j < mapHeight; j++)
            {
                Vector3 LocalPos = new Vector3(startPoint.transform.position.x + i, startPoint.transform.position.y + j, 0);
                Map[i][j].x = i;
                Map[i][j].y = j;
                Map[i][j].id = 2;
                Map[i][j].structures = Instantiate(structuresBase[1], LocalPos, Quaternion.identity);
                Map[i][j].structures.name = structuresBase[1].name;
                Map[i][j].structures.transform.SetParent(startPoint.transform);
                Map[i][j].HaveChild = false;
                Map[i][j].visit = false;
                Map[i][j].know = false;
            }
        }

        Vector2Int start, goal;
        List<Vector2Int> path;

        do
        {
            start = new Vector2Int(Random.Range(0, mapWidth), Random.Range(0, mapHeight));
            goal = new Vector2Int(Random.Range(0, mapWidth), Random.Range(0, mapHeight));

            path = new List<Vector2Int>();
            int x = start.x, y = start.y;

            while (x != goal.x)
            {
                path.Add(new Vector2Int(x, y));
                x += (goal.x > x) ? 1 : -1;
            }
            while (y != goal.y)
            {
                path.Add(new Vector2Int(x, y));
                y += (goal.y > y) ? 1 : -1;
            }
            path.Add(goal);
        } while (start == goal || path.Count < 4);

        void PlaceStructure(Vector2Int pos, int id, int structureIndex)
        {
            Vector3 pos3D = new Vector3(startPoint.transform.position.x + pos.x, startPoint.transform.position.y + pos.y, 0);
            Destroy(Map[pos.x][pos.y].structures);
            Map[pos.x][pos.y].id = id;
            if (id == 3)
            {
                StartX = pos.x;
                StartY = pos.y;
```

```
        }
        Map[pos.x][pos.y].structures = Instantiate(structuresBase[structureIndex], pos3D, Quaternion.identity);
        Map[pos.x][pos.y].structures.name = structuresBase[structureIndex].name;
        Map[pos.x][pos.y].structures.transform.SetParent(startPoint.transform);
    }

    PlaceStructure(start, 3, 2);
    PlaceStructure(goal, 4, 3);

    Vector2Int branchFrom = path[Random.Range(1, path.Count - 2)];

    List<Vector2Int> directions = new List<Vector2Int> {
    new Vector2Int(1, 0), new Vector2Int(-1, 0),
    new Vector2Int(0, 1), new Vector2Int(0, -1)
  };

    foreach (var dir in directions)
    {
        Vector2Int branch = branchFrom + dir;
        if (branch.x >= 0 && branch.x < mapWidth && branch.y >= 0 && branch.y < mapHeight &&
            !path.Contains(branch) && branch != start && branch != goal)
        {
            Map[branch.x][branch.y].id = 2;
            Destroy(Map[branch.x][branch.y].structures);
            Vector3 pos3D = new Vector3(startPoint.transform.position.x + branch.x, startPoint.transform.position.y + branch.y,
0);
            Map[branch.x][branch.y].structures = Instantiate(structuresBase[1], pos3D, Quaternion.identity);
            Map[branch.x][branch.y].structures.name = structuresBase[1].name;
            Map[branch.x][branch.y].structures.transform.SetParent(startPoint.transform);
            break;
        }
    }

    for (int i = 0; i < mapWidth; i++)
    {
        for (int j = 0; j < mapHeight; j++)
        {
            Vector2Int pos = new Vector2Int(i, j);
            if (Map[i][j].id == 3 || Map[i][j].id == 4 || path.Contains(pos))
                continue;

            if (Map[i][j].structures != null)
                Destroy(Map[i][j].structures);

            Map[i][j].id = 1;
            Vector3 pos3D = new Vector3(startPoint.transform.position.x + i, startPoint.transform.position.y + j, 0);
            Map[i][j].structures = Instantiate(structuresBase[0], pos3D, Quaternion.identity); // стіна
            Map[i][j].structures.name = structuresBase[0].name;
            Map[i][j].structures.transform.SetParent(startPoint.transform);
        }
    }
  }
```

```
    public void CleanMap()
    {
        for (int i = 0; i < mapWidth; i++)
        {
            for (int j = 0; j < mapHeight; j++)
            {
                if (Map[i][j].structures != null)
                    Destroy(Map[i][j].structures);
            }
        }
    }
}
\\SaveManager.cs
using System.IO.Abstractions;
using System.IO;
using UnityEngine;
using UnityEditor.U2D.Aseprite;

public static class SaveManager
{
    public static void Save<T>(string fileName, T saveData)
    {
        string saveFolder = Application.persistentDataPath + "/Saves/" + fileName;
        if (!Directory.Exists(saveFolder))
            Directory.CreateDirectory(saveFolder);

        string fullPath = Path.Combine(saveFolder, fileName + ".json");

        string jsonDataString = JsonUtility.ToJson(saveData, true);
        File.WriteAllText(fullPath, jsonDataString);

        Debug.Log("Saved to " + fullPath);
    }

    public static T Load<T>(string fileName) where T : new()
    {
        string saveFolder = Application.persistentDataPath + "/Saves/" + fileName;
        string fullPath = Path.Combine(saveFolder, fileName + ".json");

        if (!File.Exists(fullPath))
        {
            Debug.LogError("Save file not found: " + fullPath);
            return new T();
        }
        string jsonDataString = File.ReadAllText(fullPath);
        T Data = JsonUtility.FromJson<T>(jsonDataString);
        return Data;
    }
}
\\MenuMenu.cs
using UnityEngine;

public class MenuMenu : MonoBehaviour
{
    public GameObject menu;
    private bool MenuStatus = false;

    void Update()
    {
        if (Input.GetKey(KeyCode.Escape))
        {
```

```
            MenuStatus = !MenuStatus;
            menu.SetActive(MenuStatus);
        }
    }

    public void Open(GameObject menu)
    {
        menu.SetActive(true);
    }

    public void Close(GameObject menu)
    {
        menu.SetActive(false);
    }
}
\\Menu.cs
using UnityEngine;

public class Menu : MonoBehaviour
{
    public GameObject menu;
    private bool MenuStatus = false;

    public void OpenMenu()
    {
        MenuStatus = !MenuStatus;
        menu.SetActive(MenuStatus);
    }
}
\\Start_node.cs
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;

public class Start_node : MonoBehaviour
{
    public GameObject Player;
    public Transform Map;

    public void Start_Button()
    {
        GameObject obj = GameObject.Find("Map");
        Map = obj.GetComponent<Transform>();
        bool have_start = false;
        bool have_end = false;
        Transform start_node = null;
        for (int i = 0; i < Map.childCount; i++)
        {
            Transform node = Map.GetChild(i);
            if (node.childCount != 0)
            {
                if (node.GetChild(0).name == "Start")
                {
                    have_start = true;
                    start_node = node.GetChild(0).transform;
                }
                else if (node.GetChild(0).name == "Finish")
                {
                    have_end = true;
                }
            }
```

```
        }
        if (have_start && have_end)
        {
            GameObject AG = Instantiate(Player, new Vector3(start_node.position.x, start_node.position.y, -0.2f),
Quaternion.identity);
            AG.transform.SetParent(Map);
        }
    }
}
\\GridAgent.cs
using Unity.MLAgents;
using Unity.MLAgents.Sensors;
using Unity.MLAgents.Actuators;
using UnityEngine;
using Unity.VisualScripting;
using System.Collections;
using System.Collections.Generic;
using static UnityEditor.PlayerSettings;
using TMPro;

public class GridAgent : Agent
{
    [SerializeField] private int targetID;
    [SerializeField] private int stepCount = 50;
    private float lastResult;
    private string LogText;
    private int Steps = 0;
    private float BonusScore = 0;
    private bool isMoving = false;
    private int pendingAction = -1;
    public _Building DataMap;
    public int[][] Maps;
    public int[][] MapsVisit;
    public int[] View;
    public int x = 0;
    private int mapWidth;
    private int mapHeight;
    private float distans = float.MaxValue;
    private float targetX;
    private float targetY;

    private Dictionary<int, float[]> embeddingLookup;

    private void InitializeEmbedding()
    {
        embeddingLookup = new Dictionary<int, float[]>();

        embeddingLookup[0] = new float[] { 0f, 0f };      // Невідоме
        embeddingLookup[1] = new float[] { 1f, 0f };      // Стіна
        embeddingLookup[2] = new float[] { 0f, 1f };      // Коридор
        embeddingLookup[3] = new float[] { 0.5f, 0.5f };  // Старт
        embeddingLookup[4] = new float[] { 0.25f, 0.75f }; // Фініш
    }

    public override void Initialize()
    {
        InitializeEmbedding();
    }
```

```
  private void Visible()
  {
     CleanView();
     if (Maps[(int)transform.localPosition.x][(int)transform.localPosition.y] == 0)
     {
        BonusScore += 0.1f;
        AddReward(+0.1f);
     }
     Maps[(int)transform.localPosition.x][(int)transform.localPosition.y] =
DataMap.Map[(int)transform.localPosition.x][(int)transform.localPosition.y].id;
     CheckVisibilityInDirection((int)transform.localPosition.x, (int)transform.localPosition.y, 0, 1);  // Вгору
     CheckVisibilityInDirection((int)transform.localPosition.x, (int)transform.localPosition.y, 0, -1); // Вниз
     CheckVisibilityInDirection((int)transform.localPosition.x, (int)transform.localPosition.y, 1, 0);  // Вправо
     CheckVisibilityInDirection((int)transform.localPosition.x, (int)transform.localPosition.y, -1, 0); // Вліво
     ViewCheck();
     x = 0;
  }

  private void CheckVisibilityInDirection(int x_now, int y_now, int x_dir, int y_dir)
  {
     for (int i = 1; i < 4; i++)
     {
        int x_check = x_now + x_dir * i;
        int y_check = y_now + y_dir * i;

        if (x_check >= 0 && x_check < mapWidth && y_check >= 0 && y_check < mapHeight)
        {
           if (DataMap.Map[x_check][y_check].id != 1)
           {
              for (int j = -1; j < 2; j++)
              {
                 int x_adj = x_check + (x_dir == 0 ? j : 0);
                 int y_adj = y_check + (y_dir == 0 ? j : 0);

                 if (x_adj >= 0 && x_adj < mapWidth && y_adj >= 0 && y_adj < mapHeight)
                 {
                    if (Maps[x_adj][y_adj] == 0)
                    {
                       BonusScore += 0.05f;
                       AddReward(+0.05f);
                    }
                    Maps[x_adj][y_adj] = DataMap.Map[x_adj][y_adj].id;
                 }
              }
           }
           else
           {
              if (Maps[x_check][y_check] == 0)
              {
                 BonusScore += 0.05f;
                 AddReward(+0.05f);
              }
              Maps[x_check][y_check] = DataMap.Map[x_check][y_check].id;
              break;
           }
```

```
        }
        else
        {
            break;
        }
    }
}

private void ViewCheck()
{
    int Xpos = (int)transform.localPosition.x;
    int Ypos = (int)transform.localPosition.y;
    for (int i = -3; i < 4; i++)
    {
        int Ypos_check = Ypos + i;
        for (int j = -1; j < 2; j++)
        {
            int Xpos_check = Xpos + j;
            if (Xpos_check >= 0 && Xpos_check < mapWidth && Ypos_check >= 0 && Ypos_check < mapHeight)
            {
                View[x] = Maps[Xpos_check][Ypos_check];
                View[x + 1] = Xpos_check;
                View[x + 2] = Ypos_check;
            }
            else
            {
                View[x] = 0;
                View[x + 1] = mapWidth;
                View[x + 2] = mapHeight;
            }
            x += 3;
        }
    }

    for (int i = -3; i < 4; i++)
    {
        int Xpos_check = Xpos + i;
        if (i < -1 || i > 1)
        {
            for (int j = -1; j < 2; j++)
            {
                int Ypos_check = Ypos + j;
                if (Xpos_check >= 0 && Xpos_check < mapWidth && Ypos_check >= 0 && Ypos_check < mapHeight)
                {
                    View[x] = Maps[Xpos_check][Ypos_check];
                    View[x + 1] = Xpos_check;
                    View[x + 2] = Ypos_check;
                }
                else
                {
                    View[x] = 0;
                    View[x + 1] = mapWidth;
                    View[x + 2] = mapHeight;
                }
                x += 3;
            }
        }
    }
}
```

```
private void InitMapMemory()
{
    View = new int[99];
    Maps = new int[mapWidth][];
    MapsVisit = new int[mapWidth][];
    for (int i = 0; i < mapWidth; i++)
    {
        Maps[i] = new int[mapHeight];
        MapsVisit[i] = new int[mapHeight];

        for (int j = 0; j < mapHeight; j++)
        {
            MapsVisit[i][j] = 0;
            Maps[i][j] = DataMap.Map[i][j].know ? DataMap.Map[i][j].id : 0;
        }
    }
}
private void CleanView()
{
    for (int i = 0; i < 99; i += 3)
    {
        View[i] = 0;
        View[i + 1] = mapWidth;
        View[i + 2] = mapHeight;
    }
}

public override void OnEpisodeBegin()
{
    GameObject mapObj = transform.parent.gameObject;
    DataMap = mapObj.GetComponent<_Building>();
    if (GameData.episodeCount <= 600)
    {
        DataMap.Ease();
    }
    else if (GameData.episodeCount > 600 && GameData.episodeCount <= 1200)
    {
        DataMap.Normal();
    }
    else
    {
        DataMap.Hard();
    }
    if (GameData.episodeCount != 0)
    {
        LogText += $"Episode: {GameData.episodeCount} | Reward: {lastResult} | Steps {Steps}\n";
    }
    GameData.episodeCount++;
    BonusScore = 0f;
    stepCount = 50;
    Steps = 0;
    Debug.Log(LogText);
    LogText = "";

    isMoving = false;
    pendingAction = -1;

    transform.localPosition = new Vector3(DataMap.StartX, DataMap.StartY, -1f);
```

```
        mapWidth = DataMap.Map.Length;
        mapHeight = DataMap.Map[0].Length;

        InitMapMemory();

        Visible();
    }

    public override void CollectObservations(VectorSensor sensor)
    {
        Visible();
        float agentPosX = transform.localPosition.x;
        float agentPosY = transform.localPosition.y;
        for (int i = 0; i < 99; i += 3)
        {
            int id = View[i];
            float[] embedding = embeddingLookup.ContainsKey(id) ? embeddingLookup[id] : embeddingLookup[0];
            sensor.AddObservation(embedding[0]);
            sensor.AddObservation(embedding[1]);
            sensor.AddObservation((float)View[i + 1] / mapWidth);
            sensor.AddObservation((float)View[i + 2] / mapWidth);
        }
        sensor.AddObservation(agentPosX / mapWidth);
        sensor.AddObservation(agentPosY / mapHeight);
        float[] targetEmbedding = embeddingLookup.ContainsKey(targetID) ? embeddingLookup[targetID] : embeddingLookup[0];
        sensor.AddObservation(targetEmbedding[0]);
        sensor.AddObservation(targetEmbedding[1]);
        distans = float.MaxValue;
        float targetPosX = 1;
        float targetPosY = 1;
        targetX = mapWidth;
        targetY = mapHeight;
        for (int i = 0; i < mapWidth ; i++)
        {
            for (int j = 0; j < mapHeight ; j++)
            {

                if (Maps[i][j] == targetID && distans > Mathf.Sqrt((agentPosX - i) * (agentPosX - i) + (agentPosY - j) * (agentPosY -
j)))
                {
                    targetPosX = (float)i / mapWidth;
                    targetPosY = (float)j / mapHeight;
                    distans = Mathf.Sqrt((agentPosX - i) * (agentPosX - i) + (agentPosY - j) * (agentPosY - j));
                    targetX = i;
                    targetY = j;
                }
            }
        }
        if (distans == float.MaxValue)distans = 0;
        sensor.AddObservation(targetPosX);
        sensor.AddObservation(targetPosY);
    }

    public override void OnActionReceived(ActionBuffers actions)
    {
        if (!isMoving)
        {
```

```
            pendingAction = actions.DiscreteActions[0];
            StartCoroutine(DelayedMove(pendingAction));
        }
    }

    private IEnumerator DelayedMove(int action)
    {
        isMoving = true;
        Steps++;
        yield return new WaitForSeconds(1f);

        Vector2Int currentPos = new Vector2Int((int)transform.localPosition.x, (int)transform.localPosition.y);
        Vector2Int moveDir = Vector2Int.zero;

        switch (action)
        {
            case 0: moveDir = Vector2Int.left; break;
            case 1: moveDir = Vector2Int.right; break;
            case 2: moveDir = Vector2Int.down; break;
            case 3: moveDir = Vector2Int.up; break;
        }

        Vector2Int targetPos = currentPos + moveDir;
        if (targetPos.x >= 0 && targetPos.x < mapWidth && targetPos.y >= 0 && targetPos.y < mapHeight)
        {
            int nextID = Maps[targetPos.x][targetPos.y];
            LogText += $"Action {action} | {targetPos.x} : {targetPos.y} | ID {nextID}\n";

            if (nextID != 1)
            {

                transform.localPosition = new Vector3(targetPos.x, targetPos.y, -1);
                Visible();
                stepCount--;
                AddReward(-0.1f);
                if (targetX != mapWidth)
                {
                    float deltaDist = distans - Mathf.Sqrt((targetPos.x - targetX) * (targetPos.x - targetX) + (targetPos.y - targetY) *
(targetPos.y - targetY));
                    AddReward(deltaDist * 0.3f);
                }

                if (nextID == targetID)
                {
                    Debug.Log("+");
                    LogText += "Finish\n";
                    AddReward(+5f);
                    AddReward(+stepCount);
                    lastResult = GetCumulativeReward();
                    for (int i = 0; i < mapWidth; i++)
                    {
                        for (int j = 0; j < mapHeight; j++)
                        {
                            if (Maps[i][j] != 0)
                            {
                                DataMap.Map[i][j].know = true;
                            }
                        }
```

```
                }
                EndEpisode();
            }
        }
        else
        {
            AddReward(-1.5f);
        }
    }
    else
    {
        AddReward(-1.5f);
    }

    if (stepCount == 0)
    {
        Debug.Log("-");
        SetReward(-5f);
        AddReward(-BonusScore * 0.5f);
        lastResult = GetCumulativeReward();
        EndEpisode();
    }
    lastResult = GetCumulativeReward();
    isMoving = false;
    RequestDecision();
  }
}
```

\\Field.cs

```
using UnityEngine;

public class Field
{
    public int x;
    public int y;

    public int id;
    public GameObject structures;
    public bool HaveChild;
    public GameObject structuresChild;

    public bool visit;
    public bool know;
    public float cost;

}
```

\\LoadingM.cs

```
using SaveData;
using System.IO;
using System.IO.Abstractions;
using System.Text.RegularExpressions;
using TMPro;
using Unity.VisualScripting;
using UnityEngine;

public class LoadingM : MonoBehaviour
{
    [SerializeField] public string SaveName;
    public GameObject ButtonSave;
    public GameObject[] SaveList;
```

```
  public _Building Map;
  public TextMeshProUGUI SaveNameEnter;
  public void GetSaveFiles(int id)
  {
     string saveFolder = Application.persistentDataPath + "/Saves";

     if (Directory.Exists(saveFolder))
     {
        string[] files = Directory.GetDirectories(saveFolder);

        for (int i = 0; i < files.Length; i++)
        {
           files[i] = Path.GetFileNameWithoutExtension(files[i]);
        }
        Debug.Log(saveFolder);
        Debug.Log(files.Length);
        SaveList[id].GetComponent<RectTransform>().SetSizeWithCurrentAnchors(RectTransform.Axis.Vertical, 110 *
files.Length);

        for (int j = 0; j < files.Length; j++)
        {
           GameObject button = Instantiate(ButtonSave);
           button.name = files[j];
           button.transform.SetParent(SaveList[id].transform);
           TextMeshProUGUI text = button.GetComponentInChildren<TextMeshProUGUI>();
           text.text = files[j];
           int index = j;
           button.GetComponent<UnityEngine.UI.Button>().onClick.AddListener(() => Loaded(files[index]));
        }
     }
  }

  public void Save()
  {
     string saveFolder = Application.persistentDataPath + "/Saves";

     if (!Directory.Exists(saveFolder))
        Directory.CreateDirectory(saveFolder);

     SaveName = SaveNameEnter.text;

     SaveName = Regex.Replace(SaveName, @"[^\w\- ]", "");

     if (string.IsNullOrWhiteSpace(SaveName))
        SaveName = "Збереження";

     int i = 1;
     string originalName = SaveName;
     while (File.Exists(Path.Combine(saveFolder, SaveName + ".json")))
     {
        SaveName = originalName + " " + i;
        i++;
     }

     Field[][] map = Map.Map;
     MapData mapData = new MapData(map);
     SaveManager.Save(SaveName, mapData);
```

```
    }

    public void Loaded(string saveName)
    {
        MapData mapData = SaveManager.Load<MapData>(saveName);
        if (mapData == null || mapData.fields == null || mapData.fields.Length != 2500)
        {
            Debug.LogError("MapData is null or corrupted.");
            return;
        }

        if (Map.Map == null)
        {
            Map.Map = new Field[50][];
            for (int i = 0; i < 50; i++)
            {
                Map.Map[i] = new Field[50];
            }
        }

        for (int i = 0; i < 50; i++)
        {
            for (int j = 0; j < 50; j++)
            {
                if (Map.Map[i][j] != null && Map.Map[i][j].structures != null)
                    Destroy(Map.Map[i][j].structures);
            }
        }

        int index = 0;
        for (int i = 0; i < 50; i++)
        {
            for (int j = 0; j < 50; j++)
            {
                FieldData fd = mapData.fields[index++];
                Map.Map[i][j] = new Field();
                Map.Map[i][j].x = fd.x;
                Map.Map[i][j].y = fd.y;
                Map.Map[i][j].id = fd.id;
                if (fd.id > 2)
                {
                    Map.Map[i][j].structures = Instantiate(Map.structuresBase[1], new Vector3(fd.x, fd.y, 0), Quaternion.identity);
                    Map.Map[i][j].structures.name = Map.structuresBase[1].name;
                    Map.Map[i][j].structures.transform.SetParent(Map.map_list.transform);
                    Map.Map[i][j].structuresChild = Instantiate(Map.structuresBase[fd.id - 1], new Vector3(fd.x, fd.y, -0.1f),
Quaternion.identity);
                    Map.Map[i][j].structuresChild.name = Map.structuresBase[fd.id - 1].name;
                    Map.Map[i][j].structuresChild.transform.SetParent(Map.Map[i][j].structures.transform);
                }
                else
                {
                    Map.Map[i][j].structures = Instantiate(Map.structuresBase[fd.id - 1], new Vector3(fd.x, fd.y, 0), Quaternion.identity);
                    Map.Map[i][j].structures.name = Map.structuresBase[fd.id - 1].name;
                    Map.Map[i][j].structures.transform.SetParent(Map.map_list.transform);
                }
                Map.Map[i][j].HaveChild = fd.HaveChild;
                Map.Map[i][j].visit = fd.visit;
```

```
            Map.Map[i][j].know = fd.know;
        }
    }
    GameData.Loading = true;
    Debug.Log("Map Loaded");
  }

}
\\FieldData.cs
using Google.Protobuf;
using System;
using UnityEngine;

namespace SaveData
{
  [System.Serializable]
  public class FieldData
  {
    public int x;
    public int y;
    public int id;
    public bool HaveChild;
    public bool visit;
    public bool know;

    public FieldData() { }

    public FieldData(Field field)
    {
      x = field.x;
      y = field.y;
      id = field.id;
      HaveChild = field.HaveChild;
      visit = field.visit;
      know = field.know;
    }
  }

  [System.Serializable]
  public class MapData
  {
    public FieldData[] fields;

    public MapData() { }

    public MapData(int width, int height)
    {
      fields = new FieldData[width * height];
    }

    public MapData(Field[][] map)
    {
      int width = map.Length;
      int height = map[0].Length;
      fields = new FieldData[width * height];

      int index = 0;
      for (int i = 0; i < width; i++)
      {
```

```
            for (int j = 0; j < height; j++)
            {
                fields[index++] = new FieldData(map[i][j]);
            }
        }
    }
  }
}

\\GameData.cs
using UnityEngine;

public static class GameData
{
    public static bool Loading = false;
    public static float DungeonPoints = 2000;
    public static int episodeCount = 0;
}
```