НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
Факультет інформатики та обчислювальної техніки
Кафедра обчислювальної техніки

«На правах рукопису»
УДК __004.042_____

До захисту допущено:
В.О. Завідувача кафедри
_____ Михайло Новотарський
«___»_____2024 р.

Магістерська дисертація

на здобуття ступеня магістра
за освітньо-професійною програмою «Комп'ютерні системи та мережі» зі спеціальності 123 «Комп'ютерна інженерія»
на тему: «Підвищення ефективності систем моделювання реальних подій за допомогою нейронних мереж»

Виконав:
студент ІІ курсу, групи ІО-31мп
Максим ЄЛІСЄЄВ _____

Керівник:
зав. каф. ОТ, д.т.н., професор,
Сергій СТІРЕНКО _____

Консультант з нормоконтролю:
проф. каф. ОТ, д.т.н., професор
Юрій КУЛАКОВ_____

Рецензент:
_____

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2024 року

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ «КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

Факультет інформатики та обчислювальної техніки
(повне найменування інституту, факультету)

Кафедра обчислювальної техніки
(повна назва кафедри)

Рівень вищої – освіти – другий (магістерський)

Спеціальність – 123 «Комп'ютерна інженерія»

Освітньо-професійна програма – «Комп'ютерні системи та мережі»

До захисту допущено:

Завідувач кафедри

_____ Михайло НОВОТАРСЬКИЙ
(підпис) (ім'я, прізвище)

«___»_____ 2024 р.

ЗАВДАННЯ

на магістерську дисертацію студенту

Єлісєєву Максиму Олеговичу

1. Тема дисертації "*Підвищення ефективності систем моделювання реальних подій за допомогою нейронних мереж*", керівник дисертації *Стіренко Сергій Григорович, професор, доктор технічних наук* , затверджені наказом по університету від « 08 » листопада 2024р.№ 5016-с

2. Строк подання студентом проекту *17.12.2024р.*

3. Об'єкт дослідження *Використання нейронних мереж в системах моделювання реальних подій*

4. Вихідні дані: *технічне завдання, теоретичні дані, програмна реалізація.*

5. Перелік завдань, які потрібно розробити: *огляд методів моделювання ти систем моделювання реальних подій, проектування власної системи моделювання реальних подій, проектування та розробка програмного забезпечення, огляд існуючих рушень для моделювання реальних подій, проектування власної системи моделювання реальних подій, побудова*

інтерфейсу передачі інформації між системою моделювання та нейронною мережею, тестування та аналіз розробленої моделі

6. Перелік графічного (ілюстративного) матеріалу : *дослідження результатів моделювання розробленого сценарію.*

7. Консультанти розділів дисертації:

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв
1.			
2.			
3.			
4.			

8. Дата видачі завдання 24.02.2024 року

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Найменування етапів дипломного проекту (роботи)	Строк виконання етапів проекту(роботи)	Примітки
1	Обговорення теми дисертації. Визначення предмету дослідження	17.02.24-24.02.24	
2	Дослідження існуючих рішень та систем моделювання реальних подій	02.03.24-31.05.24	
3	Побудова архітектурного рішення	02.06.24-05.07.24	
4	Програмна реалізація створеної системи моделювання реальних подій	01.08.24-31.09.24	
5	Розборка функціоналу та сценарію агент-орієнтованого моделювання подій	10.10.24-31.10.24	
6	Розробка процесу обробки, передачі інформації між системою	01.11.24-20.11.24	
7	Розробка стартам-проекту	21.11.24-23.11.24	
8	Оформлення магістерської дисертації.	24.11.24-29.11.24	

Студент Максим ЄЛІСЄЄВ _____

Керівник Сергій СТІРЕНКО _____

(підпис)

РЕФЕРАТ

Магістерська дисертація на тему «Підвищення ефективності систем моделювання реальних подій за допомогою нейронних мереж» містить 74 сторінки основного тексту, в неї входять 2 ілюстрації 22 таблиці та 11 прикладів вихідного коду. Загальний обсяг дисертації складає приблизно 103 сторінок

Об'єкт дослідження: інтеграція системи моделювання системи моделювання реальних подій з нейронною мережею.

Предмет дослідження: система моделювання реальних подій на основі агентного моделювання з механізмом прийняття рішень на основі нейронної мережі.

Мета дослідження: підвищення ефективності виявлення та демонстрації емерджентної поведінки під час моделювання.

Задачі дослідження:

1. Розробити систему моделювання для демонстрації емерджентної поведінки під час моделювання сценарію
2. Інтегрувати нейронну мережу до системи моделювання реальних подій для підвищення адаптивності модельованого процесу.
3. Аналіз результатів системи моделювання.

Актуальність: Системи моделювання реальних подій є надзвичайно важливим інструментом у багатьох сучасних галузях, включаючи авіацію, медицину, урбаністику, логістику та інші. Ці системи дозволяють створювати віртуальні середовища, в яких можна тестувати сценарії, аналізувати можливі наслідки прийняття рішень, оптимізувати процеси та ресурси.

Поєднання таких систем із нейронними мережами відкриває можливості для автоматизації та інтелектуалізації процесів. Завдяки здатності нейронних мереж до навчання, аналізу великих обсягів даних і виявлення прихованих закономірностей, вони можуть покращити точність прогнозів, адаптуватися до змін середовища в реальному часі та допомагати у прийнятті оптимальних рішень. В авіації такі системи можуть використовуватися для прогнозування несправностей, оптимізації маршрутів або аналізу ризиків. В рятувальних операціях така система може пропонувати варіанти рішень для координаторів операції. Що може підвищити швидкість прийняття найбільш ефективних рішень завдяки постійному аналізу актуальної інформації. Що в свою черго може підвищити вірогідність успішності операції.

Ключові слова: системи моделювання реальних подій, імітаційне моделювання, моделювання на основі агентів, нейронні мережі.

ABSTRACT

The master's thesis on the topic "Enhancing the efficiency of real-world event simulation systems using neural networks" the main text consists of 74 pages, including 2 illustrations, 22 tables and 11 examples of source code. The total length of the dissertation is approximately 103 pages.

Object of Research: Integration of a real-world event simulation system with a neural network

Subject of Research: A real-world event simulation system based on agent-based modeling with a decision-making mechanism powered by a neural network.

Research Objective: To enhance the efficiency of detecting and demonstrating emergent behavior during simulations.

Research Tasks:

1. Develop a simulation system to demonstrate emergent behavior in scenario modeling.
2. Integrate a neural network into the real-world event simulation system to increase the adaptability of the modeled processes.
3. Analyze the results produced by the simulation system.

Relevance: Simulation systems of real-world events are an extremely important tool in many modern industries, including aviation, medicine, urban planning, logistics, and others. These systems enable the creation of virtual environments where various scenarios can be tested, the potential consequences of decisions analyzed, and processes and resources optimized.

Integration between simulation systems and neural networks may open up possibilities for automation and process intelligence. Because of neural networks' ability to learn, analyze large volumes of data, and detect hidden patterns, they can improve the accuracy of forecasts, adapt to environmental changes in real time,

and assist in making optimal decisions. In aviation, such systems can be used for fault prediction, route optimization, or risk analysis. In rescue operations, such a system can suggest decision-making options for operation coordinators, enabling them to quickly evaluate scenarios and choose the most effective course of action. These systems can analyze real-time data, predict outcomes, and provide recommendations based on the specifics of the situation, significantly improving the efficiency and success rates of rescue missions.
Keywords: real-world event simulation systems, simulation modeling, agent-based modeling, neural networks

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

ABM (Agent-Based Model) – агент-орієнтована модель, модель, що базується на використанні агентів.

DES (Discrete Event Simulation) – імітація дискретних подій.

FEA (Finite Element Analysis) – метод кінцевих елементів, що використовується для чисельного аналізу фізичних процесів.

CA (Cellular Automata) – клітинний автомат, дискретна модель, що базується на використанні сітки клітин із правилами їх оновлення.

LLM (Large Language Model) – велика мовна модель, модель на базі машинного навчання, натренована на обробці природної мови.

NN (Neural Network) – нейронна мережа, обчислювальна система, натхненна принципами роботи біологічних нервових систем.

MANA (Map Aware Non-uniform Automata) – система моделювання, що поєднує агент-орієнтований підхід з аналізом топографічних даних.

GIS (Geographic Information System) – геоінформаційна система, що використовується для збору, аналізу та візуалізації просторових даних.

РОЗДІЛ 1
СФЕРИ ЗАСТОСУВАННЯ СИСТЕМ МОДЕЛЮВАННЯ РЕАЛЬНИХ ПОДІЙ

Сучасний світ, вимагає все більш точного і всебічного розуміння складних процесів і явищ у різних галузях: транспорті, енергетиці, природних науках, охороні здоров'я, фінансах, науці, інженерії та багатьох інших. З огляду на це, моделювання реальних подій стає незамінним інструментом для аналізу, прогнозування і підтримки прийняття рішень. Завдяки використанню математичних, статистичних та комп'ютерних методів, моделювання дозволяє створювати точні копії реальних процесів і експериментувати з ними в цифровому середовищі

Моделювання реальних подій є науково-технічним підходом, що полягає у створенні цифрових моделей об’єктів, процесів або явищ для дослідження, аналізу та прогнозування їх поведінки за певних умов. У найбільш загальному сенсі модель — це спрощене представлення реальності, що описує ключові характеристики і закономірності певного процесу. Моделювання дозволяє замінити експериментування з реальним об'єктом або процесом на аналіз його цифрової копії, що є особливо актуальним у випадках, коли реальний експеримент є дорогим, небезпечним або навіть неможливим.

Основними поняттями в галузі моделювання є:

- Модель: абстрактна або матеріальна копія об’єкта чи процесу, що відображає його структуру і ключові властивості.
- Моделювання: процес створення та використання моделей для вивчення характеристик реальних об’єктів або явищ.

- Імітація: процес відтворення реальних процесів на основі моделі, за допомогою комп'ютерних обчислень або експериментів.
- Симуляція: форма моделювання, що включає проведення серії обчислень або експериментів над моделлю для отримання прогнозів або оцінок поведінки системи.

Основною метою моделювання є покращення розуміння реальних об’єктів і процесів, підвищення ефективності управління та прийняття рішень, оптимізація ресурсів і підвищення безпеки операцій.

1.1 Класифікація методів моделювання

Методи моделювання класифікуються за різними критеріями залежно від цілей, типів об’єктів, що вивчаються, і технічних можливостей

1.1.1 Математичне моделювання

Під час математичного моделювання використовуються методи математичних рівнянь для опису закономірностей і поведінки об’єктів. Дозволяє проводити кількісні аналізи та прогнози.
Прикладами їх застосування можуть бути такі галузі як фізика, фінанси, інженерія. Найбільш розповсюдженими програмами для цих цілей є

- **MATLAB**: Для розв’язання систем диференційних рівнянь, аналізу і обробки даних.
- **Maple**: Використовується для розв'язання алгебраїчних рівнянь і символьних обчислень.

- **Mathematica**: Застосовується для математичного моделювання і складних чисельних обчислень.

1.1.2 Комп'ютерне моделювання

Створення комп'ютерних моделей для імітації та прогнозування поведінки складних систем, зокрема природних і соціальних. Частіше за все вони використовуються у сферах дослідження клімату, транспорту чи медицини також можуть використовуватися у військових цілях.
Прикладами таких програм можна навести:

- **ANSYS**: Використовується в обчислювальній динаміці рідин та інженерних симуляціях.
- **Simulink**: Додаток MATLAB для моделювання динамічних систем, зокрема в автомобільній і аерокосмічній промисловості.
- **COMSOL Multiphysics**: Дозволяє проводити багатофізичні моделювання, зокрема для симуляцій теплопередачі та електромагнітних полів.
- **MANA (Map-Aware Non-uniform Automata) Agent-Based Simulation Tool**: це агентно-орієнтований інструмент моделювання, розроблений для симуляції складних систем із взаємодією агентів у динамічному середовищі. MANA особливо широко використовується у військових і стратегічних дослідженнях для моделювання поведінки груп і військових одиниць у сценаріях бойових дій.

MANA дозволяє створювати агентів, які діють за заданими алгоритмами, реагують на зміну навколишнього середовища і взаємодіють один з одним відповідно до заданих правил. Кожен агент має певні характеристики та

стратегії, що визначають його дії, а також здатність адаптувати поведінку на основі змін у середовищі чи поведінки інших агентів.

- Особливості:
  - MANA має можливість моделювати великі групи агентів у геопросторово прив’язаних картах.
  - Агентів можна налаштовувати, надаючи їм унікальні властивості, такі як мотивація, здатність до спілкування з іншими агентами, агресивність або обережність.
  - Підтримує розробку багатосценарійних симуляцій для аналізу тактики, ресурсів, розташування і реакції на зовнішні зміни.
  - Інструмент має простий інтерфейс і дозволяє візуалізувати симуляції, що робить його корисним для стратегічного аналізу та оцінки ризиків.
- Приклади застосування:
  - Військові симуляції: Вивчення тактики, ефективності бойових одиниць і групової динаміки.
  - Оцінка ризиків: Аналіз наслідків для населення та планування евакуацій під час надзвичайних ситуацій.
  - Розробка стратегій: Використовується в навчальних симуляціях для підготовки військових офіцерів та стратегів.

Завдяки можливості моделювання великих та складних систем, MANA дозволяє досліджувати результати стратегічних рішень, тестувати й оптимізувати сценарії, що робить його ефективним для досліджень у галузях оборони, безпеки та управління кризовими ситуаціями.

1.1.3 Статистичне моделювання

Використання статистичних методів для виявлення зв’язків між змінними та побудови прогнозів. Частіше за все використовуються в економіці, соціології та медицині. Прикладом можуть бути наступні програми

- **R**: Пакет для статистичних розрахунків, широко використовується для аналізу даних і візуалізації.
- **SPSS**: Програма для аналізу статистичних даних, яка застосовується у маркетингу та соціальних науках.
- **SAS**: Використовується для передбачення та управління ризиками, особливо у сфері фінансів та охорони здоров’я.

1.1.4 Системне і структурне моделювання

Моделювання структурних компонентів та їхніх зв’язків у рамках цілісної системи. Дані рішення частіше за все використовуються у біології, менеджменті та інформаційних технологіях. Прикладами програм можна навести наступні рішення.

- **Enterprise Architect**: Для побудови UML-діаграм, використовується у програмній інженерії.
- **Vensim**: Програма для моделювання системної динаміки, особливо корисна для управління проектами.
- **AnyLogic**: Інструмент для моделювання бізнес-процесів і логістичних систем із підтримкою агентного моделювання.

1.1.5 Імітаційне моделювання

Відтворення реальних процесів із можливістю оцінки поведінки при різних сценаріях і умовах.
Основні сфери застосування це: логістика, виробництво, охорона здоров'я.
Прикладами програм можна навести

- **Arena**: Інструмент для моделювання виробничих процесів, логістичних ланцюгів, управління чергами.
- **FlexSim**: Програма для 3D-імітаційного моделювання, використовується у сфері логістики та складського обслуговування.
- **AnyLogic**: Використовується для імітаційного моделювання з підтримкою різних типів симуляцій, включаючи агентне і дискретно-подійне моделювання.

1.1.6 Геопросторове моделювання

Моделювання та аналіз географічних даних для прогнозування змін у просторі. Частіше за все використовуються при досліжєні екології, містобудування, сільське господарство. Для цих цілей частіше за все використовують наступні програми

- **ArcGIS**: ГІС-система для збору, обробки та аналізу геопросторових даних.
- **QGIS**: Відкрита платформа для обробки географічної інформації, має вбудовані інструменти аналізу.
- **ERDAS IMAGINE**: Інструмент для аналізу супутникових знімків і геопросторових даних, часто використовується в екології та геології.

Ця класифікація і приклади програм демонструють широкий спектр можливостей для моделювання, які застосовуються у різних науках і галузях для вирішення складних завдань.Застосування комп'ютерного та імітаційного моделювання, особливо з використанням таких спеціалізованих інструментів, як MANA, відкриває широкі можливості для дослідження та вирішення складних завдань, що вимагають моделювання і прогнозування взаємодії елементів у різних сценаріях. Завдяки цьому, можна ефективно оцінювати і адаптувати стратегії в динамічних середовищах, приймати обґрунтовані рішення та забезпечувати високу надійність результатів для стратегічного і тактичного планування.

1.2 Типи моделей

1.2.1 Агент-орієнтоване моделювання (ABM)

Агент-орієнтоване моделювання (ABM) є підходом до моделювання складних систем, в якому система представляється як сукупність автономних агентів, що взаємодіють між собою та з навколишнім середовищем. Основні поняття ABM включають:

- **Агенти:**
  Агенти є основними будівельними блоками ABM. Вони представляють окремі об'єкти або одиниці, які мають свої властивості, поведінку та здатність до автономного прийняття рішень. Агенти можуть бути людьми, тваринами, транспортними засобами, організаціями тощо.

Їх головними характеристиками є:

- Автономія: Агенти мають здатність діяти незалежно, приймаючи рішення на основі внутрішніх правил і зовнішньої інформації.

- Адаптивність: Агенти можуть змінювати свою поведінку та стратегії у відповідь на зміни в середовищі.

- Правила поведінки:

Правила поведінки визначають, як агенти взаємодіють один з одним і з середовищем. Ці правила можуть бути простими або складними і зазвичай включають:

- Реакції на події: Як агент реагує на певні події або зміни в середовищі.
- Стратегії прийняття рішень: Як агент обирає дію з доступного набору можливостей.
- Взаємодії з іншими агентами: Як агенти співпрацюють, конкурують або взаємодіють іншим чином.

- Середовище

  Середовище є контекстом, у якому агенти діють і взаємодіють. Воно може бути фізичним, соціальним або віртуальним і включати різні ресурси, обмеження та умови, що впливають на поведінку агентів.

Елементами середовища можуть бути їх просторові характеристики тобто: Географічні або топологічні аспекти. Також розташування агентів, дистанції між ними або межі середовища. Ресурси, тобто об’єкти або особливості середовища які може використовувати агент. Як середовище змінюється під час симуляції.

- Взаємодії

Взаємодії між агентами та між агентами і середовищем є ключовим елементом ABM. Вони можуть бути:

- Прямими: Пряма тобто агенти на пряму взаємодіють один з одним

    - Непрямими: Дії агентів можуть вносити зміни у середовище що впливає на інших агентів.
- Емерджентні властивості

  Емерджентні властивості – це складні патерни та явища, що виникають з простих взаємодій між агентами. Ці властивості не можуть бути передбачені шляхом аналізу окремих агентів і є результатом колективної поведінки системи.
- Валідація та калібрування

  Валідація та калібрування є критично важливими аспектами ABM, що забезпечують достовірність і точність моделі. Валідація включає перевірку того, що модель правильно відображає реальний світ, тоді як калібрування полягає у налаштуванні параметрів моделі для узгодження з емпіричними даними

До переваг агент-орієнтованого моделювання можна віднести, здатність моделювати складні системи з нерегулярною динамікою, їх гнучкість у відображенні різних рівнів взаємодії а також можливість дослідження емерджентних властивостей системи

Головними проблемами є їх висока обчислювальна складність при введенні великої кількості агентів у симуляцію, необхідність спеціалізованих знань для розробки та інтерпретації даних моделі. Також важким аспектом є збір та інтеграція даних для валідації моделі.

1.2.2 Методи Монте-Карло

Методи Монте-Карло є класом алгоритмів, що використовують випадкове вибірку для наближення рішень складних або невизначених проблем. Ці

методи широко застосовуються у багатьох галузях, включаючи фізику, фінанси, інженерію та науку про дані.

Принцип випадкової вибірки

Методи Монте-Карло базуються на ідеї випадкової вибірки, де випадкові числа використовуються для моделювання різних можливих станів системи. Це дозволяє оцінити ймовірності різних результатів і обчислити статистичні властивості складних систем.

Основними кроками є:

- Генерація випадкових чисел: Створення випадкових чисел або послідовностей для моделювання можливих станів системи.
- Моделювання сценаріїв: Використання випадкових чисел для створення великої кількості можливих сценаріїв або траєкторій.
- Аналіз результатів: Обчислення статистичних властивостей результатів, таких як середнє значення, дисперсія, ймовірнісні розподіли та довірчі інтервали.

Області застосування

Методи Монте-Карло застосовуються у широкому спектрі галузей для вирішення завдань, що включають невизначеність, ризик та складні системи. Це можуть бути:

- Фінанси: Оцінка ризику інвестицій, ціноутворення деривативів, аналіз портфеля.
- Інженерія: Оцінка надійності систем, аналіз термінів виконання проектів, моделювання розповсюдження тріщини.
- Наука про дані: Оцінка ймовірності рідкісних подій, тестування гіпотез, симуляція експериментів.

Випадкові числа є ключовим елементом методів Монте-Карло. Важливо використовувати надійні генератори випадкових чисел для забезпечення коректності результатів. Генератори можна розподілити на 2 категорії:
Псевдовипадкові генератори: Алгоритми, що генерують числа, які виглядають випадковими, але є детермінованими.
Справжні випадкові генератори: Пристрої або методи, що використовують фізичні процеси для генерації випадкових чисел.
Для моделювання сценаріїв за допомогою методів Монте-Карло, використовуються випадкові числа для імітації можливих станів системи. Це включає:

- Вибір моделей: Визначення математичних моделей або функцій, що описують систему.
- Вибір параметрів: Визначення параметрів моделі, які можуть змінюватися у випадково.
- Симуляція: Проведення великої кількості симуляцій для оцінки результатів.

Результати симуляцій аналізуються для оцінки статистичних властивостей системи. Оцінюються наступні статистичні показники:

- Середнє значення параметрів
- Дисперсія
- Діапазон значень, у якому з певною ймовірністю знаходиться істинне значення.
- Розподіл ймовірностей різних результатів.

Методи Монте-Карло є потужним інструментом для вирішення проблем, що включають невизначеність та складність, забезпечуючи гнучкі та точні оцінки для широкого спектра застосувань. Головними перевагами

даного методу можна назвати: можливість моделювання широкого спектру складних систем, можливість оцінки систем з будь-яким рівнем складності, та забезпечення високої точності за рахунок проведення великої кількості симуляцій. Недоліками ж є: потреба в великій кількості обчислювальних ресурсів для проведення великої кількості симуляцій, важливість використання справжньо-випалкових генераторів випадкових чисел.

1.2.3 Клітинні автомати

Клітинні автомати (CA) є математичною моделлю, яка використовується для симуляції складних систем, що складаються з великої кількості взаємодіючих компонентів. Вони широко застосовуються для моделювання просторово-часових процесів у різних галузях, таких як фізика, біологія, екологія та інформатика.

Основою клітинних автоматів є ґратка, яка складається з дискретних клітин. Кожна клітина може мати один з обмеженого числа станів.
Ґратка може бути одновимірною, двовимірною або багатовимірною. Форма та розташування клітин на ґратці, що може бути прямокутною, трикутною, гексагональною тощо.

Кожна клітина в ґратці може знаходитися в одному з кількох можливих станів. Стан клітини змінюється з часом відповідно до правил переходу.

- Бінарні стани: Наприклад, "0" або "1".
- Багатостанові: Наприклад, різні рівні концентрації, кольори або інші властивості.

Правила переходу визначають, як стан кожної клітини змінюється на основі станів сусідніх клітин. Ці правила можуть бути локальними, тобто залежати тільки від найближчих сусідів клітини. Прикладами можуть бути:

- Мажоритарні правила: Стан клітини визначається станом більшості її сусідів.
- Правила суми: Стан клітини змінюється залежно від суми станів її сусідів.

Сусідство клітини визначає набір клітин, стан яких впливає на зміну стану даної клітини.

Можемо розділити їх 2 головні категорії

- Сусідство фон Неймана: Включає чотири сусіди (зверху, знизу, зліва, справа) для двовимірної ґратки.
- Сусідство Мура: Включає вісім сусідів (включаючи діагональні) для двовимірної ґратки.

Еволюція системи у клітинних автоматах відбувається через дискретні кроки. На кожному кроці стан кожної клітини оновлюється відповідно до правил переходу.

Клітинні автомати здатні генерувати складні та непередбачувані патерни з простих локальних правил. Ці патерни є емерджентними властивостями системи, які не можуть бути передбачені тільки з правил переходу окремих клітин. До таких властивостей можна віднести: самоорганізацію, виникнення структурованих патернів без централізованого контролю, фрактали геометричні структури, що повторюються на різних масштабах. А також хаотичність системи тобто складна та непередбачувана поведінка системи

Клітинні автомати використовуються для моделювання різноманітних процесів у природі та техніці. Наприклад це може бути моделювання процесів дифузії, гідродинаміка, фазові переходи. В біології це може бути моделювання росту клітин, моделювання екосистем чи поширення епідемій. В інформатиці це можна використати для криптографії чи моделювання алгоритмів.

В цілому клітинні автомати є потужним інструментом для дослідження динамічних систем, що дозволяє моделювати складні процеси з використанням простих локальних правил. Клітинні автомати (CA) є потужним методом для моделювання складних систем. Однак, як і будь-яка методика, вони мають свої переваги та недоліки.

Наприклад до переваг можна віднести те що клітинні автомати базуються на простих локальних правилах, які відносно легко визначити та інтерпретувати. Також результати роботи клітинних автоматів досить легко візуалізувати. Клітинні автомати здатні генерувати складні патерни поведінки з простих локальних правил, що в свою чергу дозволяє досліджувати емерджентні властивості системи. Вони є досить гнучкими у налаштуванні правил і структури сусідства для моделювання різних типів процесів. Клітинні автомати підходять для паралельних обчислень, оскільки стан кожної групи клітин може бути оновлено незалежно від інших.
До недоліків можна віднести: Простота правил клітинних автоматів може призводити до надмірного спрощення реальних процесів, що обмежує їх точність і реалістичність. А також багато моделей клітинних автоматів не враховують випадкові фактори, які можуть бути важливими для певних систем. Важливо також враховувати, що правила клітинних автоматів базуються на взаємодіях лише між сусідніми клітинами, що може не

враховувати більш глобальні або віддалені взаємодії, які мають місце в реальних системах.

Клітинні автомати є потужним інструментом для моделювання складних систем з локальними взаємодіями. Вони пропонують простоту, гнучкість і здатність генерувати емерджентні властивості, що робить їх корисними для широкого спектра застосувань. Однак їх простота може бути і їхнім обмеженням, оскільки вона може призводити до надмірного спрощення реальних процесів. Крім того, вони можуть вимагати значних обчислювальних ресурсів і мати труднощі з валідацією і калібруванням моделей.

1.2.4 Дискретно-подієве моделювання

Дискретно-подієве моделювання (DES) є методом симуляції, який використовується для аналізу систем, що змінюються у визначені моменти часу в результаті виникнення окремих подій. DES є ефективним інструментом для моделювання систем з дискретними змінами стану, таких як виробничі процеси, мережі обслуговування, логістичні системи тощо.

Подія — це окремий інцидент, який змінює стан системи в певний момент часу. Події є основними будівельними блоками дискретно-подієвого моделювання.

Події можна розділити на 2 основні категорії:

- Заплановані події: Тобто події, що відбуваються через визначенні інтервали часу
- Випадкові події: Тобто події, що виникають у випадкові моменти часу

Стан системи визначається набором змінних, які описують поточний стан усіх її компонентів. Стан змінюється тільки в моменти виникнення подій.

Приклади станів:

- Число клієнтів у черзі.
- Статус обладнання (працює/не працює).
- Рівень запасів на складі.

У DES час є дискретним і змінюється від одного моменту виникнення події до іншого. Відлік часу може бути абсолютним або відносним залежно від вимог моделі.

Особливості часової шкали:

- Дискретність: Час між подіями може бути змінним або фіксованим.
- Послідовність: Події обробляються в порядку їх виникнення.

Черга подій (або список подій) зберігає всі події, заплановані для обробки. Кожна подія має позначку часу, яка вказує, коли ця подія має бути оброблена.

Симуляційний годинник відстежує поточний час у моделі. Він оновлюється при обробці кожної події до часу цієї події.

До основних функцій симуляційного годинника відносяться:

- Синхронізація: Зміна стану системи відбувається відповідно до симуляційного часу.
- Керування: Використовується для контролю за тривалістю симуляції.

DES дозволяє збирати та аналізувати статистичні дані про поведінку системи протягом симуляційного періоду.

Приклади статистичних показників:

- Середні значення: Наприклад, середній час очікування клієнтів у черзі.
- Дисперсії: Вимірювання розсіювання даних, таких як варіація часу між подіями.
- Гістограми: Розподіл значень певних показників, таких як довжина черги

Багато систем, які моделюються за допомогою DES, включають управління ресурсами, такими як машини, оператори або транспортні засоби.

Прикладами можуть бути:

- Визначення, які ресурси використовуються для виконання подій.
- Ведення черг очікування для ресурсів, коли вони зайняті.

До переваг цього методу можна віднести: Можливість моделювати системи з детальним описом подій і станів, він відходять для широкого спектра застосувань, цей метод дозволяє збирати і аналіз детальних даних про систему. Але цей підхід потребує досить високої точності моделі та даних необхідний для моделювання сценаріїв.

DES часто застосовується для моделювання виробничих процесів: моделювання потоків роботи, задіяного обладнання та його обслуговування. Моделювання логістичних процесів: управління запасами та транспортними системами. У сервісних системах, наприклад: моделювання черги в банках, лікарнях, кол-центрах. В телекомунікаціях для аналізу пропускної здатності мереж та управління трафіком.

Дискретно-подієве моделювання є потужним інструментом для аналізу та оптимізації систем з дискретними змінами стану, забезпечуючи точність та гнучкість у моделюванні різних типів процесів і систем.

1.2.5 Метод скінченних елементів (FEA)

Метод скінченних елементів (Finite Element Analysis, FEA) є чисельним методом, який використовується для розв'язання задач з області інженерії та фізики. Він дозволяє моделювати поведінку складних структур та систем шляхом розбиття їх на менші, простіші частини, звані скінченними елементами.

Основою методу скінченних елементів є розбиття безперервної області на скінченну кількість елементів.

Етапи дискретизації:

- Розбиття області: Розбиття складної геометрії на прості елементи, такі як трикутники, квадрати, тетраедри тощо.
- Вузли: Визначення точок (вузлів) на межах і всередині елементів, де будуть обчислюватись значення шуканих величин.

При формулюванні задачі методом скінченних елементів використовуються математичні рівняння, що описують фізичні закони, такі як рівняння рівноваги, теплопровідності або електромагнітного поля.

Типові задачі:

- Механіка твердого тіла
- Теплопровідність
- Гідродинаміка
- Електромагнетизм

Розв'язок задачі шукається у вигляді наближеної функції, що виражається через значення у вузлах елементів і інтерполяційні функції. Прикладами апроксимаційних функцій можна привести

- Лінійні функції: Простий тип апроксимації, де значення змінюється лінійно між вузлами.
- Квадратичні та кубічні функції: Більш складні типи апроксимації для підвищення точності розрахунків.

Для кожного елемента формується локальна система рівнянь, яка потім об'єднується в глобальну систему рівнянь для всієї моделі. На початку виводяться рівняння для кожного окремого елементу. На наступному кроці ці рівняння поєднуються у єдину систему з урахуванням граничних умов. Глобальна система рівнянь розв'язується за допомогою чисельних методів, таких як методи Гауса, ітераційні методи або методи розкладання.

Це можуть бути:

- Прямі методи: Використовуються для точного розв'язування систем рівнянь (наприклад, метод Гауса).
- Ітераційні методи: Використовуються для великих систем, де прямі методи неефективні (наприклад, методи Крилова (Krylov–Bogoliubov averaging method)).

Після розв'язування системи рівнянь отримані результати аналізуються для оцінки поведінки моделі. На цьому етапі візуалізується розподіл величин, таких як напруги, деформації, температури тощо. Та обчислюються похідні величин, такі як головні напруги, фактори безпеки та інші інженерні характеристики.

Важливим аспектом моделювання є визначення граничних умов та прикладених навантажень, які впливають на розв'язок задачі.

Це можуть бути:

- Фіксовані значення шуканих величин у межах певної області.
- Задання потоків або градієнтів на межах області

Валідація та верифікація моделі є важливими кроками для забезпечення коректності та надійності результатів.

Етапи валідації та верифікації:

- Верифікація: Перевірка правильності реалізації математичної моделі та обчислювальних алгоритмів.
- Валідація: Порівняння результатів моделювання з експериментальними даними або теоретичними розрахунками.

Метод скінченних елементів є потужним інструментом для чисельного аналізу різноманітних фізичних процесів. Він дозволяє моделювати складні системи з високою точністю та ефективністю, забезпечуючи надійні результати для інженерних та наукових задач

Висновок до розділу 1

У цьому розділі було розглянуто основні методи моделювання реальних подій, з особливим акцентом на комп'ютерному та імітаційному підходах, які мають високу цінність для сучасного дослідження складних динамічних систем. Комп'ютерне моделювання, завдяки своїм можливостям обробки великого обсягу змінних та налаштування численних умов, дозволяє відтворювати поведінку складних систем і досліджувати їхню динаміку в умовах змінних параметрів, що робить його незамінним для точного аналізу процесів у кліматології, транспортних системах, фінансових ринках та інших сферах.

Системи моделювання реальних подій, такі як агент-орієнтоване моделювання (ABM), методи Монте-Карло, клітинні автомати (CA), дискретно-подієве моделювання (DES) та метод скінченних елементів (FEA), є потужними інструментами для аналізу, прогнозування та оптимізації складних систем і процесів у різних галузях науки та техніки. Кожен з цих методів має свої унікальні особливості, переваги та обмеження, що робить їх придатними для різних типів задач.

Саме ж моделювання реальних подій є критично важливим інструментом для розуміння, аналізу та оптимізації складних систем у різних сферах діяльності. Вибір конкретного методу моделювання залежить від характеру задачі, вимог до точності та доступних обчислювальних ресурсів. Комбінація різних методів може забезпечити ще більшу ефективність та точність моделювання, надаючи можливість отримання всебічного розуміння досліджуваних систем та процесів.

Імітаційне моделювання, і особливо MANA, є унікальним інструментом для відтворення поведінки агентів в інтерактивних і динамічних умовах. MANA

надає можливість моделювати різні сценарії, змінюючи характеристики агентів і умови середовища для оптимізації процесів, таких як управління ресурсами або стратегічне планування в кризових ситуаціях. Завдяки широкій гнучкості та зручності налаштування, MANA дозволяє досліджувати взаємодію елементів системи, створюючи надійні прогнози для ухвалення ефективних рішень.

Застосування сучасних методів комп'ютерного і імітаційного моделювання надає дослідникам і фахівцям потужні інструменти для глибокого аналізу і розробки адаптивних стратегій, особливо у випадках, що потребують врахування різноманітних факторів ризику і сценаріїв розвитку подій. Такі інструменти підвищують точність прогнозів і надійність рішень, що сприяє оптимізації управлінських і дослідницьких процесів.

РОЗДІЛ 2.
ОГЛЯД СИСТЕМ МОДЕЛЮВАННЯ РЕАЛЬНИХ ПОДІЙ

Моделювання реальних подій є дуже важливим методом для тестування, оптимізації та дослідження складних систем і явищ. Проте створення та проведення реалістичних і надійних симуляцій вимагає ретельного проектування, реалізації та валідації моделей, алгоритмів і даних, що представляють цільовий сценарій. У цьому розділі ми розглянемо деякі з найбільш ефективних методів моделювання реальних подій які активно використовуються

2.1 Постановка задачі

Основна задача полягає у створенні модульної та адаптивної моделі, здатної навчати агентів відповідно до заданих критеріїв, використовуючи можливості Unity для візуалізації й обробки складних агентних взаємодій. Дослідницька задача передбачає оптимізацію агентної поведінки за допомогою нейронних мереж для забезпечення гнучкої адаптації під час взаємодії агентів в змінному середовищі.
У нашій агентній моделі кожен агент повинен мати можливість "бачити" своє оточення та адаптувати поведінку залежно від отриманої інформації. Модель враховує типи агентів, які можуть взаємодіяти між собою, обмінюючись інформацією відповідно до їхніх функцій та цілей.

2.1.1 Основні аспекти моделі

Ініціалізація параметрів середовища передбачає створення умов, у яких діятимуть агенти, щоб реалістично відобразити просторові обмеження й особливості. Основним елементом є висотна мапа, яка відтворює рельєф території та задає тривимірну структуру простору, в якому функціонуватимуть агенти. Висотна мапа забезпечує доступність різних рівнів території, що може впливати на мобільність агентів, визначаючи зони, де пересування потребує більше ресурсів або взагалі є неможливим. Також середовище наповнюється природними обмеженнями, які представлені специфічними географічними умовами, наприклад, лісами та річками. Ці елементи відіграють важливу роль у формуванні просторових бар'єрів, якими агенти повинні або скористатися, або обійти. Взаємодія з цими обмеженнями стимулює агента до прийняття адаптивних рішень, уникаючи, наприклад, важкопрохідні зони або зони, що можуть бути вигідними з точки зору захисту.

Алгоритми сприйняття середовища створюють у агентів здатність орієнтуватися в заданому просторі, аналізувати його та приймати рішення на основі отриманих даних. Сприйняття здійснюється через спеціальні механізми, які дають можливість агентам сканувати навколишнє середовище в певному радіусі, дозволяючи їм помічати та визначати об'єкти, зони та інші сутності, важливі для виконання завдань. Крім того, сприйняття дозволяє агентам оцінювати придатність виявлених зон для пересування чи взаємодії. Наприклад, при зустрічі з річкою або схилом агент оцінює, чи зможе він подолати цю перешкоду або краще шукати інший шлях. Таким чином, механізми сприйняття забезпечують агентам базову інформованість про

територію і дозволяють орієнтуватися в умовах, які можуть змінюватися відповідно до середовищних факторів чи дій інших агентів.
Взаємодія між агентами реалізується через розробку системи комунікації, яка дозволяє обмінюватися інформацією залежно від типу агентів та їхніх функцій у системі. Комунікація відбувається на основі певних критеріїв та залежить від того, які види інформації потрібні для досягнення цілей. Агенти можуть обмінюватися даними про структуру території, перешкоди, наявність інших агентів, потенційні небезпеки чи вигідні маршрути. Наприклад, агент розвідник може повідомити групу про виявлену перепону, в той час як інші агенти можуть передати інформацію про найшвидший шлях до цілі.
Взаємодія також може відбуватися на рівні спільного планування, де кілька агентів синхронізують свої дії для вирішення завдань колективно. Такий підхід підвищує гнучкість та адаптивність агентів, дозволяючи їм взаємодіяти в середовищі з високою змінністю й непередбачуваністю, що в цілому підсилює ефективність та реалізм симуляції.

2.1.2 Створення середовища

Створення умов оточення, орієнтуючись на GIS-дані з відкритих ресурсів та дані з супутника Sentinel-1, надає можливість відтворити реалістичну й складну структуру навколишнього середовища. Використання таких джерел дозволяє детально моделювати ландшафт, зокрема відображати його висоту, рельєфні особливості, а також розташування природних елементів, таких як ліси, річки, водойми та навіть тимчасові чи сезонні зміни в структурі ландшафту. В цьому контексті інформація із Sentinel-1 є особливо цінною, оскільки надає дані про радіолокаційні спостереження за землею, незалежно

від часу доби та погодних умов, що значно розширює можливості для точної симуляції природних умов.
Отримані умови можуть стати основою для адаптивної поведінки агентів, що дає змогу моделям відображати складні сценарії, у яких поведінка агентів змінюється відповідно до середовища. Використання актуальної і точної інформації значно підсилює можливості агентів щодо взаємодії з реальними обмеженнями, формуючи поведінку, яка базується не тільки на встановлених правилах, а й на обліку природних характеристик середовища.

2.1.3 Планувальник задач

Механізм функціонування моделі будується навколо планувальника задач, який виконує роль центрального вузла для координації дій агентів і забезпечення узгодженості їх поведінки в рамках моделі. Кожен агент, незалежно від його типу чи ролі, діє через планувальник, що надає єдину структуру для реєстрації й управління всіма діями в системі. Це означає, що, перш ніж будь-яка дія буде виконана, агент має зареєструвати її в планувальнику. Кожна з цих дій отримує певний пріоритет, що допомагає визначити її черговість у загальній черзі завдань.
Планувальник розподіляє дії агентів на основі їхнього пріоритету, дозволяючи найбільш важливим завданням виконуватися першими, що є ключовим для імітації поведінки агентів у реалістичних умовах, коли певні дії можуть мати негайний або вирішальний вплив на перебіг подій. Наприклад, якщо агент помічає небезпеку або нову можливість, планувальник надасть цій дії високий пріоритет для негайного виконання,

тоді як рутинні дії з нижчим пріоритетом можуть бути виконані пізніше або навіть відкладені.

Завдяки планувальнику система не просто керує чергою задач, а й створює складну модель поведінкових стратегій, у якій дії можуть бути переглянуті й адаптовані залежно від обставин. Це надає агентам можливість виявляти, реєструвати та виконувати дії в умовах, що змінюються, не порушуючи загальної послідовності процесів. Такий механізм дозволяє підтримувати динаміку моделі, в якій події розвиваються послідовно, а агенти реагують на них відповідно до пріоритетів своїх завдань і чинних умов. У підсумку планувальник дозволяє ефективно координувати активності, підтримуючи чітку чергу й відповідний розподіл ресурсів для кожного агента, що підвищує надійність і точність симуляції.

2.1.4 Використання нейронних мереж як механізм прийняття рішень для агентів моделі

Інтеграція між Unity та обчислювальними процесами нейронної мережі, зокрема на основі локальної великої мовної моделі (LLM), є важливим і складним етапом, який ми зараз активно реалізуємо. Для цього розробляються алгоритми зв’язку, що здатні забезпечити двосторонню комунікацію між компонентами Unity та нейронною мережею. Основне завдання полягає в тому, щоб передавати та обробляти інформацію з Unity у реальному часі, надаючи агентам змогу отримувати оновлення про зміни в їхньому оточенні. Це створює основу для адаптивної поведінки, коли агенти здатні моментально коригувати свої дії на основі нових параметрів середовища.

Unity, як основний рушій, служить платформою для моделювання віртуального середовища, у якому діють агенти. У цьому середовищі фіксуються різноманітні зміни, такі як оновлення параметрів рельєфу, умов оточення, появи чи зникнення ресурсів або перешкод. Алгоритми взаємодії з нейронною мережею дозволяють автоматично передавати ці зміни до LLM, що дає можливість агентам оцінювати нові умови й аналізувати їх у контексті своїх цілей. Нейронна мережа, своєю чергою, обробляє ці дані та надає агентам відповідні інструкції чи рекомендації, які вони використовують для прийняття адаптивних рішень.

Таке рішення значно розширює потенціал симуляції. Завдяки LLM агенти можуть взаємодіяти не лише з навколишнім середовищем, але й один з одним на більш складному рівні, що дозволяє відтворювати процеси колективного навчання та соціальної поведінки. Наприклад, агенти можуть «навчатися» один від одного, обмінюючись інформацією про ресурсні зони, небезпеки або інші ключові елементи середовища, що вивчається. Це стає можливим завдяки здатності LLM зберігати і використовувати контекст попередніх взаємодій, допомагаючи агентам накопичувати знання та використовувати їх у процесі прийняття рішень.

Ключовою особливістю цього підходу є можливість контекстного самонавчання агентів. Локальна LLM забезпечує збереження контексту дій та середовищних змін, що дає агентам змогу застосовувати накопичений досвід для розробки оптимальних стратегій. Це дозволяє моделі імітувати не лише реактивну, але й проактивну поведінку агентів: вони можуть передбачати певні наслідки на основі попереднього досвіду та, відповідно, планувати свої дії на кілька кроків уперед. Таким чином, агент у симуляції стає суб'єктом,

здатним до стратегічного мислення, що включає прогнозування та модифікацію поведінки відповідно до ситуації.

Використання локальної LLM також оптимізує обчислювальні витрати, оскільки модель, розгорнута на локальному сервері, може обробляти значний обсяг даних без потреби в зовнішньому підключенні. Це підвищує швидкість та стабільність симуляції, що особливо важливо для складних багатокористувацьких середовищ, де необхідно підтримувати високу продуктивність системи.

Поєднання Unity та локальної LLM також дозволяє створювати сценарії, де агенти не тільки реагують на зміни у середовищі, але й зможуть адаптувати свою поведінку з урахуванням прогнозованих змін..

Висновок до розділу 2

У другому розділі ми детально розглянули ключові аспекти розробки агентної моделі, орієнтуючись на інтеграцію системи моделювання з Unity та локальною великою мовною моделлю (LLM). Основною метою нашої роботи є створення умов, у яких агенти зможуть активно взаємодіяти з навколишнім середовищем, адаптуючи свою поведінку в реальному часі на основі змін, що відбуваються. Розглянуті параметри середовища, такі як висотна мапа, а також обмеження, що визначаються оточенням, зокрема лісами та ріками, надають агентам необхідну інформацію для ухвалення рішень.

Механізм функціонування моделі, заснований на планувальщику задач, дозволяє агентам реєструвати свої дії, а виконання цих дій відбувається відповідно до пріоритетів та черги. Це сприяє організованій та логічній поведінці агентів у рамках моделі. Завдяки інтеграції з LLM, агенти отримують можливість не лише реагувати на зміни у середовищі, але й адаптувати свої стратегії поведінки, спираючись на контекст та накопичений досвід.

На даний момент ми працюємо над реалізацією інтеграції між системою моделювання та моделями, розгорнутими через локальну LLM. Це дозволить створити гнучку та інтелектуально насичену систему, в якій агенти зможуть приймати рішення на основі динамічного аналізу середовища. Таким чином, наш підхід забезпечує значне розширення можливостей симуляції та моделювання поведінки агентів у складних умовах, що відкриває нові горизонти для подальших досліджень і розробок у цій сфері.

РОЗДІЛ 3
РОЗРОБКА СИСТЕМИ МОДЕЛЮВАННЯ РЕАЛЬНИХ ПОДІЙ НА ОСНОВІ МОДЕЛЮВАННЯ АГЕНТІВ З ВИКОРИСТАННЯМ НЕЙРОННОЇ МЕРЕЖІ.

В даному розділі розглянуто основні етапи створення системи моделювання реальних подій, що базується на агентному підході та використовує нейронну мережу як механізм прийняття рішень. Поставлено завдання розробки функціональної системи, здатної демонструвати емерджентну поведінку агентів та адаптуватися до змін умов у процесі моделювання.

3.1 Розробка планувальника задач та системи моделювання подій

Першим етапом дослідження є побудова системи моделювання, ключовим елементом якої є планувальник задач. Планувальник забезпечує узгодженість і послідовність дій агентів у системі, дозволяє координувати їхню взаємодію із середовищем та між собою.

Основні функції планувальника:

- Пріоритизація задач: планувальник визначає порядок виконання задач залежно від важливості та терміновості.
- Моніторинг стану системи: постійний контроль за станом агентів і середовища.

3.1.1 Абстрактний клас Queueable

В основі планувальника задач лежить клас Queueable є базовим класом у системі моделювання агентів, який забезпечує управління чергами завдань

(queues) для агентів. Він відповідає за створення, реєстрацію, виконання та видалення черг. Ось детальний опис функціональності цього класу:

Основні можливості класу:

1. Робота з чергами завдань:
    - Клас дозволяє створювати черги завдань з різними параметрами, такими як частота виконання, пріоритет, і затримка перед початком.
    - Черги використовуються для управління діями агентів у симуляції.
2. Інтеграція з контролером:
    - Клас зберігає посилання на об'єкт контролера (AbstractController), який керує виконанням черг у масштабах всієї симуляції.
    - При створенні нової черги вона автоматично реєструється в контролері.
3. Автоматичне ініціалізування:
    - У методі Awake відбувається автоматичне налаштування посилання на контролер, якщо воно ще не встановлене.
    - У методі Init створюється початковий список черг.
4. Гнучке створення черг:
    - Є кілька перевантажених методів для створення черг:
        - З типовими параметрами (частота = 1, пріоритет = NORMAL, затримка = 0).
        - З вказаними параметрами частоти, пріоритету та затримки.
    - Черги можуть бути створені як негайно, так і з відкладеним стартом через корутину (IEnumerator).
5. Реєстрація та видалення черг:

- Реєстрація черг: додає чергу до списку та реєструє її в контролері.
- Видалення черг: видаляє чергу зі списку, дерегіструє її в контролері і знищує її.
- При знищенні об'єкта (OnDestroy) усі черги, пов'язані з ним, видаляються, щоб уникнути витоку пам'яті.

Структура коду:

Поля:

- _queues— список черг, які пов'язані з цим об'єктом.
- _controller— посилання на контролер, який керує чергами.

Властивості:

- Controller— забезпечує доступ до контролера.
- Queues— забезпечує доступ до списку черг.

Методи:

- Awake— налаштовує контролер під час ініціалізації.
- Init— ініціалізує список черг.
- CreateQueue— створює нову чергу з різними параметрами.
- CreateDefaultQueue— універсальний метод створення черг з типових значень.
- RegisterQueue— додає чергу до списку й реєструє її в контролері.
- DeregisterQueue— видаляє чергу зі списку й дерегіструє її в контролері.
- DestroyQueue— знищує чергу.
- OnDestroy— видаляє всі черги при знищенні об'єкта.

Цей клас призначений для того, щоб агент або будь-який інший об'єкт, який взаємодіє з чергами, міг легко додавати, видаляти та керувати завданнями, а також інтегруватися із загальною системою симуляції.

3.1.2 Черга

Клас Queue визначає механізм управління чергами у планувальнику, що дозволяє задавати порядок та частоту виконання дій у симуляції. Він працює із завданнями (екземплярами делегатів), які виконуються відповідно до заданих параметрів. Основними параметрами черги є частота виконання (вказує, як часто черга повинна запускатись), пріоритет (визначає, які черги виконуються раніше, де нижчі значення мають вищий пріоритет) та момент початку виконання (номер кадру, з якого черга реєструється для виконання). Черга також має назву, яка відповідає імені методу, прив'язаного до делегата, і власника, який відповідає за її створення.

Клас підтримує три рівні порядку виконання: HIGH, NORMAL та LOW. Пріоритет може бути заданий явно або автоматично визначений на основі порядку. Додатково є можливість валідації значень пріоритету, щоб вони залишалися в межах від 0 до 1000.

Метод Step запускає дію, пов'язану з чергою, якщо вона має власника, а також є віртуальним, що дозволяє перевизначити його в похідних класах. Крім цього, клас реалізує інтерфейс IComparable, що дозволяє порівнювати черги між собою за пріоритетом, забезпечуючи правильне сортування черг у планувальнику.

Черга інтегрується із системою планувальника для управління порядком і частотою виконання завдань, дозволяючи створювати складні сценарії симуляції з контролем за виконанням дій у часі.

3.1.3 Планувальник задач

Клас Scheduler відповідає за управління чергами в агентно-орієнтованій моделі ABM. Його головна функція полягає в координації виконання черг Queues з різними пріоритетами і частотою у симуляції. Основний функціонал планувальника задача полягають у: Реєстрації черг що виконуються під час симуляції. Черги поділяються на черги що виконуються на кожному кроці симуляції та на ті що виконуються через певні інтервали чи на певних кроках моделювання.

```
        /// <summary>
        /// Registers queue to be executed every tick
        /// for queues with step = 1
        /// </summary>
        /// <param name="queue">Queue to register see <see
cref="Queue"/></param>
        private void RegisterFutureQueueExecutedEveryTick(ABMQueue queue)
        {
            Dictionary<int, List<ABMQueue>> queueExecutedEveryTickPriority;
            if (queuesExecutedEveryTick.TryGetValue(queue.QueueOrder, out
queueExecutedEveryTickPriority))
            {
                if (queueExecutedEveryTickPriority.ContainsKey(queue.Priority))
```

```
                queueExecutedEveryTickPriority[queue.Priority].Add(queue);
            else
                queueExecutedEveryTickPriority.Add(queue.Priority, new
List<ABMQueue>() { queue });
        }
        else
        {
            queuesExecutedEveryTick.Add(queue.QueueOrder, new
Dictionary<int, List<ABMQueue>>());
            queuesExecutedEveryTick[queue.QueueOrder].Add(queue.Priority,
new List<ABMQueue>() { queue });
        }
    }
```

Вихідний код 3.1Метод для реєстрації черг які виконуються на кожному кроці симуляції

3.1.4 Контроль симуляції

Клас AbstractController є базовим контролером для симуляцій у середовищі, який забезпечує управління агентною системою та планувальником. Він відповідає за ініціалізацію симуляції, ведення обліку агентів, керування чергами у планувальнику та контроль часу виконання. Клас реалізує основні методи для реєстрації та зняття реєстрації агентів і черг, що дозволяє гнучко додавати або видаляти їх під час роботи симуляції. Клас підтримує механізм паузи симуляції, дозволяючи зупинити її на визначеному кадрі за допомогою Unity Editor. Завдяки інтеграції з Unity методи Awake і LateUpdate

забезпечують автоматичне ініціалізування контролера та виконання симуляції відповідно до оновлення кадрів. Загалом цей клас забезпечує основний функціонал для запуску, керування і моніторингу симуляції, використовуючи планувальник і агентів як основні компоненти, і дозволяє створювати складні симуляційні сценарії.

Клас містить приватний список агентів _agents, який використовується для відстеження всіх зареєстрованих агентів у симуляції. Ці агенти є окремими сутностями, що можуть взаємодіяти між собою чи з середовищем.

```
/// <summary cref="AbstractAgent">
/// List of agents in simulation
/// </summary>
private List<AbstractAgent> _agents;

/// <summary cref="ABMScheduler">
/// Reference to scheulder
/// </summary>
private ABMScheduler _scheduler;
```

Вихідний код 3. 2 Репрезентація агентів та планувальника задач в контролері симуляції

Планувальник _scheduler відповідає за виконання дій у певній послідовності на кожному кроці симуляції.

При виклику методу Init відбувається ініціалізація: створюється список агентів, встановлюється початковий час симуляції (на основі кількості кадрів з моменту запуску програми), створюється новий об'єкт планувальника і налаштовується базовий лог. Це забезпечує базові умови для роботи симуляції.

```
        /// <summary>
        /// Controller initializer method. Initializes the agent list and records
simulation start time (in frames)
        /// </summary>
        public virtual void Init()
        {
            _agents = new List<AbstractAgent>();
            simStartTime = Time.frameCount;
            currentTick = 0;
```

```
            this.Scheduler = new ABMScheduler();
            log = new Utils.Log(this.GetType().Name);
            Utils.Logger<AbstractController>.DebugLogError(Log.Name + "
initialized");

        }
```

Вихідний код 3. 3 Ініціалізація контролера симуляції

Завдяки такому підходу кожний контролер симуляції має свій особистий планувальник задач, а кожен елемент симуляції пов'язаний з контролером що його ініціалізує. Тому ми можемо проводити більшу кількість симуляцій паралельно. Враховуючи що частина сценарію керується псевдовипадковим генератором випадкових чисел, це надає нам опції: Ми можемо проводити більше ніж один потенційний сценарій поведінки невідомої цілі, і обрати найбільш вірогідну чи обрати сценарій поведінки агентів який покриває найбільшу кількість випадків з невідомою цілью.

Метод Step виконується на кожному кадрі. Він збільшує лічильник поточного кроку симуляції currentTick, викликає метод Tick у планувальника для виконання черги дій, а також вимірює час виконання цих операцій. Зібрані дані про час додаються в журнал log для подальшого аналізу. Крім того, клас обчислює середню затримку виконання симуляції _delayAvarage, використовуючи накопичену затримку _delayCumulative.

```
        /// <summary>
        /// Steps the simulation forward by one tick.
        /// This method is called every frame by Unity.
        /// It increments the current tick, executes the scheduler, and logs the
time it took to execute the scheduler.
        /// </summary>
        public virtual void Step()
        {
            if (Time.frameCount - simStartTime <= 0)
                return;

            if (!isSimulationPaused)
            {
                currentTick++;

                Stopwatch sw = new Stopwatch();
                sw.Start();
```

```
                Scheduler.Tick();
                sw.Stop();
                _delayCumulative += sw.Elapsed.Milliseconds;
                if (Time.frameCount - simStartTime !=0)
                {
                    _delayAvarage = _delayCumulative / (Time.frameCount -
simStartTime);
                    log.AverageDelay.Add(currentTick, _delayAvarage);
                }
                log.TickTime.Add(currentTick, sw.Elapsed.Milliseconds);

            }
        }
```

Вихідний код 3. 4 Метод просування симуляції

Задля роботи з агентами передбачені методи RegisterAgent і DeregisterAgent, які додають або видаляють агентів із списку _agents. Аналогічно, для черг є методи RegisterQueue і DeregisterQueue, що керують реєстрацією черг у планувальнику. Це дозволяє динамічно змінювати стан симуляції залежно від необхідності. Клас також забезпечує функціонал для управління паузою симуляції через методи PauseSimulationAtFrame і UnPauseSimulation. Якщо кількість кроків симуляції досягає встановленого значення endFrame, симуляція автоматично ставиться на паузу.

Загалом, AbstractController є основним інструментом для створення симуляційного середовища, де можна керувати агентами, чергами та динамікою симуляції, підтримуючи високу гнучкість і можливість моніторингу процесу. Також активно використовує механізм ведення журналів через об'єкт log, де зберігаються дані про час виконання кроків і середні затримки. Це важливо для аналізу продуктивності симуляції.

3.1.5 Процес симуляції в системі

Процес симуляції в системі, реалізованій на основі класу AbstractController, проходить кілька ключових етапів, від створення агентів до завершення симуляції.

Перед запуском симуляції викликається метод Init() контролера. У цьому методі:

- Ініціалізується список агентів.
- Встановлюється початковий час симуляції (simStartTime) у кадрах.
- Створюється об'єкт планувальника (ABMScheduler) для управління чергами подій.
- Створюється об'єкт для ведення журналу (Utils.Log).

Після ініціалізації симуляція переходить у режим очікування виконання першого кроку.

Коли створюється новий агент, він ініціалізується і реєструється в контролері. Це робиться через метод RegisterAgent(AbstractAgent agent). Агент додається до списку _agents, який зберігає всі активні агенти симуляції. На цьому етапі агент може отримати початкові параметри, такі як положення в середовищі, стан або цілі.

```
        /// <summary>
        /// Adds and agent to the register list.
        /// </summary>
        /// <param name="agent">The agent to add</param>
        public void RegisterAgent(AbstractAgent agent)
        {
            this.Agents.Add(agent);
        }

        /// <summary>
        /// Registers a <see cref="ABMQueue"/> to the scheduler
        /// </summary>
        /// <param name="queue">The <see cref="ABMQueue"/> to register</param>
        public void RegisterQueue(ABMQueue queue)
        {
            this.Scheduler.RegisterQueue(queue);
        }
```

Вихідний код 3. 5 Методи реєстрації агентів та черг їх дій в контролері симуляції

Кроки симуляції виконуються в методі Step(), який викликається на кожному кадрі Unity через LateUpdate() або незалежно за будь якою іншою логікою. Наприклад це може бути реалізовано через елементи інтерфейсу.

- Лічильник currentTick збільшується на 1.
- Викликається метод Scheduler.Tick(), який виконує всі події, заплановані в чергах для поточного кроку.
- Використовується таймер (Stopwatch), щоб обчислити час, потрібний для виконання кроку, і зберегти цю інформацію в журналі.
- Розраховується середня затримка (_delayAvarage) для аналізу продуктивності симуляції.

Кожен агент може виконувати свої дії в рамках симуляції. Це може включати:

- Взаємодію з іншими агентами або середовищем.
- Виконання своїх завдань або оновлення стану.
- Запис своїх дій у черги, які потім обробляються планувальником.

Дії агентів залежать від логіки, реалізованої в їхніх власних класах, і часто синхронізуються через планувальник.

У процесі симуляції можуть реєструватися нові агенти або видалятися існуючі.

- Новий агент додається через RegisterAgent().
- Агента можна видалити через DeregisterAgent().

Також у планувальник можуть додаватися або видалятися черги подій через методи RegisterQueue() та DeregisterQueue().

Симуляція може бути призупинена або завершена: Якщо кількість кроків симуляції досягає значення endFrame, метод PauseSimulationAtFrame() автоматично зупиняє виконання (якщо запущено в редакторі Unity).

Симуляція також може бути вручну призупинена шляхом встановлення прапорця isSimulationPaused у true. Коли симуляція завершується:

- Зупиняється виконання планувальника.
- Дані, зібрані в журналі (log), можуть бути збережені або проаналізовані для оцінки результатів симуляції.
- Список агентів очищається.
- Можливе виконання дій очищення ресурсів або підготовка до нового запуску симуляції. Процес симуляції в системі побудований на циклічному виконанні подій і взаємодії між агентами через механізм черг. Це дозволяє моделювати складні системи, де кожен агент виконує свої дії, взаємодіє з іншими, а планувальник координує цей процес для досягнення заданих цілей або виявлення закономірностей.

3.2 Конкретна реалізація абстрактних класів задіяних у процесі моделювання сценарію пошуку людини яка загубилась під час туристичного походу.

3.2.1 SimulationController задачі та методи

Клас SimulationController відповідає за управління симуляцією пошуку віртуального агента в симуляційному середовищі. Він реалізує функціональність створення агентів, цільових зон, пошукових груп, управління їхньою взаємодією, а також запис і збереження результатів симуляції. Він є конкретною реалізацією класу AbstractController і відповідно наслідує усі попередні методи та властивості, додаючи необхідні методи та

властивості для поточного сценарію а також інтеграції з нейронною мережею.

Спочатку ініціалізуються ціль і пошукові групи. Для кожного офіцера створюється певна кількість пошукових агентів, які прив'язуються до нього. Агенти і офіцери отримують унікальні ідентифікатори і прив'язуються до зон, де вони будуть працювати. Ціль генерується в випадковій точці визначеної зони. Кожен агент має можливість взаємодіяти з іншими агентами і офіцерами, що дозволяє коригувати стратегії пошуку на основі отриманих даних. Цей процес керується алгоритмами, що враховують відстань до останньої знайденої підказки місцезнаходження цілі а також визначеної на початку зони потенційного місцезнаходження цілі, навколишні умови та реакцію на змінювані фактори, як-от збої в пошукових маршрутах чи несподівані зміни в оточенні.

Клас також використовує систему випадкових подій для додавання непередбачуваних елементів до симуляції. Це включає випадкові зміни в поведінці агентів, наприклад, їх відхилення від маршруту чи нові неочікувані підказки, що можуть з'являтися в процесі пошуку. Такі випадковості додають елементи реалізму, дозволяючи агентам проявляти людську невизначеність. Це є одним з найголовніших аспектів необхідних для проявлення емерджетності під час моделювання

Система підказок активно використовується для відслідковування прогресу пошуку та корекції стратегій. Кожна підказка записується в окремий лог, що дозволяє аналізувати ефективність пошукових операцій і виявляти, на якій стадії агенти найбільше наближаються до цілі. Крім того, клас здійснює постійну оцінку ефективності роботи кожної пошукової групи. На основі цих даних коригуються стратегії подальших пошуків, що включає

зміну напрямку, додавання нових агентів або зміщення фокусної точки пошуку, якщо одна з груп не знаходить ціль вчасно. Завершення симуляції завжди включає підбиття підсумків щодо часу, витраченого на пошук, кількості агентів, які брали участь, та успішності пошуку. Всі ці результати зберігаються у вигляді звітів, які можна переглядати для подальшого аналізу та оптимізації майбутніх симуляцій. Чи для порівняння впливу саме емерджентності на результати симуляції при збережені інших параметрів незмінними. Важливим аспектом є візуалізація даних про пошукові операції. Використовуючи графічні інтерфейси, клас надає візуалізацію маршруту агентів, відмітки підказок, а також поточне місце знаходження цілі в реальному часі. Це дозволяє операторам або аналітикам краще розуміти процес симуляції і коригувати його для більш реалістичних сценаріїв.

Таким чином, SimulationController є потужним інструментом для створення складних моделей пошукових операцій, в яких враховуються різноманітні фактори, що можуть змінюватися під час симуляції, забезпечуючи при цьому гнучкість і можливість подальшого аналізу результатів. А також він може бути значно розширеним у подальшому для створення більш комплексних сценарій чи навколишніх умов .

3.2.2 Середовище в сценарії моделювання

Середовище в сценарії моделювання є критично важливим компонентом, оскільки воно безпосередньо впливає на поведінку агентів і їхні можливості виконувати завдання в рамках симуляції. У цьому проекті середовище моделюється з використанням даних GIS-систем, зокрема карт висот і призначення землі, що дозволяє створити реалістичне і динамічне

середовище для агентів. Дані з Sentinel-1 і Sentinel-2 надають високоякісну інформацію про географічні особливості місцевості, що включає різні типи покриття, такі як ліси, водні об'єкти, сільськогосподарські території, а також висоти рельєфу. Завдяки цьому проект може точно відтворити різноманітні умови, з якими агент може стикатися під час своєї діяльності. Наприклад, через наявність лісів або водних бар'єрів, агент може змінювати свою стратегію пересування або сприйняття навколишнього середовища.

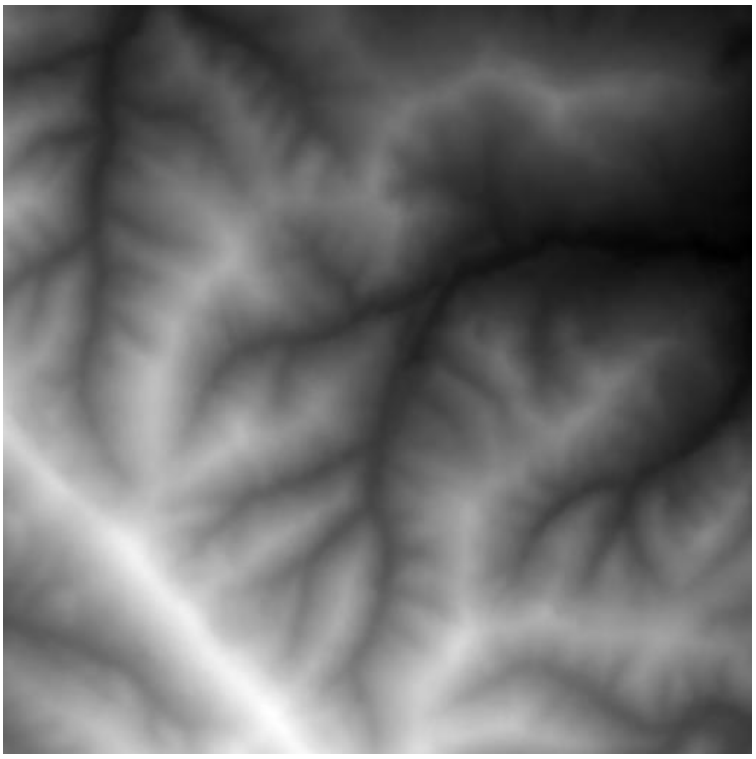


Рис.3.1. Приклад мапи висот для генерації ландшафту у сценарії пошуку.

Середовище також включає в себе фізичні характеристики території, такі як типи поверхні (дороги, ліси, відкриті простори), що визначають можливості пересування агентів. Наприклад, деякі агенти можуть мати більшу швидкість руху по дорогах, але знижувану ефективність у лісах чи болотистих місцевостях. Це додає елемент складності до моделі, дозволяючи

точніше відтворити реальні умови, в яких агенти можуть взаємодіяти з навколишнім середовищем.

Таким чином, середовище в моделюванні не лише забезпечує контекст для дій агентів, але й впливає на їхню ефективність, поведінку та взаємодію з іншими об'єктами в системі. Використання реальних геопросторових даних надає можливість створювати складні сценарії, що більш точно відображають реальність і дозволяють здійснювати більш ефективне планування і оцінку результатів. Поєднання геопросторових даних з навігаційною системою Unity надає нам можливість досить чітко керувати параметрами пошуку маршруту та швидкістю агентів під час моделювання.

Модульна інтеграція даних у цьому проекті передбачає гнучкість у роботі з різними типами інформації, що зберігається в коректних ГІС-форматах або може бути виражена у стандартизованих форматах. Завдяки цьому, система може ефективно працювати з даними різних джерел і форматів, що дозволяє легко додавати нові набори даних без необхідності переглядати або змінювати структуру моделювання.

Рис.3.2. Приклад негативу для створення зон лісу у сценарії пошуку.

Завдяки використанню стандартних ГІС-форматів, таких як GeoTIFF, Shapefile, GML, а також можливості перетворення даних у інші популярні формати, система підтримує інтеграцію даних з різних джерел. Це дозволяє використовувати дані, наприклад, від супутникових знімків, картографічних матеріалів, знімків з безпілотників, а також дані від різноманітних сенсорів. ГІС-формати є універсальними та підтримуються численними бібліотеками та інструментами для обробки просторових даних. Важливо, що модульна інтеграція забезпечує можливість ефективного обміну даними між різними компонентами системи. Якщо дані мають бути отримані або передані з інших систем або джерел, вони можуть бути легко конвертовані у відповідний формат, що підтримується ГІС-інструментами або іншими програмними засобами. Наприклад, набір даних про призначення землі чи рельєф території з однієї платформи можна експортувати у формат, зручний для використання

в іншій платформі, без втрати важливої інформації або спотворення результатів.

Така інтеграція дозволяє створювати більш точні і детальні моделі середовища, використовуючи широкий спектр даних з різних джерел, що робить систему гнучкою та адаптивною до змін або доповнень у майбутньому. Крім того, це дозволяє вбудовувати нові інструменти та технології, забезпечуючи безперешкодний потік інформації та її безпечне оброблення в рамках всієї моделі.

3.2.3 Конкретні реалізації агентів в сценарії пошуку

Клас TargetAgent реалізує агентний об'єкт у середовищі, людину що загубилась під час туристичного походу в лісі в горах, який взаємодіє з навколишнім середовищем і здійснює переміщення по сцені в Unity з використанням системи навігації на основі NavMesh. Ось основні елементи його роботи:

У методі PostInit встановлюються основні компоненти контроллером що ініціалізує агента, такі як SimulationController і NavMeshAgent, а також налаштовується детектор (Detector), який відстежує навколишні об'єкти, і підключається подія на зміну виявлених об'єктів. Додатково, для агента встановлюються параметри навігації.

У класі є методи для руху агента: Move і StartMoveSequence. У першому методі агент використовує NavMeshAgent для руху, а у другому - ініціюється послідовність руху, де агент рухається та перевіряє відстань до мети за допомогою методу CheckDistanceToCurrentTargetDestination.

```
    private void CheckDistanceToCurrentTargetDestination()
    {
```

```
        float distance = Vector3.Distance(this.transform.position,
currentTargetDestinationPosition);
        if(distance < distanceToDestinationThreshold){ //reached target
            isNearDestination = true;

            navMeshAgent.isStopped = true;

            DestroyQueue("CheckDistanceToCurrentTargetDestination");
            DestroyQueue("Move");

            SetTargetDirection(null);

            //SetupStationary();
        }
        else{
            isNearDestination = false;
        }
    }
```

Вихідний код 3. 6. Метод для перевірки умов під ходження до поточної цілі

Методи Move та CheckDistanceToCurrentTargetDestination виконуются виключно через планувальник задач. Створюється черга для кожного з них і під час виконання черги CheckDistanceToCurrentTargetDestination може бути видалено обидві черги.

В даного типу агента перевірка досить проста, адже в сценарії він просто намагається вийти з лісу.

Метод SetTargetDirection визначає напрямок, в якому агент має рухатися, до заданої цілі. Якщо параметр destination дорівнює null, метод використовує значення за замовчуванням, яке міститься в полі cardinalTarget. Це означає, що агент буде орієнтуватися на цю ціль, якщо конкретне призначення не було передано. Далі, метод обчислює напрямок від поточної позиції агента до позиції цілі (визначеної через destination). Для цього використовується вектор, який обчислюється як різниця між позицією цілі та позицією агента. Цей вектор нормалізується, щоб отримати напрямок. Потім є ймовірність того, що агент вийде з правильного напрямку (за допомогою випадкової величини). Якщо випадкове значення менше або рівне заданій

ймовірності, агент випадковим чином змінює напрямок у межах 60 градусів від початкового напрямку, що робить його рух менш передбачуваним. В цьому випадку виводиться повідомлення в лог, яке показує старий і новий напрямки. Після визначення напрямку, розраховується точка на землі, до якої агент має рухатись. Ця точка знаходиться на відстані nextWaypointDistance від поточної позиції агента в напрямку цілі.

Цей метод створює рух агента до випадкової точки в заданому напрямку, обчислюючи висоту місцевості і випадкові відхилення для більш реалістичного переміщення. Вся інша логіка працює через зв’язку між UnityNavMesh та планувальником задач де за візуальне переміщення та оновлення координат агенту відповідає Unity а за обробку даних та координацію процесу планувальник задач.

3.2.4 Реалізація пошукових агентів в сценарії пошуку

Для опису конкретної реалізації пошукових агентів у сценарії пошуку, нам потрібно описати клас, який моделює поведінку пошукового агенту. Клас SearchOfficerAgent та SearchAgent призначені для моделювання агента, який бере участь у пошуковій операції в заданому середовищі. Цей клас розширює абстрактний клас AbstractAgent і представляє персонажа, який взаємодіє з іншими агентами та об'єктами в симуляції, виконуючи команди, що надходять від системи. Агент має низку властивостей, що визначають його поведінку, таких як ідентифікатор для ідентифікації, посилання на контролер симуляції, NavMeshAgent для руху та цільову зону. Крім того, є кілька налаштовуваних параметрів, зокрема ймовірність помилкового руху,

параметри точок маршруту та швидкість руху. В цілому його базові функції майже не відрізняються від попереднього типу.

```
private void RequestCommandFromLLM()
{
    var simulationToLLM = simulationController.simulationToLLM;
    if (waitingForResponse == false)
        waitingForResponse = true;
        simulationToLLM.RequestCommandFromLLM(this);
```

Вихідний код 3.7. Метод для ініціалізації обміну інформацією між симуляцією та нейронною мережею

Головна різниця полягає в тому що між пошуковими агентами реалізована система комунікації через контролер симуляції а також те що пошукові агенти визначенні як офіцери мають систему для комунікації з нейронною мережею та передачі інформації про оточення та стан системи через контролер симуляції. Звичайні пошукові агенти мають два базових сценарії поведінки, вони чи супроводжують офіцера до яких вони прив’язанні чи шукають навколишнє середовище на певній відстані від їх офіцера. Все інше це результат команд які вони отримують від офіцера. Який в свою чергу отримує команди від нейронної мережі. В нашому випадку використовуються InterLM.

3.2.5 Реалізація зв’язку між нейронною мережею та симуляцією

Клас SimulationToLLM відповідає за взаємодію між сценарієм пошукової операції та зовнішньою моделлю, яку представляє LLM, через API запити. Клас має посилання на об'єкт SimulationController, який збирає дані про поточний стан симуляції, зокрема, інформацію про пошукові операції.

Клас має декілька основних функцій, що дозволяють надсилати запити до LLM для генерації команд пошуковим агентам. Коли метод

RequestCommandFromLLM викликається, він збирає інформацію про поточний сценарій, створює запит з конкретними даними для LLM, а потім надсилає цей запит через HTTP POST запит.

```
    public void RequestCommandFromLLM(SearchOfficerAgent searchOfficerAgent,
string rejectReason = "")
    {
        // Step 1: Prepare the scenario data for the LLM prompt
        string scenarioData = simulationController.CollectScenarioInfo();

        // Step 2: Create a complete system + user prompt
        string systemPrompt = GenerateSystemPrompt();
        string userPrompt = GenerateUserPrompt(scenarioData, searchOfficerAgent);
        userPrompt += rejectReason;

        // Step 3: Send the prompt to the LLM and handle the response
        StartCoroutine(SendScenarioToLLM(systemPrompt, userPrompt, response =>
        {
            HandleResponse(response, searchOfficerAgent);
        }));
    }
```

Вихідний код 3.8. Метод для відправки запиту

У відповідь LLM повертає команду, яку потрібно виконати агенту, наприклад, перемістити агента до конкретної точки або провести сканування в зазначеній локації.

```
public string CollectScenarioInfo(bool end = false)
{
    string scenarioInfo = $"Last known approximate target position is in circle
with center {targetPotentialLocation} within {targetZone.transform.localScale.x}
radius\n";

    SortedDictionary<int, string> sortedRecievedCommands = new
SortedDictionary<int, string>();
    SortedDictionary<int, string> sortedExecutingCommands = new
SortedDictionary<int, string>();
    SortedDictionary<int, string> sortedExecutedCommands = new
SortedDictionary<int, string>();

    foreach (var recievedCommand in this.recievedCommands)
    {
        sortedRecievedCommands.Add(recievedCommands.IndexOf(recievedCommand),
recievedCommand.CommandPrompt);

        if (executingCommands.Contains(recievedCommand))

sortedExecutingCommands.Add(executingCommands.IndexOf(recievedCommand),
recievedCommand.CommandPrompt);

        if (executedCommands.Contains(recievedCommand))
```

```
            sortedExecutedCommands.Add(executedCommands.IndexOf(recievedCommand),
recievedCommand.CommandPrompt);
    }
    if (end)
        foreach (var recievedCommand in sortedRecievedCommands)
        {
            scenarioInfo += "Executed: " + sortedExecutedCommands.Where(e =>
e.Value == recievedCommand.Value).FirstOrDefault().Value + "\n";
        }
    else
    {
        SortedDictionary<int, string> usefulCommands = new SortedDictionary<int,
string>();
        if (sortedExecutedCommands.Count > 5)
        {
            for (int i = 0; i < 5; i++)

usefulCommands.Add(sortedExecutedCommands.Reverse().Skip(i).First().Key,
sortedExecutedCommands.Reverse().Skip(i).First().Value);
        }
        else
        {
            foreach (var command in sortedExecutedCommands)
            {
                usefulCommands.Add(command.Key, command.Value);
            }
        }

        foreach (var usefulCommand in usefulCommands)
        {
            scenarioInfo += "Executed: " + usefulCommand.Value + "\n";
        }
    }

    foreach (var agent in officerAgentsInstances)
    {
        scenarioInfo +=
$"SearchOfficr[{agent.GetComponent<SearchOfficerAgent>().ID}] now locatd at
{agent.transform.position} \n";
    }
    if (end)
    {
        if (agentClues.Count > 0)
        {
            foreach (var kvp in agentClues)
            {
                scenarioInfo += $"Agent[{kvp.Key.ID}] found clues of target path
at [";

                foreach (var clue in kvp.Value)
                {
                    scenarioInfo += $"Clue found at{clue.transform.position}";
                }

                scenarioInfo += "]";
            }
        }
```

```
    }
    else
    {
        if (agentClues.Count > 0)
        {
            var usefulCluesCount = agentClues.Count % 5;
            HashSet<GameObject> usefulClues = new HashSet<GameObject>();
            float minimalDistanceToTarget = float.MaxValue;
            for (int i = 0; i < usefulCluesCount; i++)
            {
                GameObject closestClue = null;
                foreach (var kvp in agentClues)
                {
                    foreach (var clue in kvp.Value)
                    {
                        float distanceToTargetFromClue =
Vector3.Distance(clue.transform.position, targetInstance.transform.position);
                        if (distanceToTargetFromClue < minimalDistanceToTarget
&& !usefulClues.Contains(clue))
                        {
                            minimalDistanceToTarget = distanceToTargetFromClue;
                            closestClue = clue;
                        }
                    }
                }
                if (closestClue != null)
                    usefulClues.Add(closestClue);
            }
            foreach (var clue in usefulClues)
            {
                scenarioInfo += $"Clue found at{clue.transform.position}";
            }
        }
    }

    return scenarioInfo;
}
```

Вихідний код 3.9. Метод SimulationController для сбіру та форматування поточного стану сценарію до LLM

Запит до LLM включає системний опис завдання, в якому йдеться про ролі і можливості агентів, а також користувацький опис сценарію, що містить інформацію про пошукові позиції і можливі обмеження на рух або виконання завдань. Після отримання відповіді від LLM, клас обробляє її, розбираючи команду і передаючи її до відповідного агента для виконання. Якщо команда неправильна або не може бути виконана, агент відхиляє її.

```
  private void HandleResponse(string response, SearchOfficerAgent
searchOfficerAgent)
  {
      // Parse the response and extract commands
      var jsonResponse = JsonUtility.FromJson<OSTResponse>(response);
      string commands = jsonResponse.choices[0].message.content;

      string[] commandLines = commands.Split('\n');
      ABM.Utils.Command agentCommand = new ABM.Utils.Command();

      var command = commandLines[0];



          if (command.Contains("Command:") && command.Contains("Move"))
          {
              agentCommand.CommandPrompt = command;



              ExecuteMoveCommand(agentCommand, searchOfficerAgent);
          }
          else if (command.Contains("Command:") && command.Contains("Send"))
          {
              agentCommand.CommandPrompt = command;

              ExecuteScanCommand(agentCommand, searchOfficerAgent);
          }
          else
          {
              Debug.Log($"Error: {command}");
              RequestCommandFromLLM(searchOfficerAgent, $"{command} This command
incorectly formated. \r\nHere is an example response:\r\nCommand: Move
SearchOfficer[1] to (5, 0, 10). Command: Send Units under SearchOfficer[1] to (8,
0, 12). Command: SearchOfficer[2] scans the area around (6, 0, 11) \r\n\r\nYou
must give only 1 command in response.");
          }

  }
```

Вихідний код 3.10. Обробка відповіді від LLM

Клас також має методи для створення повідомлень для LLM і перевірки правильності координат, що надаються в команді. Вся комунікація з LLM виконується через корутину, яка здійснює асинхронний запит до API та обробляє результат.

```
    private IEnumerator SendScenarioToLLM(string systemPrompt, string userPrompt,
System.Action<string> onResponse)
    {
        // Manually construct the JSON payload
        string jsonData = $@"
```

```
    {{
        ""model"": ""internlm2_5-20b-chat"",
        ""messages"": [
            {{ ""role"": ""system"", ""content"":
""{EscapeJson(systemPrompt)}"" }},
            {{ ""role"": ""user"", ""content"": ""{EscapeJson(userPrompt)}"" }}
        ],
        ""max_tokens"": 50,
        ""temperature"": 0.7
    }}";

        Debug.Log($"Sending JSON Payload: {jsonData}");

        using (UnityWebRequest request = new UnityWebRequest(API_URL, "POST"))
        {
            request.SetRequestHeader("Content-Type", "application/json");
            //request.SetRequestHeader("Authorization", $"Bearer {API_KEY}");
            request.uploadHandler = new
UploadHandlerRaw(Encoding.UTF8.GetBytes(jsonData));
            request.downloadHandler = new DownloadHandlerBuffer();

            yield return request.SendWebRequest();

            if (request.result == UnityWebRequest.Result.Success)
            {
                string responseText = request.downloadHandler.text;
                Debug.Log($"Response: {responseText}");
                onResponse?.Invoke(responseText);
            }
            else
            {
                Debug.LogError($"LLM Request Failed: {request.error}\nResponse:
{request.downloadHandler.text}");
                onResponse?.Invoke(null); // Handle failure gracefully
            }
        }
    }
```

Вихідний код 3.11. Відправлення сценарію до LLM

Клас дозволяє керувати операцією пошуку з підвищеною автоматизацією, де пошукові агенти отримують конкретні вказівки по виконанню дій на основі реальних або згенерованих даних. Це дозволяє реалізувати складні стратегії для пошукових операцій в 3D середовищі.

3.3 Проблеми використання LLM в контексті оператора

На жаль, однією з головних проблем, з якими ми зіткнулися під час розробки, стала відсутність спеціально натренованих моделей, які б відповідали нашим задачам. Використання існуючих великих мовних моделей виявило кілька суттєвих проблем. По-перше, такі моделі мають генералізовану природу, яка не завжди дозволяє їм адаптуватися до специфічних вимог нашого проекту. По-друге, існують певні обмеження щодо обробки великих обсягів даних та ефективного керування симуляціями, які створюють додаткові виклики для інтеграції моделей у наш підхід.

Ці особливості, зокрема, включають труднощі з інтерпретацією результатів, оптимізацією прийняття. Це свідчить про необхідність розробки моделей, які не тільки будуть генералізованими, але й враховуватимуть специфіку завдань нашого проекту.

Інша серйозна перепона це проблема галюцинацій у великих мовних моделях. Це одна з ключових проблем, яка суттєво обмежує надійність використання великих мовних моделей у критично важливих системах. У випадках, коли модель не має чіткої відповіді, вона намагається "заповнити прогалину", генеруючи припущення, які виглядають переконливо. Сучасні дослідження зосереджені на розробці алгоритмів, які краще відокремлюють достовірну інформацію від потенційно вигаданої. Такі як: Self-Consistency Decoding: Використання багаторазових спроб генерації для вибору найбільш узгодженого варіанту. Retrieval-Augmented Generation (RAG): Моделі поєднують генерацію з пошуком по зовнішній базі даних, що дозволяє зменшити ймовірність помилок. А також головною перепоною ми вважаємо що моделі не здатні «розуміти контекст» це твердження є дещо категоричним

і потребує уточнення. LLM демонструють здатність імітувати розуміння контексту, проте це не справжнє розуміння, як у людини. Моделі вчаться розпізнавати статистичні залежності між словами, фразами чи текстами в межах великих датасетів, але це не те саме, що розуміння сенсу. Моделі здатні обробляти обмежену кількість інформації в межах одного запиту або попереднього діалогу (визначеного розміром вікна контексту, наприклад, 32768 або більше токенів). Моделі не «пам'ятають2 попередніх взаємодій поза межами одного сеансу, якщо спеціально не інтегровані механізми збереження контексту. Моделі оперують символами (токенами) без жодного розуміння їхнього реального значення. Моделі не мають концептуальної бази для структурування знань або відображення причинно-наслідкових зв’язків. Відповіді моделі залежать від того, як вона була навчена, але вона не здатна створювати справжні "нові" знання. Модель генерує тексти, спираючись на ймовірності появи токенів, а не на аналізі смислових аспектів.

Це призводить до того що: Модель може правильно відповісти на запит, якщо запит коректний і чіткий. Але в складніших випадках модель може втратити контекст, змішати деталі чи створити вигадану інформацію. Якщо контекст вимагає багаторівневого аналізу або розгляду кількох джерел інформації, модель часто видає відповідь, яка лише виглядає переконливою. При тривалих діалогах модель може суперечити сама собі, оскільки їй важко узгоджувати інформацію на великих відстанях у контексті.

Використання великих мовних моделей (LLM) як оператора та координатора в пошуковій операції породжує серйозні проблеми через їх обмеження у розумінні контексту. У такій складній задачі, як координація дій і прийняття рішень, недоліки моделей можуть мати критичні наслідки. Пошукова операція включає багатоетапну взаємодію, аналіз великої кількості

інформації та оновлення стратегії в реальному часі. LLM обмежені «вікном контексту» і часто «забувають» попередні важливі деталі. Модель може приймати неузгоджені рішення, забувати про попередні інструкції або втрачати важливі координати чи умови, що призводить до помилок у пошукових сценаріях. Модель може вигадувати інформацію (галюцинації) під час генерації тексту, видаючи її за правдиву. Це особливо небезпечно, коли модель використовується для прийняття важливих рішень. Пошукові операції часто залежать від точного аналізу просторових даних, карт, маршрутів і взаємозв’язків між об’єктами. LLM не мають внутрішньої моделі простору і працюють лише з текстовими описами, які можуть бути некоректно інтерпретовані. Помилки у вказівках, пов’язаних із простором (напрямки, відстані, розташування). Хибна інтерпретація топографічних даних або нездатність об'єднати просторові й текстові дані для прийняття рішень. LLM не здатні до справжнього аналізу причинно-наслідкових зв'язків, що є критично важливим у пошукових сценаріях, де потрібно оцінювати ризики, ймовірності або враховувати змінні фактори. Нездатність обґрунтувати, чому певний маршрут або стратегія пошуку були обрані, що ускладнює корекцію дій. Пошукові операції часто вимагають роботи з різними джерелами даних — текстовими, візуальними, сенсорними. Нерозуміння багатомодальних даних стандартні LLM працюють тільки з текстом і не здатні інтегрувати інформацію з різних типів джерел. Моделі не завжди повторювано генерують однакові відповіді на ідентичні запити через стохастичну природу генерації тексту.

Через всі ці фактори та відсутність глибокого розуміння контексту, причинно-наслідкових зв’язків, обмеження в роботі з просторовими даними та проблеми з адаптацією до динамічних умов великі мовні моделі є

небезпечними в ролі оператора або координатора пошукової операції. Їхні обмеження можуть призводити до критичних помилок, втрати ресурсів і навіть ставити під загрозу життя людей. Для ефективного використання таких моделей необхідна їх інтеграція з іншими системами (геоінформаційними, сенсорними), а також розробка спеціалізованих моделей або механізмів, які зменшують ці недоліки.

Саме ці недоліки великих мовних моделей парадоксально роблять їх цінними інструментами для імітаційного моделювання людської поведінки у складних сценаріях. Люди, особливо в стресових або незвичних ситуаціях, також часто роблять помилки через неповне розуміння контексту або забування деталей. LLM, що "забувають" контекст або іноді втрачають логічний зв’язок між діями, можуть правдиво імітувати такі когнітивні помилки людей. Як приклад можна навести ситуацію де імітація рятувальників чи свідків, які плутають напрямки або неправильно інтерпретують ситуацію.

Люди іноді вигадують інформацію, щоб заповнити прогалини в знаннях або через впевненість у хибних припущеннях. LLM, що створюють галюцинації, можуть правдиво відтворювати такі ситуації, моделюючи, наприклад, пошукові помилки або неточні повідомлення. У сценаріях, де оператори помилково повідомляють про інформацію або приймають бажане за дійсне, такі моделі можуть бути корисними.

Стохастичність відповіді моделі добре імітує людську непослідовність та емоційну нестабільність у стресових умовах. Це дозволяє відтворювати реалістичні сценарії, де учасники приймають ірраціональні рішення. Як приклад можна привести ситуацію, в якій учасники пошукової операції

пропонують суперечливі стратегії або змінюють свій план без очевидних причин.

Великі мовні моделі через свої обмеження здатні правдоподібно імітувати людські помилки, когнітивні обмеження та поведінку в складних сценаріях. Їх використання в ролі агентів в симуляціях дозволяє створювати реалістичні сценарії для дослідження емерджентної поведінки, тестування стратегій та виявлення слабких місць у пошукових чи інших операціях. Це відкриває нові можливості для дослідження динаміки групових дій, аналізу ефективності рішень і розробки більш стійких систем координації.

Висновок до розділу 3

У розділі 3 ми детально описали процес розробки та реалізації основних елементів моделі, що дозволяє нам вивчати емерджентну поведінку в контексті агентних систем. Модель включає кілька основних складових: створення агентів, визначення їхньої поведінки та взаємодії з оточенням, а також механізм взаємодії між агентами для досягнення спільної мети. Ми розглянули, як ці агенти можуть змінювати свою поведінку в залежності від умов середовища та від того, як вони взаємодіють між собою. Описана архітектура дозволяє виявляти та аналізувати емерджентні властивості системи, такі як колективне ухвалення рішень, адаптація до змінюваних умов та еволюція взаємодії між агентами.

Одним із важливих аспектів цього проекту є здатність моделі демонструвати емерджентну поведінку, коли нескладні локальні взаємодії між агентами призводять до утворення складних глобальних явищ. Це дає змогу більш глибоко зрозуміти, як виникають та розвиваються складні системи, в яких результати не є простою сумою індивідуальних дій, а з'являються через взаємодію та самоорганізацію. Важливо зазначити, що наша модель не лише відображає механізми агентної поведінки, але й здатна виявляти нові патерни взаємодії, які можуть виникати в реальних системах.

Ми вважаємо, що досягнуті результати можуть стати основою для глибших досліджень у таких напрямках, як адаптація агентів до зміни середовища, поліпшення ефективності алгоритмів прийняття рішень або введення нових типів агентів з різними властивостями і поведінковими характеристиками. Також важливим є розвиток механізмів аналізу та візуалізації емерджентної поведінки, що дозволить краще розуміти, як такі процеси проявляються в реальних ситуаціях. Додатково, можна розглянути

можливість інтеграції цієї моделі в інші сфери, наприклад, для моделювання соціальних, економічних або екологічних систем, де емерджентна поведінка є важливою складовою. Модульність та гнучкість запропонованої архітектури дозволяють масштабувати систему для вирішення більш складних задач, що робить її перспективною для практичного використання в різних галузях. Отже, ми вважаємо, що даний проект має потенціал не тільки для дослідження емерджентної поведінки в контексті агентних систем, але й для подальшого розвитку в сторону більш складних та інтегрованих моделей, здатних вирішувати широкий спектр задач у різних сферах діяльності. Адже в процесі реалізації проекту також було досліджено можливості використання великих мовних моделей у ролі оператора та координатора пошуково-рятувальних операцій. Проведений аналіз показав, що попри наявні обмеження, такі як схильність до галюцинацій, відсутність просторового мислення, непослідовність та обмежене розуміння контексту, ці недоліки можуть стати ключовою перевагою для симуляційного моделювання людської поведінки. Завдяки своїй стохастичності та непередбачуваності, LLM здатні імітувати когнітивні помилки, емоційну нестабільність і нерішучість, властиві людям у стресових або незвичних умовах. Це дозволяє створювати реалістичні сценарії для моделювання дій груп у кризових ситуаціях, оцінювати вплив людського фактору на результати операцій, а також вивчати емерджентну поведінку в складних динамічних системах. У цілому проект демонструє, що використання великих мовних моделей має значний потенціал для досліджень емерджентної поведінки та подальшого розширення. Це відкриває нові перспективи для розвитку систем моделювання і вдосконалення процесів прийняття рішень у складних операціях.

РОЗДІЛ 4
РОЗРОБКА СТАРТАП ПРОЕКТУ

4.1. Загальна характеристика стартап-проекту

У цьому розділі описується концепція стартап-проекту, його мета, ключові елементи та перспективи розвитку. Стартап-проект є інноваційною ініціативою, яка спрямована на створення нового продукту чи послуги, здатної вирішувати актуальні проблеми, підвищувати ефективність процесів або створювати додаткову цінність для користувачів.

Стартап-проект спрямований на створення симуляційної платформи для дослідження поведінки агентів у різних сценаріях, таких як пошуково-рятувальні операції, екологічний моніторинг або реагування на надзвичайні ситуації, або в застосування у оборонній промисловості. Головна мета – надати користувачам інструмент для моделювання, аналізу та тестування складних сценаріїв з використанням сучасних технологій, таких як великі мовні моделі, нейронні мережі та геоінформаційні системи.

4.1.1 Інформаційна карта проекту

Таблиця 4.1 Іноформаційна карта проекту

1. Назва проекту	**Ad**vanced **A**gent-based **G**enerative **I**ntelligence for **O**perations (**AdaGiO**)
2. Автори проекту	Єлісєєв Максим
3. Коротка анотація *(не більше 1/3 сторінки)*	Ця робота присвячена розробці стартап-проекту, спрямованого на створення інноваційної симуляційної платформи для дослідження та аналізу поведінки агентів у складних сценаріях. Проект базується на використанні агентно-

	орієнтованих моделей, великих мовних моделей, нейронних мереж та геоінформаційних систем. Основна мета – забезпечити користувачів інструментом для моделювання, аналізу та оптимізації рішень у таких сферах, як пошуково-рятувальні операції, екологічний моніторинг та реагування на надзвичайні ситуації. Проект має великий потенціал для подальшого розвитку та впровадження у практичну діяльність.
4. Термін реалізації проекту	12 *Тривалість проекту (в місяцях)*
5. Необхідні ресурси	1. Матеріальні ресурси ПК – 95 632 грн Ноутбук – 35 857 грн Планшет – 48 730 грн Ноутбук ПМ – 86 000 грн ПК Розробника – 56 175 грн Меблі 25 000 * 3 грн 2. Фінансові ресурси Оренда приміщення 8400 грн/місяць Витрати AWS 400 -1300 грн/місяць Кава – 1200 грн/місяць Канцелярія – 200-400 грн/місяць Матеріальне забезпечення IT-спеціаліста = 20 000 грн/місяць Матеріальне забезпечення менеджера проекту = 17 000 грн/місяць Матеріальне забезпечення розробника = 16 000 грн/місяць Підписка на Weights&Biases 1875 грн/місяць Додаткові витрати – 4,942

	*95632 + 35857 +48730 + 86000 + 56175 + (1875 * 12) + (25000 * 3) + (8400 * 12) + (850 * 12) + (1200 * 12) + (300 * 12) + (20000 * 12) + (17000 * 12) +(16000 * 12) + 4942 = 1 189 836 грн* *Перелік усіх необхідних ресурсів (фінансових, матеріальних інтелектуальні та ін.)*
6. Опис проблеми, яку вирішує проект	1. Сучасні складні системи, такі як управління пошуково-рятувальними операціями, реагування на надзвичайні ситуації або оптимізація використання природних ресурсів, вимагають точного аналізу, прогнозування та координації дій у мінливих умовах. Існуючі інструменти часто не враховують динаміку поведінки учасників системи, обмежені у врахуванні зовнішніх факторів або не забезпечують достатньо гнучких засобів моделювання. 2. Однією з ключових проблем є недостатня інтеграція сучасних технологій, таких як великі мовні моделі, геоінформаційні системи (GIS) та агентно-орієнтовані моделі (ABM), для створення масштабованих та адаптивних симуляційних платформ. Це ускладнює імітацію складних сценаріїв та зменшує ефективність підготовки до непередбачуваних ситуацій. 3. Також проблемою є необхідність гнучкого підходу до моделювання людської поведінки, яка відрізняється емерджентними (виникаючими) властивостями та залежить від численних факторів, включаючи соціальні, екологічні та інформаційні.
7. Головні цілі та завдання проекту	Головна мета проєкту – створення інноваційної симуляційної платформи, яка дозволяє імітувати складні сценарії з використанням агентно-орієнтованих моделей (ABM), інтегрованих з великими мовними моделями (LLM), для аналізу, планування та координації дій у багатофакторному середовищі.
	1. Розробка основної архітектури платформи: 2. Створення ядра системи, яке підтримує моделювання агентів та взаємодію між ними. 3. Інтеграція великих мовних моделей для управління сценаріями та імітації поведінки. 4. Моделювання складних сценаріїв:

	5. Реалізація механізмів, які дозволяють імітувати емерджентну поведінку агентів. 6. Створення можливості адаптивного реагування на змінні умови в реальному часі. 7. Інтеграція геоінформаційних систем (GIS): 8. Використання геопросторових даних для точнішого відтворення середовища. 9. Забезпечення можливості роботи з великими наборами даних з реальних джерел. 10. Розробка алгоритмів координації та аналізу: 11. Впровадження інструментів для аналізу ефективності дій агентів у рамках симуляцій. 12. Створення інтелектуальних алгоритмів для координації дій між агентами.

8. Очікувані результати

На глобальному рівні:

- Створення універсальної симуляційної платформи:
- Розробка рішення, яке дозволить моделювати складні багатофакторні сценарії з використанням великих мовних моделей (LLM) та агентно-орієнтованих моделей (ABM).
- Інтеграція інноваційних технологій:
- Демонстрація можливостей поєднання штучного інтелекту та симуляційних систем для розв’язання складних завдань.
- Підвищення якості прийняття рішень:
- Надання інструментів для аналізу можливих сценаріїв і підтримки ухвалення рішень в реальному часі.
- Розвиток міждисциплінарного підходу:
- Сприяння співпраці між фахівцями з різних галузей, таких як екологія, рятувальні операції, урбаністика, логістика та військове планування.
- Внесок у розвиток штучного інтелекту:
- Покращення алгоритмів мовних моделей для імітаційного моделювання, що відкриє нові можливості їх застосування.

На локальному рівні:

- Рішення конкретної проблеми:
- Розробка симуляційного середовища для моделювання сценаріїв, таких як пошуково-рятувальні операції чи управління кризовими ситуаціями.
- Точне моделювання середовища:
- Використання локальних геопросторових даних для реалістичної імітації середовища та взаємодії агентів із ним.
- Оптимізація ресурсів:

- Надання інструментів для оптимального розподілу ресурсів у локальних сценаріях, наприклад, рятувальних місій або планування маршрутів.
- Ефективна взаємодія агентів:
- Забезпечення координації дій між агентами з урахуванням локальних умов та цілей.
- Тестування в реальних умовах:
- Використання платформи для локального аналізу проблемних ситуацій і визначення ефективних стратегій їх вирішення.
- Масштабування для специфічних задач:
- Адаптація системи під конкретні локальні потреби, що дозволить вирішувати завдання конкретного регіону чи організації.

4.2 Формування команди стартапу

Таблиця 4. 2 Спеціальність та ролі учасників команді

Спеціальність	**Роль**
IT-спеціаліст	Генератор ідеї & Дослідник
Маркетолог & Проектний менеджер	Дипломат & Організатор
Розробник	Реалізатор

РОЛІ в КОМАНДІ

IT-спеціаліст

Дослідження та Комунікації:

- *Роль:* **Дослідник.**
- *Особливості:* Екстраверт, комунікабельний, завжди в пошуках можливостей. Здатний налагоджувати контакти та привертати інформацію. Оптимістичний, але може втрачати ентузіазм з часом.

Технічна Реалізація та Технічний Експерт:

- *Роль:* **Технічний Експерт.**

- *Особливості:* Розважливий, стратегічний, бачить всі можливості. Може надати точну оцінку та розуміння технічних питань. Має великий вплив на технічні аспекти проекту.

Ідея та Креативність:

- *Роль:* **Генератор ідеї.**
- *Особливості:* Креативний, уявний, здатний вирішувати складні проблеми. Може бути незвичайним у підходах та ігнорувати випадковості. Зайнятий своєю творчістю, але може вносити ключові ідеї для проекту.

Маркетолог & Проектний менеджер

Координація та Керівництво:

- *Роль:* **Організатор.**
- *Особливості:* Зрілий лідер з впевненістю, відмінний організатор. Встановлює цілі, стимулює прийняття рішень та делегує повноваження. Може сприйматися як маніпулятор, але допомагає розвантажувати команду.

Дипломатія та Взаємодія:

- *Роль:* **Дипломат.**
- *Особливості:* Має навички взаємодії, м'якість та вміння уникати конфліктів. Спроможний слухати, будувати відносини та уникати проблемних ситуацій.

Організатор та Логіст:

- *Роль:* **Організатор.**
- *Особливості:* Комунікабельний, організований та сформований. Забезпечує внутрішню взаємодію та логістику у команді.

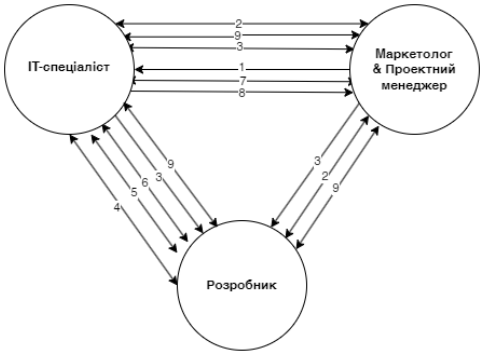
Розробник

Реалізація:

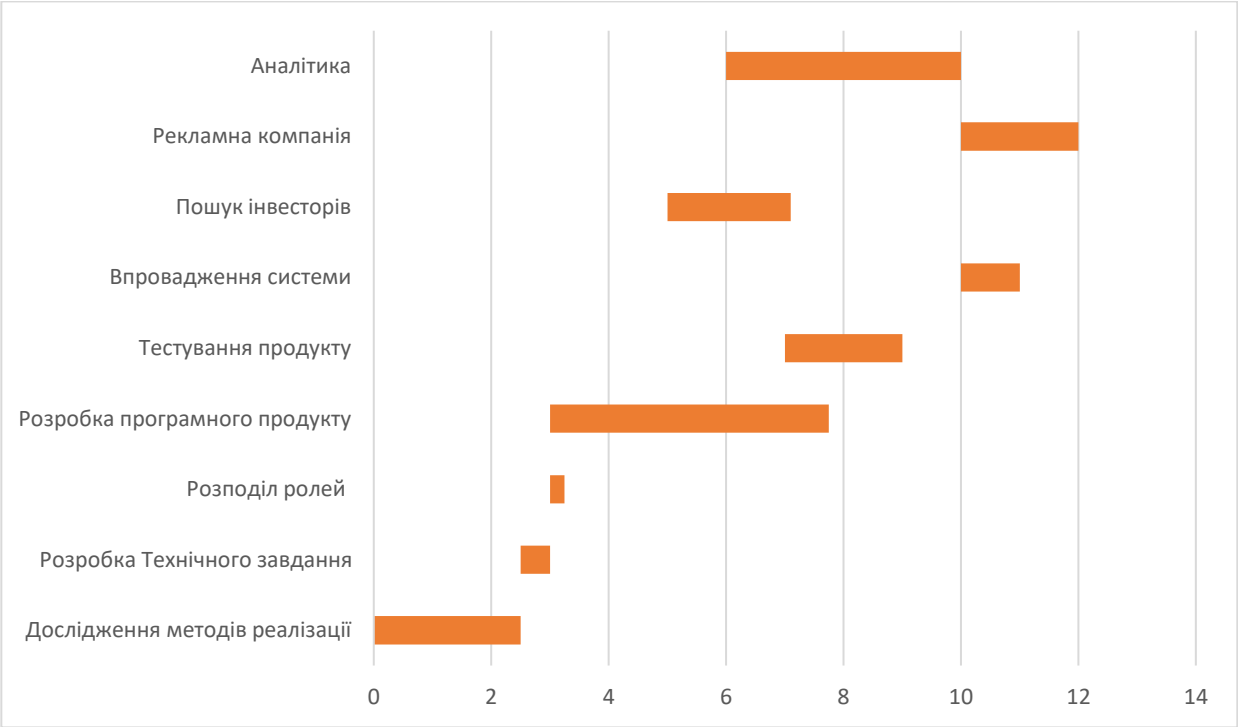
- *Роль:* Розробник**.**
- *Особливості:* Дисциплінований, надійний, практичний. Перетворює ідеї в практичні дії. Може бути менш гнучким, але реагує оперативно

Таблиця 4. ЗПоставлені завдання та час на їх виконання

	Завдання	**Час (місяці)**	**В тижнях**
1	Дослідження методів реалізації	2,5	6 тижнів
2	Розробка Технічного завдання	0,5	2 тижня
3	Розподіл ролей	0,25	1 тиждень
4	Розробка програмного продукту	4.75	19 тижнів
5	Тестування продукту	2	8 тижнів
6	Впровадження системи	1	4 тижні
7	Пошук інвесторів	2	8 тижнів
8	Рекламна компанія	2	8 тижнів
9	Аналітика	4	16 тижнів

Таблиця 4. 4. Граф розподілення часу та завантаженості	**Розрахунок завантаженості учасників команди**
	**Завантаженість = Вхід / Вихід** IT-спеціаліст = 9/ 10 = **0.9** Маркетолог & Проектний менеджер= 7 / 8= **0.875** Розробник = 8 / 6= **1.33**

Графік розподілення часу



IT-спеціаліст = (2.5 + 0.5 + 0.25 + 4.75 + 2 + 1 + 4) / 12 = **1.25** Маркетолог & Проектний менеджер = (0.25 + 2 + 2 + 2 + 4) / 12 = **0.85** Розробник = (0.25 + 4.75 + 2 + 1 + 4) / 12 = **1**	IT-спеціаліст = 0.9 / 1.25 = **0.72** Маркетолог & Проектний менеджер = 0.875 / 0.85 = **1.02** Розробник = 1.33 / 1 = **1.33**

Таблиця 4. 5. Визначення важливості факторів щодо їх вкладу у створення та реалізацію інноваційного проекту

Фактор	**Вага (важливість)**
Ідея	8
Підготовка бізнес плану	7
Компетентність	9
Залученість і ризики	4
Обов'язки	7

Таблиця 4. 6 Оцінювання особистого внеску кожного партнера у створення та реалізацію інноваційного проекту

Фактор	**Вага**	**ІТ-спеціаліст**	**Маркетолог & ПМ**	**Розробник**
Ідея	8	10	3	5
Підготовка бізнес плану	7	3	10	2
Компетентність	9	8	4	6
Залученість і ризики	4	4	5	3
Обов'язки	7	9	7	5

Таблиця 4. 7. Визначення дольової участі у інноваційному проекті кожного учасника

Фактор	**ІТ-спеціаліст**	**Маркетолог & ПМ**	**Розробник**	**Сума**
Ідея	80	24	40	
Підготовка бізнес плану	21	70	14	
Компетентність	72	36	54	
Залученість і ризики	16	20	12	
Обов'язки	63	49	35	

Разом	252	199	155	606
Процент	42%	33%	26%	100%

4.3 Морфологічна карта

Морфологічна карта для проекту AdaGiO охоплює декілька ключових параметрів. Першим з них є модель взаємодії, яка може бути реалізована у вигляді децентралізованої системи, централізованої координації або комбінованого підходу. Цей параметр є основою для визначення архітектури проекту і впливає на ефективність комунікації між агентами та масштабованість системи. Другим важливим параметром є тип агентів. У проекті можуть використовуватися людські моделі, роботизовані агенти або їхні гібридні варіанти. Кожен з цих підходів має свої переваги: людські моделі краще відображають поведінку людей у реальних ситуаціях, а роботизовані агенти можуть бути швидшими й ефективнішими в розрахунках. Ще одним ключовим параметром є середовище моделювання. Для реалізації проекту може бути використано готові рушії, такі як Unity чи Unreal Engine, або ж створено власну платформу. Вибір середовища моделювання впливає на продуктивність, гнучкість і потенціал масштабування системи. Аналіз усіх можливих варіантів і їхніх комбінацій у рамках морфологічної карти дозволяє створити більш повну картину проекту. Це сприяє ухваленню оптимальних технічних і стратегічних рішень. Крім того, морфологічна карта допомагає виявити «білі плями» в системі, які потребують додаткового дослідження, і пропонує потенційні напрямки розвитку проекту.

Таким чином, використання морфологічної карти забезпечує систематичний і всеосяжний підхід до розробки стартапу AdaGiO, дозволяючи знаходити інноваційні рішення і робити проект більш гнучким та ефективним у вирішенні поставлених завдань.

Морфологічна карта

Таблиця 4. 8 Морфологічна карта

Основні параметри	Проміжні рішення				
	1-ше	2-ше	3-ше	4-ше	5-ше
Модель взаємодії	Централізована	Децентралізована	Гібридна	Локальні вузли	Автономні агенти
Тип агентів	Віртуальні персонажі	Модель поведінки людей	Гібридні агенти		Агенти-координатори
Середовище моделювання	Unity	Unreal Engine		AnyLogic	Власне середовище
Динамічні сценарії	Використання нейронної мережі для аналізу стану та визначення інтенсивності сценаріїв	Використання нейронної мережі для визначення випадкових сценаріїв взаємодії	Використання нейронної мережі для створення випадкових сценаріїв взаємодії	Створення динамічних сценарії на основі стану а також аналізу останніх дій	
Тип даних	Реальні дані	Синтетичні дані	Комбіновані	GIS-дані	Статистичні дані
Метод взаємодії	API	Інтерфейси обміну даними	Пряма інтеграція	Модульні компоненти	Метод взаємодії

- Модель взаємодії: Визначає структуру системи, від централізованого управління до повністю автономних агентів.
- Тип агентів: Обирається залежно від мети моделювання: симуляція людей, автоматизовані системи або їхня комбінація.

- Середовище моделювання: Вибір залежить від специфіки проекту: готові рушії забезпечують швидший старт, а власне середовище — більшу гнучкість.
- Тип даних: Використання реальних чи синтетичних даних впливає на точність і адаптивність моделі.
- Метод взаємодії: Спосіб, яким компоненти системи обмінюються інформацією.
- Модель обробки: Обирається залежно від масштабів системи й обчислювальних потреб.

4.4 Розроблення ринкової стратегії інноваційного проекту

Таблиця 4. 9. Вибір цільових груп потенційних споживачів

№ п/п	*Опис профілю цільової групи потенційних клієнтів*	*Готовність споживачів сприйняти продукт*	*Орієнтовний попит в межах цільової групи (сегменту)*	*Інтенсивність конкуренції в сегменті*	*Простота входу у сегмент*
1	Самостійні дослідники (команди із 1-3 людей)	Висока, тому що вони не мають великого людського ресурсу	80% - високий	Низький рівень конкуренції, немає конкурентних продуктів	Легко, тому що відсутні аналоги
2	Маленькі дослідницькі команди	Висока, цей набір інструментів дозволить зменшити навантаження на команду розробника та	90% - високий	Низький рівень конкуренції, немає конкурентних продуктів	Легко, оскільки відсутні аналоги

		прискорить процес розробки			
3	Дослідницькі інститути	Середня, вони мають достатньо ресурсів, та людей.	55% - середній	Низький рівень конкуренції, немає конкурентних продуктів	Важко, тому що вони можуть вирішити проблему завдяки більшій кількості ресурсів
4	Корпоративні клієнти та великі компанії	Низька, мають достатню кількість людських ресурсів. Мають достатню кількість матеріальних ресурсів для розробки аналогічного набору інструментів	40% - низький	Середня конкуренція, мають змогу розробити власний набір інструментів	Важко не зацікавленні в сторонніх рішеннях, мають можливість розробити власний набір інструментів
Які цільові групи обрано: Самостійні дослідники, Маленькі дослідницькі команди. А також Корпоративні клієнти та великі компанії які погодяться співпрацювати					

Таблиця 4. 10. Визначення базової стратегії конкурентної поведінки

№ п/п	*Чи є проект «першопрохідцем» на ринку?*	*Чи буде компанія шукати нових споживачів, або забирати існуючих у конкурентів?*	*Чи буде компанія копіювати основні характеристики товару конкурента, і які?*	*Стратегія конкурентної поведінки**
1	Так, це нова технологія	Компанія буде формувати свою групу споживачів, з перевагою виходу на нових споживачів	ні	Стратегія виклику лідера

Таблиця 4. 11. Визначення стратегії позиціонування

№ п/п	*Вимоги до товару цільової аудиторії*	*Базова стратегія розвитку*	*Ключові конкурентоспроможні позиції власного інноваційного проекту*	*Вибір асоціацій, які мають сформувати комплексну позицію власного проекту (три ключових)*
1	Доступність по вартості	Стратегія концентрованого зростання	Індивідуальні цінові плани в залежності від розміру студії	Нейронні мережі, адаптивність, зручність
2	Легкість використання		Зручність інтерфейсу, документація	

4.5. Розроблення маркетингової програми інноваційного проекту

Таблиця 4. 12. Опис трьох рівнів моделі товару

<table>
<tr><td>Рівні товару</td><td colspan="3">Сутність та складові</td></tr>
<tr><td>I. Товар за задумом</td><td colspan="3">Проект сприятиме глобальному розвитку технологій симуляції та штучного інтелекту, одночасно забезпечуючи локальні інструменти для вирішення практичних задач.</td></tr>
<tr><td rowspan="8">II. Товар у реальному виконанні</td><td>Властивості/характеристики</td><td>М/Нм</td><td>Вр/Тх /Тл/Е/Ор</td></tr>
<tr><td>Модель взаємодії</td><td>Нм</td><td>Тл</td></tr>
<tr><td>Навчання агентів</td><td>Нм</td><td>Тл</td></tr>
<tr><td>Середовище моделювання</td><td>Нм</td><td>Тл/Е</td></tr>
<tr><td>Джерела даних</td><td>М</td><td>Тл/Е</td></tr>
<tr><td>Сценарії застосування</td><td>Нм</td><td>Тл</td></tr>
<tr><td colspan="3"></td></tr>
<tr><td colspan="3"></td></tr>
<tr><td colspan="3">Марка: Advanced Agent-based Generative Intelligence for Operations</td></tr>
<tr><td rowspan="2">III. Товар із підкріпленням</td><td colspan="3">До продажу: Реклама, Пробні версії з обмеженим функціоналом</td></tr>
<tr><td colspan="3">Після продажу: Оновлення</td></tr>
<tr><td colspan="4">За рахунок чого потенційний товар буде захищено від копіювання: необхідність реєстрації та оновлення ліцензії</td></tr>
</table>

Таблиця 4. 13. Формування системи збуту

№ п/п	*Специфіка закупівельної поведінки цільових клієнтів*	*Функції збуту, які має виконувати постачальник товару*	*Глибина каналу збуту*	*Оптимальна система збуту*
	Клієнти купують через сайт компанії	Дистрибуція ліцензій Зворотній зв'язок	Прямий канал збуту	Сайт з можливістю обрати тип ліцензії

4.6 Аналіз ринкових можливостей запуску інноваційного проекту

Таблиця 4. 14. Попередня характеристика потенційного ринку інноваційного проекту

№ п/п	*Показники стану ринку (найменування)*	*Характеристика*
1	Кількість головних гравців, од	10-40 Nvidia, OpenAI
2	Загальний обсяг продаж (реалізації послуг), грн/ум.од	1,471,250
3	Динаміка ринку (якісна оцінка)	Зростає
4	Наявність обмежень для входу (вказати характер обмежень)	немає
5	Специфічні вимоги до стандартизації та сертифікації	ДСТУ ISO/IEC 16350:2016 Інформаційні технології. Розроблення інформаційних систем та програмного забезпечення. Керування програмами (ISO/IEC 16350:2015, IDT)

		ДСТУ ISO/IEC/IEEE 16326:2015 Розроблення систем та програмного забезпечення. Процеси життєвого циклу. Керування проектами (ISO/IEC/IEEE 16326:2009, IDT)
6	Середня норма рентабельності в галузі (або по ринку), %	113

Таблиця 4. 15. Фактори загроз

№ п/п	*Фактор*	*Зміст загрози*	*Можлива реакція компанії*
1	Зростаюча конкуренція	Виникнення нових гравців на ринку, що може призвести до зменшення ринкової частки.	Запуск нових маркетингових кампаній, розширення лінійки продуктів, партнерства та стратегічні альянси.
2	Технологічні ризики	Можливість застаріння технологій або технічних проблем, які можуть вплинути на якість продукту.	Інвестування в дослідження та розвиток, підтримка технічних інновацій, створення резервних планів.
3	Зміни в законодавстві	Зміни в законодавстві, які можуть вплинути на діяльність компанії або її продукти.	Адаптація до нових правил та вимог, співпраця з правозахисними органами, лобіювання інтересів компанії.

Таблиця 4. 16. Ступеневий аналіз конкуренції на ринку

Особливості конкурентного середовища	*В чому проявляється дана характеристика*	*Вплив на діяльність підприємства (можливі дії компанії, щоб бути конкурентоспроможною)*
1. Вказати тип конкуренції чиста	На ринку існує багато конкурентів, і кожен з них має певний вплив, але жоден не володіє значущою часткою ринку.	Компанія повинна акцентувати увагу на унікальних перевагах свого продукту, щоб привертати увагу та забезпечувати конкурентоспроможність.
2. За рівнем конкурентної боротьби інтернаціональний	Конкуренція поширюється на міжнародний рівень, що вказує на наявність конкурентів та можливих клієнтів за межами національного ринку.	Компанія повинна розширювати міжнародні мережі продажу, адаптувати продукт до різних культур та ринків, і розвивати міжнародну стратегію маркетингу
3. За галузевою ознакою Внутрішньогалузева	Конкуренція в межах однієї галузі, що вказує на те, що компанія конкурує з іншими гравцями, які працюють в тій же сфері.	Компанія повинна вивчати та реагувати на стратегії конкурентів у власній галузі, виявляти переваги та впроваджувати стратегії диференціації.
4. Конкуренція за видами товарів: Товарно-родова та між бажаннями	Конкуренція заснована на властивостях товарів та задоволенні бажань споживачів.	Компанія повинна акцентувати увагу на визначенні та рекламуванні унікальних характеристик

		свого продукту для привертання споживачів.
5. За характером конкурентних переваг - цінова та нецінова	Конкуренція, де ціни є фактором вибору, але також важливі нецінові аспекти.	Компанія повинна акцентувати увагу на визначенні та рекламуванні унікальних характеристик свого продукту для привертання споживачів..
6. За інтенсивністю не марочна	Конкуренція, де товар чи послуга сама по собі має основну значущість.	Компанія повинна фокусуватися на характеристиках свого продукту та якості обслуговування для відзначення від конкурентів.

Таблиця 4. 17. SWOT- аналіз інноваційного проекту

Сильні сторони: 1. Інноваційність та унікальність продукту. 2. Висока якість розроблюваного продукту. 3. Гнучкість у внесенні змін та адаптації.	Слабкі сторони: 1. 1. Високі витрати на дослідження та розробку. 2. Потреба у підтримці для самостійних розробників. 3. Залежність від сторонніх постачальників 4. Можливі труднощі з сертифікацією та стандартизацією.
Можливості: 1. Зростання інтересу до ігрової індустрії. 2. Розширення міжнародного ринку. 3. Партнерства з великими студіями та видавництвами. 4. Зростання інтересу до ігор з використанням нейронних мереж. 5. Розвиток та впровадження нових технологій у галузі.	Загрози: 1. Зміни у законодавстві, що можуть вплинути на розробку. 2. Конкуренція з боку великих студій та видавництв. 3. Технічні або безпекові проблеми, які можуть виникнути. 4. Зменшення попиту через зміни в споживчому попиті. 5. Вплив економічних криз або фінансових труднощів.

Таблиця 4. 18. Альтернативи ринкового впровадження інноваційного проекту

№ п/п	*Альтернатива (орієнтовний комплекс заходів) ринкової поведінки*	*Ймовірність отримання ресурсів*	*Строки реалізації*
	Розширення маркетингових зусиль для самостійних дослідників.	Висока	5 місяців
	Укладання партнерств з великими гравцями індустрії.	Середня	8 місяців
	Участь у великих галузевих виставках та конференціях.	Середня	3 місяці

4.7 Виробничий план

Таблиця 4. 19. Календарний план-графік реалізації інноваційного проекту

№ з/п	*Етапи реалізації*	*Період реалізації проекту*						
		0-й рік				*1-й рік*	*2-й рік*	*3-й рік*
		1-й кв.	*2-й кв.*	*3-й кв.*	*4-й кв.*			
1.	Проведення НДДКР	+	+	+	-	+	+	+
2.	Розробка проектних матеріалів і ТЕО	+	+	-	-	+	+	+
3.	Робоче проектування і прив'язка проекту	+	+	+	+	+	-	-
4.	Створення компанії	+	-	-	-	-	-	-
5.	Придбання нематеріальних активів, отримання дозвільних документів тощо	-	-	-	-	-	-	-
6.	придбання й оренда земельних ділянок, будівель, приміщень, споруд	-	+	+	+	+	+	+

7.	Придбання обладнання, устаткування та пристроїв	+	-	-	-	-	+	+
8.	Передвиробничі маркетингові дослідження	+	+	+	-	+	+	+
9.	Приймально-здавальні випробування	+	+	+	-	-	+	-
10.	Пусконалагоджувальні роботи	-	-	+	+	-	-	-
11.	Освоєння проектних потужностей	+	+	+	+	-	-	-
13.	Придбання матеріальних ресурсів	+	+	+	-	+	-	+
13.	Запуск виробництва	-	-	-	-	-	-	-
14.	Продаж продукції	-	-	-	+	+	+	+

Даний план-графік реалізації інноваційного проекту є оптимальним для нашої команди, оскільки він враховує конкретні особливості та можливості нашого колективу та спрямований на досягнення успіху проекту. Графік визначає послідовність та призначає конкретні завдання на кожному етапі проекту, що спрощує роботу команди та забезпечує чітку відповідальність. Графік відображає доступні ресурси та розподіляє їх відповідно до потреб проекту, що допомагає ефективно використовувати наявні ресурси. Загалом, цей план-графік відображає оптимальний підхід до реалізації інноваційного проекту для нашої команди та допоможе нам досягти успіху у виконанні поставлених завдань.

4.7.1 Планова потреба у виробничих площах та обладнанні

Таблиця 4. 20. Планова потреба у виробничих площах

№ з/п	*Тип приміщення (будівлі, ділянки, споруди)*	*Кількість одиниць*	*Площа, кв. м*	*Вимоги до приміщення (будівлі, ділянки, споруди)*	*Умови надання*	*Вартість, тис. грн.*
1.	Офіс	1	30	Охорона праці в офісі: нормативна база ДСН 2.3.6.037-99 ДСН 3.3.6.039-99 ДСН 3.3.6.042-99 Вимоги до офісного приміщення та організації робочого місця ДСанПіН 3.3.2.007-98 Вимоги до вентиляції, опалення, кондиціонування, мікроклімату ДБН В.2.5-67:2013 Вимоги до освітлення ДБН В.2.5-28-2006 Вимоги до рівнів шуму та вібрації ДСанПіН 3.3.2.007-98 Допустимі параметри неіонізуючого	оренда	8,4 місяць

				електромагнітного випромінювання ДСанПіН 3.3.2.007-98		
…						
Разом				—	—	100,8

Потрібно лише офісне приміщення.

Вимоги до офісного приміщення та організації робочого місця повинні відповідати нормативам, зазначеним у ДСН 2.3.6.037-99, ДСН 3.3.6.039-99, ДСН 3.3.6.042-99, ДСанПіН 3.3.2.007-98, ДБН В.2.5-67:2013, ДБН В.2.5-28-2006, ДСанПіН 3.3.2.007-98, та іншим відповідним стандартам.

Таблиця 4. 21. Планова вартість нематеріальних активів

№ з/п	*Вид активів*	*Активи, що можуть бути віднесені до даного виду*	*Вартість, тис. грн.*
1.	Права на комерційні позначення	права на торговельні марки (знаки для товарів і послуг), комерційні (фірмові) найменування тощо	6
2.	Права на об’єкти промислової власності	право на винаходи	1,6
3.	Авторське право та суміжні з ним права	На програми відеогри, передачі (програми) тощо	2
4.	Право на оренду	Право на оренду офісного приміщення	10,8
Разом			20,4

Планова вартість нематеріальних активів розподіляється на такі види активів:

Права на комерційні позначення: Вартість - 6 тис. грн. Вони включають права на торговельні марки, знаки для товарів і послуг, комерційні (фірмові)

найменування тощо. Права на об’єкти промислової власності: Вартість - 1,6 тис. грн. Це право на винаходи. Авторське право та суміжні з ним права: Вартість - 2 тис. грн. Це включає розроблені програми відеогри, передачі (програми) тощо. Право на оренду: Вартість - 10,8 тис. грн. Визначення відсотка від орендної плати, які сплачуються відповідно до законодавства. В цілому, на нематеріальні активи планується витратити 20,4 тис. грн., і ці активи будуть використані для покриття потреб в правах на комерційні позначення, правах на об’єкти промислової власності, авторському праві та суміжних правах.

Ми плануємо розвивати інноваційний проект, який передбачає виробництво ліцензій на використання набору інструментів. Наша стратегія передбачає поступове збільшення обсягів виробництва протягом трьох років. У перший рік нашої діяльності ми плануємо продати 300 ліцензій. Це допоможе нам встановити нашу позицію на ринку і побудувати клієнтську базу. У другий рік ми плануємо збільшимо обсяг продаж до 500 ліцензій. У третій рік ми плануємо продати 800 ліцензій. Ми впевнені, що наш інноваційний продукт матиме попит серед клієнтів, і ми готові забезпечити їх потреби.

Таблиця 4. 22. Плановий обсяг виробництва продукції інноваційного проекту

Вид продукції	*Одиниця виміру*	*Обсяги виробництва за період*		
		1-й рік	*2-й рік*	*3-й рік*
Ліцензія на використання набору інструментів	шт.	300	500	800

Висновок до розділу 4

Розробка стартап-проекту є комплексним і багатоступеневим процесом, що вимагає ретельного планування та аналізу. У цьому розділі були визначені основні характеристики майбутнього проекту, його цілі, завдання та очікувані результати. Проект має чітку стратегію, спрямовану на досягнення значних результатів як на локальному, так і на глобальному рівнях, що підтверджується деталізованим описом кожного етапу його реалізації.

Морфологічна карта, розглянута у цьому розділі, дозволяє зібрати всі можливі варіанти рішень для ключових параметрів проекту, що є основою для подальшого вибору оптимальних шляхів розвитку. Параметри, такі як модель взаємодії агентів, тип середовища моделювання, та методи обробки даних, дозволяють нам визначити гнучку стратегію для створення ефективної системи.

Загалом, цей розділ надає чітке уявлення про майбутній стартап, демонструючи не тільки його теоретичну цінність, але й практичний потенціал для досягнення високих результатів у межах визначених завдань. Застосування таких інструментів, як морфологічна карта, дозволяє оптимізувати процеси розробки та прийняття рішень, що є важливим кроком для досягнення успіху в реалізації проекту.

ВИСНОВКИ

Проект, спрямований на розробку та впровадження агент-орієнтованої моделі для моделювання складних систем з використанням великих мовних моделей, продемонстрував важливість інтердисциплінарного підходу та гнучкості в процесі розробки. Вивчені основні аспекти теоретичного та практичного застосування LLM, зокрема їхню здатність до імітації людської поведінки в симуляціях, а також аналіз їхніх обмежень та можливостей.
У процесі роботи над проектом були досягнуті кілька значущих результатів. Зокрема, було розроблено механізм для взаємодії агентів у симуляційному середовищі, що дозволяє ефективно управляти різними аспектами пошукових операцій. Водночас виявлені певні проблеми, зокрема обмежена здатність LLM розуміти контекст ситуацій, що обмежує їхню ефективність у деяких аспектах операційної координації. Це стало одним із ключових викликів для проекту, але водночас ілюструє потребу в удосконаленні моделей для специфічних задач.

У рамках цього проекту вдалося створити систему, яка не лише коректно працює в заданих умовах, а й може бути використана для вивчення емерджентної поведінки в агентних системах. Проект демонструє великий потенціал у плані подальшого розвитку та розширення можливостей, таких як додавання нових типів агентів, удосконалення алгоритмів прийняття рішень і поглиблення інтеракцій між агентами. Таким чином, ми вважаємо, що даний проект не тільки є корисним для досліджень в рамках симуляційних моделей, але й має можливість бути розширеним для використання в інших, більш складних сценаріях, що дозволить глибше досліджувати емерджентні явища та їх застосування у різних галузях.

У контексті проект має високий потенціал для подальших досліджень та реалізації в реальних умовах. Він демонструє не лише практичну цінність для створення систем, що імітують людську поведінку, але й має великий потенціал для удосконалення технологій і методів, що використовуються в симуляціях.

У майбутньому проект може бути значно вдосконалений шляхом тренування більш специфічних моделей, адаптованих до конкретних задач. Це дозволить підвищити точність симуляцій та ефективність управлінських рішень, що, в свою чергу, збільшить застосовуваність таких технологій у реальних операціях та широких галузях, таких як пошуково-рятувальні операції та інші комплексні системи управління.

Цей проект є значним кроком вперед у застосуванні великих мовних моделей та агент-орієнтованих систем для вирішення складних задач, що вимагають інтеграції різноманітних підходів і методів, відкриваючи нові горизонти для подальших досліджень та реалізації.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. A. Giordano, A. De Rango, D. D'Ambrosio, R. Rongo and W. Spataro, "Strategies for Parallel Execution of Cellular Automata in Distributed Memory Architectures," 2019 27th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP), Pavia, Italy, 2019, pp. 406-413, doi: 10.1109/EMPDP.2019.8671639.
2. Fish, J., & Belytschko, T. (2007). A First Course in Finite Elements. John Wiley & Sons.
3. Zienkiewicz, O. C., Taylor, R. L., & Zhu, J. Z. (2013). The Finite Element Method: Its Basis and Fundamentals (7th ed.). Butterworth-Heinemann.
4. Bonabeau, E. (2002). Agent-based modeling: Methods and techniques for simulating human systems. Proceedings of the National Academy of Sciences, 99(Suppl 3), 7280-7287.
5. Macal, C. M., & North, M. J. (2010). Tutorial on agent-based modeling and simulation. Journal of Simulation, 4(3), 151-162.
6. Railsback, S. F., & Grimm, V. (2011). Agent-Based and Individual-Based Modeling: A Practical Introduction. Princeton University Press.
7. Metropolis, N., & Ulam, S. (1949). The Monte Carlo method. Journal of the American Statistical Association, 44(247), 335-341.
8. Rubinstein, R. Y., & Kroese, D. P. (2017). Simulation and the Monte Carlo Method (3rd ed.). John Wiley & Sons.
9. Wolfram, S. (1986). Theory and Applications of Cellular Automata. World Scientific.

10. Ilachinski, A. (2001). Cellular Automata: A Discrete Universe. World Scientific.
11. Chopard, B., & Droz, M. (1998). Cellular Automata Modeling of Physical Systems. Cambridge University Press.
12. Law, A. M., & Kelton, W. D. (2000). Simulation Modeling and Analysis (3rd ed.). McGraw-Hill.
13. Banks, J., Carson, J. S., Nelson, B. L., & Nicol, D. M. (2010). Discrete-Event System Simulation (5th ed.). Prentice Hall.
14. Schriber, T. J., & Brunner, D. T. (2014). Inside discrete-event simulation software: How it works and why it matters. Proceedings of the 2014 Winter Simulation Conference.
15. Shanghai AI Laboratory. InternLM2.5-20B-Chat GGUF Model [Електронний ресурс] / Shanghai AI Laboratory – Режим доступу до ресурсу: https://huggingface.co/internlm/internlm2_5-20b-chat-gguf.
16. Self-Consistency Improves Chain of Thought Reasoning in Language Models Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, Denny Zhou
17. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks Patrick Lewis, Ethan Perez , Aleksandra Piktus, Fabio Petroni , Vladimir Karpukhin , Naman Goyal† , Heinrich Küttler , Mike Lewis , Wen-tau Yih , Tim Rocktäschel Sebastian Riedel, Douwe Kiela

ДОДАТОК 1

Частина програмного коду

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

public class EnviromentManager : MonoBehaviour
{
    [SerializeField]
    private Mesh forestMesh;  // Reference to your forest mesh
    [SerializeField]
    private Terrain terrain;  // Reference to the terrain

    [SerializeField]
    private TerrainLayer insideLayer;  // Terrain layer for inside color
    [SerializeField]
    private TerrainLayer outsideLayer; // Terrain layer for outside color

    // Start is called before the first frame update
    void Start()
    {
        RecolorTerrain(forestMesh);
    }

    void RecolorTerrain(Mesh mesh)
    {
        // Get the bounds of the mesh
        Bounds bounds = mesh.bounds;
        Vector3 minBounds = bounds.min;
        Vector3 maxBounds = bounds.max;

        // Get the terrain data
        TerrainData terrainData = terrain.terrainData;

        // Get the position of the terrain
        Vector3 terrainPosition = terrain.transform.position;

        // Get the resolution of the terrain's alpha map
        int alphaMapWidth = terrainData.alphamapWidth;
        int alphaMapHeight = terrainData.alphamapHeight;

        // Create a new alpha map for the terrain
        float[,,] alphaMaps = new float[alphaMapWidth, alphaMapHeight,
terrainData.terrainLayers.Length];

        // Set the alpha values for each pixel in the alpha map
        for (int y = 0; y < alphaMapHeight; y++)
        {
            for (int x = 0; x < alphaMapWidth; x++)
            {
                // Calculate the world position of the terrain pixel
```

```
                float worldX = terrainPosition.x + (x / (float)alphaMapWidth) *
terrainData.size.x;
                float worldZ = terrainPosition.z + (y / (float)alphaMapHeight) *
terrainData.size.z;

                // Check if this pixel is inside the mesh bounds
                bool isInside = worldX >= minBounds.x && worldX <= maxBounds.x &&
                                worldZ >= minBounds.z && worldZ <= maxBounds.z;

                // Determine which layer to apply
                int layerIndex = isInside ? GetLayerIndex(insideLayer) :
GetLayerIndex(outsideLayer);

                // Set alpha for the corresponding layer
                alphaMaps[x, y, layerIndex] = 1f; // Full strength for the active
layer
            }
        }

        // Assign the modified alpha maps back to the terrain
        terrainData.SetAlphamaps(0, 0, alphaMaps);
    }

    // Helper function to get the index of a TerrainLayer
    private int GetLayerIndex(TerrainLayer layer)
    {
        for (int i = 0; i < terrain.terrainData.terrainLayers.Length; i++)
        {
            if (terrain.terrainData.terrainLayers[i] == layer)
            {
                return i;
            }
        }
        return -1; // Layer not found
    }
}
```

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

[CreateAssetMenu(fileName = "EnviromentSpeedSO", menuName =
"ABM/Enviroment/EnviromentSpeedSO", order = 1)]
public class EnviromentSpeedSO : ScriptableObject
{
    [SerializeField] public EnviromentEnum EnviromentType;
    [SerializeField] public float EnviromentSpeed;

    public EnviromentEnum GetEnviromentType() => EnviromentType;
    public float GetEnviromentSpeed() => EnviromentSpeed;
}
```

```
using UnityEngine;
using UnityEngine.Networking;
using System.Collections;
using System.Text;
using System.Collections.Generic;
using ABM.Utils;
using System.Diagnostics.Tracing;
using System.Runtime.InteropServices.WindowsRuntime;
using static ABM.Utils.Utilities;
using System;

public class SimulationToLLM : MonoBehaviour
{
    [SerializeField]
    private SimulationController simulationController; // Reference to gather
scenario data

    private const string API_URL = "http://localhost:1234/v1/chat/completions";
// LM Studio OST endpoint
                                                    //private const string
API_KEY = "your_api_key_here"; // Replace with your LM Studio API key



    private void Awake()
    {
        simulationController = GetComponent<SimulationController>();
```

```
    }

    public void RequestCommandFromLLM(SearchOfficerAgent
searchOfficerAgent, string rejectReason = "")
    {
        // Step 1: Prepare the scenario data for the LLM prompt
        string scenarioData = simulationController.CollectScenarioInfo();

        // Step 2: Create a complete system + user prompt
        string systemPrompt = GenerateSystemPrompt();
        string userPrompt = GenerateUserPrompt(scenarioData,
searchOfficerAgent);
        userPrompt += rejectReason;

        // Step 3: Send the prompt to the LLM and handle the response
        StartCoroutine(SendScenarioToLLM(systemPrompt, userPrompt, response
=>
        {
            HandleResponse(response, searchOfficerAgent);
        }));
    }

    private string GenerateSystemPrompt()
    {
        return @"
```

You are an advanced mission coordinator for a team of autonomous search units operating in a 3D virtual environment. Your task is to guide search units to locate a target unit based on the following information:
It's recomended to move search officer closer to target position.
1. Each search unit knows its initial position in the environment.
2. The approximate starting position of the target unit is provided.
3. The environment may include obstacles that search units cannot pass through.
4. Search units can move to specified coordinates or send it's subordinates to perform localized scans for the target.
5. Search units can move by youre command but if you move search officer all his subordinates must follow him ignoring all previous orders
6. Target can move, it's up to you to find out where it's moving based on clues.
7. You can't repeat commands!

Your response must include clear and actionable commands for each search unit. Use the following guidelines:
- Specify movement commands in the format: `Move [UnitName] to ([X], [Y], [Z])`.
- Specify scan commands in the format: `[UnitName] scans the area around ([X], [Y], [Z]).`
- Avoid unnecessary details or assumptions not supported by the input.
- Prioritize efficiency: Minimize redundant actions and suggest commands that improve the likelihood of finding the target.

- Maximum move distance of any officer is 1000 meters from their current position
- Maximum move distance of any regular agent is 200 meters from their officer position
- You can't repeat commands!

Here is an example response:
Command: Move SearchOfficer[1] to (5, 0, 10). Command: Send Units under SearchOfficer[1] to (8.0, 0.0, 12.0). Command: SearchOfficer[2] scans the area around (6, 0, 11)

You must give only 1 command in response.

You must give only commands until target is found

If the target is found, include a summary of the search status at the end of your response.

Youre responses must contain only commands until target is found
You can't repeat commands!

";
    }

    private string GenerateUserPrompt(string scenarioData, SearchOfficerAgent searchOfficerAgent)

```
    {
        return $@"
Scenario:
{scenarioData}

The agent you need to assist is SearchOfficer[{searchOfficerAgent.ID}]. Provide commands for their next actions. They can move only 1000 meters from their current position {searchOfficerAgent.transform.position}
You can't repeat commands!
";
    }

    private string GenerateReportPrompt(string scenarioData)
    {
        return $@"
Scenario:
{scenarioData}

Target was found. Provide summary of search operation
";
    }

    private IEnumerator SendScenarioToLLM(string systemPrompt, string userPrompt, System.Action<string> onResponse)
    {
        // Manually construct the JSON payload
```

```
        string jsonData = $@"
    {{
        ""model"": ""internlm2_5-20b-chat"",
        ""messages"": [
            {{ ""role"": ""system"", ""content"":
""{EscapeJson(systemPrompt)}"" }},
            {{ ""role"": ""user"", ""content"": ""{EscapeJson(userPrompt)}"" }}
        ],
        ""max_tokens"": 50,
        ""temperature"": 0.7
    }}";

        Debug.Log($"Sending JSON Payload: {jsonData}");

        using (UnityWebRequest request = new UnityWebRequest(API_URL,
"POST"))
        {
            request.SetRequestHeader("Content-Type", "application/json");
            //request.SetRequestHeader("Authorization", $"Bearer {API_KEY}");
            request.uploadHandler = new
UploadHandlerRaw(Encoding.UTF8.GetBytes(jsonData));
            request.downloadHandler = new DownloadHandlerBuffer();

            yield return request.SendWebRequest();

            if (request.result == UnityWebRequest.Result.Success)
```

```
        {
            string responseText = request.downloadHandler.text;
            Debug.Log($"Response: {responseText}");
            onResponse?.Invoke(responseText);
        }
        else
        {
            Debug.LogError($"LLM Request Failed: {request.error}\nResponse:
{request.downloadHandler.text}");
            onResponse?.Invoke(null); // Handle failure gracefully
        }
    }
}

public string ReportEnd(Action<string> ReportHandle)
{
    simulationController.UnPauseSimulation();
    // Step 1: Prepare the scenario data for the LLM prompt
    string scenarioData = simulationController.CollectScenarioInfo();

    // Step 2: Create a complete system + user prompt
    string systemPrompt = GenerateSystemPrompt();
    string userPrompt = GenerateReportPrompt(scenarioData);
    simulationController.PauseSimulationAtFrame();
    // Step 3: Send the prompt to the LLM and handle the response
    StartCoroutine(ReportEndToLLM(systemPrompt, userPrompt, response =>
```

```
        {
            ReportHandle(response);
        }));

        return "";
    }

    public IEnumerator ReportEndToLLM(string systemPrompt, string
userPrompt, System.Action<string> onResponse)
    {
        // Manually construct the JSON payload
        string jsonData = $@"
    {{
        ""model"": ""internlm2_5-20b-chat"",
        ""messages"": [
            {{ ""role"": ""system"", ""content"":
""{EscapeJson(systemPrompt)}"" }},
            {{ ""role"": ""user"", ""content"": ""{EscapeJson(userPrompt)}"" }}
        ],
        ""max_tokens"": 10000,
        ""temperature"": 0.7
    }}";

        Debug.Log($"Sending end JSON Payload: {jsonData}");
```

```
    using (UnityWebRequest request = new UnityWebRequest(API_URL,
"POST"))
    {
        request.SetRequestHeader("Content-Type", "application/json");
        //request.SetRequestHeader("Authorization", $"Bearer {API_KEY}");
        request.uploadHandler = new
UploadHandlerRaw(Encoding.UTF8.GetBytes(jsonData));
        request.downloadHandler = new DownloadHandlerBuffer();

        yield return request.SendWebRequest();

        if (request.result == UnityWebRequest.Result.Success)
        {
            string responseText = request.downloadHandler.text;
            Debug.Log($"Response: {responseText}");
            onResponse?.Invoke(responseText);
        }
        else
        {
            Debug.LogError($"LLM Request Failed: {request.error}\nResponse:
{request.downloadHandler.text}");
            onResponse?.Invoke(null); // Handle failure gracefully
        }
    }
  }
```

```
    private string EscapeJson(string text)
    {
        return text.Replace("\\", "\\\\").Replace("\"", "\\\"").Replace("\n",
"\\n").Replace("\r", "\\r");
    }



    private void HandleResponse(string response, SearchOfficerAgent
searchOfficerAgent)
    {
        // Parse the response and extract commands
        var jsonResponse = JsonUtility.FromJson<OSTResponse>(response);
        string commands = jsonResponse.choices[0].message.content;

        string[] commandLines = commands.Split('\n');
        ABM.Utils.Command agentCommand = new ABM.Utils.Command();

        var command = commandLines[0];



            if (command.Contains("Command:") && command.Contains("Move"))
            {
                agentCommand.CommandPrompt = command;
```

```
            ExecuteMoveCommand(agentCommand, searchOfficerAgent);
        }
        else if (command.Contains("Command:") &&
command.Contains("Send"))
        {
            agentCommand.CommandPrompt = command;

            ExecuteScanCommand(agentCommand, searchOfficerAgent);
        }
        else
        {
            Debug.Log($"Error: {command}");
            RequestCommandFromLLM(searchOfficerAgent, $"{command} This
command incorectly formated. \r\nHere is an example response:\r\nCommand:
Move SearchOfficer[1] to (5, 0, 10). Command: Send Units under
SearchOfficer[1] to (8, 0, 12). Command: SearchOfficer[2] scans the area around
(6, 0, 11) \r\n\r\nYou must give only 1 command in response.");
        }

    }
```

```
    private void ExecuteMoveCommand(Command command,
SearchOfficerAgent searchOfficerAgent)
    {
        if (ParsePosition(command.CommandPrompt, out Vector3 targetPosition))
        {
            command.Action = new AgentAction<Vector3>((targetPosition) => { });

            simulationController.RegisterRecievedCommand(command);
            searchOfficerAgent.SetMoveDirection(targetPosition, command);
            Debug.Log($"{searchOfficerAgent.ID} moving to {targetPosition}");
        }
        else
        {
            searchOfficerAgent.RejectCommand();
        }

    }

    private void ExecuteScanCommand(Command command,
SearchOfficerAgent searchOfficerAgent)
    {
        if (ParsePosition(command.CommandPrompt, out Vector3 scanLocation))
        {
            command.Action = new AgentAction<Vector3>((scanLocation) => { });

            simulationController.RegisterRecievedCommand(command);
```

```
            searchOfficerAgent.Scan(scanLocation, command);
            Debug.Log($"{searchOfficerAgent.ID} scanning the area around
{scanLocation}");
        }
        else
        {
            searchOfficerAgent.RejectCommand();
        }

    }

    private bool TryParsePosition(string command, out Vector3 result)
    {
        result = Vector3.zero;
        int start = command.IndexOf("(") + 1;
        int end = command.IndexOf(")");
        string[] coords = command.Substring(start, end - start).Split(',');

        if (string.IsNullOrEmpty(coords[0]))
            return false;
        if (string.IsNullOrEmpty(coords[1]))
            return false;
        if (string.IsNullOrEmpty(coords[2]))
            return false;
```

```
        if (float.TryParse(coords[0], out float x) && float.TryParse(coords[1], out
float y) && float.TryParse(coords[2], out float z))
        {
            result = new Vector3(x, y, z);
            return true;
        }



        else
        {
            result = Vector3.zero;
            simulationController.RegisterREjectedCommand(command);
            return false;
        }
    }

    private bool ParsePosition(string command, out Vector3 result)
    {
        int start = command.IndexOf("(") + 1;
        int end = command.IndexOf(")");
        string[] coords = command.Substring(start, end - start).Split(',');

        if (float.TryParse(coords[0], out float x) && float.TryParse(coords[1], out
float y) && float.TryParse(coords[2], out float z))
        {
            result = new Vector3(x, y, z);
```

```
            return true;
        }



        else
        {
            result = Vector3.zero;
            simulationController.RegisterREjectedCommand(command);
            return false;
        }




    }
}

// Helper class to parse the API response
[System.Serializable]
public class OSTResponse
{
    public Choice[] choices;

    [System.Serializable]
    public class Choice
    {
```

```
        public Message message;


        [System.Serializable]
        public class Message
        {
            public string role;
            public string content;
        }
    }
}
```

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.Serialization;
using ABM.Core;
using ABM.Base;
using System.Diagnostics;
using UnityEngine.AI;
using ABM.Enviroment;
using System.Linq;
using System;
using Random = UnityEngine.Random;
using ABM.Utils;
using Debug = UnityEngine.Debug;
using System.IO;
```

```
using System.Text.Json;
using static UnityEngine.UIElements.UxmlAttributeDescription;
using Unity.VisualScripting;

public class SimulationController : AbstractController
{
    [Header("Simulation Parameters")]
    public GameObject simulationTerrainGameObject;

    public GameObject searchAgentPrefab;
    public GameObject officerPrefab;
    public GameObject targetPrefab;

    public GameObject targetZone;
    public GameObject searchPartyZone;

    public int numAgentsPerOfficer;
    public int numOfficerAgents;
    public int numTargetAgents;

    public SimulationToLLM simulationToLLM;

    public List<GameObject> agentsInstances;
    public List<GameObject> officerAgentsInstances;
    public GameObject targetInstance;
```

```
    [SerializeField]
    private List<GameObject> cardinals;

    [SerializeField]
    private List<EnviromentSpeedSO> enviromentSpeed;

    [Header("Agent Parameters")]
    public float distToTarget;

    public LayerMask agentLayer;

    public Vector3 targetPotentialLocation;

    [Header("Debug")]
    [SerializeField, FormerlySerializedAs("Debug")] private bool isDebug;

    private List<Command> recievedCommands = new List<Command>();

    private List<Command> executingCommands = new List<Command>();

    private List<Command> executedCommands = new List<Command>();

    private List<string> rejectedCommadns = new List<string>();

    private Dictionary<SearchAgent, List<GameObject>> agentClues = new
Dictionary<SearchAgent, List<GameObject>>();
```

```
private Guid SimulationGUID;

[SerializeField]
private Camera simulationCamera;

[SerializeField]
private Camera secondaryCamera;

[SerializeField]
private DateTime currentDateTime;

[SerializeField]
private List<GameObject> clues;

[SerializeField]
private List<Vector3> cluePositions;

[SerializeField]
private string directoryPath;

public override void Init()
{
    base.Init();
    SimulationGUID = Guid.NewGuid();
    simulationToLLM = GetComponent<SimulationToLLM>();
```

```
        targetPotentialLocation = targetZone.transform.position;
        InitializeTarget();
        InitializeSearchParty();

        currentDateTime = DateTime.UtcNow;
        CreateDirectory();
    }

    private void InitializeSearchParty()
    {
        var searchPartyZoneBounds =
searchPartyZone.GetComponent<Collider>().bounds;
        GetRandomPositionInBounds(searchPartyZoneBounds);
        for (int i = 1; i <= numOfficerAgents; i++)
        {
            var officerInstance = Instantiate(officerPrefab,
GetRandomPositionInBounds(searchPartyZoneBounds), Quaternion.identity);

            officerAgentsInstances.Add(officerInstance);

            GameObject sphereCollider =
GameObject.CreatePrimitive(PrimitiveType.Sphere);
            sphereCollider.transform.position = officerInstance.transform.position;

            //sphereCollider.transform.SetParent(officerInstance.transform);
```

```
            sphereCollider.transform.localScale = new Vector3(30f, 30f, 30f);

            Physics.SyncTransforms();

            List<GameObject> agentsListToOfficer = new List<GameObject>();
            for (int j = 1; j <= numAgentsPerOfficer; j++)
            {
                var searchAgentInstance = Instantiate(searchAgentPrefab,
GetRandomPositionInBounds(sphereCollider.GetComponent<Collider>().bound
s), Quaternion.identity);

                searchAgentInstance.GetComponent<SearchAgent>().Init((i * 100 +
j), targetZone, officerInstance, this);
                agentsInstances.Add(searchAgentInstance);
                agentsListToOfficer.Add(searchAgentInstance);
            }

            officerInstance.GetComponent<SearchOfficerAgent>().Init(i, targetZone,
agentsListToOfficer, this);

            Destroy(sphereCollider);
        }
    }

    private void InitializeTarget()
    {
```

```
        // Spawn target in random position inside target zone, higher than terrain
        var targetZoneBounds = targetZone.GetComponent<Collider>().bounds;

        targetInstance = Instantiate(targetPrefab,
GetRandomPositionInBounds(targetZoneBounds), Quaternion.identity);

        Debug.Log(targetInstance);
        Debug.Log(this);
        Debug.Log(cardinals[Random.Range(0, cardinals.Count)]);
        //Vector3 targetPosition = targetInstance.transform.position;
        //targetPosition.y = simulationTerrainGameObject

//    .GetComponent<Terrain>().SampleHeight(targetInstance.transform.position)
;
        //Vector3 test = targetPosition;
        //Trace.Write(test);
        //targetPosition.y = targetPosition.y +
(targetInstance.GetComponent<Collider>().bounds.extents.y * 1.5f);
        //targetInstance.transform.position = targetPosition;
        Debug.Log(targetInstance.GetComponent<TargetAgent>());
        var targetAgent = targetInstance.GetComponent<TargetAgent>();

        targetAgent.PostInit(cardinals[Random.Range(0, cardinals.Count)], this);
    }

    private Vector3 GetRandomPositionInBounds(Bounds targetZoneBounds)
```

```
   {
      float spawnPointX = Random.Range(targetZoneBounds.min.x,
targetZoneBounds.max.x);
      float spawnPointY = Random.Range(targetZoneBounds.min.y,
targetZoneBounds.max.y);
      float spawnPointZ = Random.Range(targetZoneBounds.min.z,
targetZoneBounds.max.z);

      Vector3 targetZoneCenter = targetZone.transform.position;

      //var terrainHeight =
simulationTerrainGameObject.GetComponent<Terrain>().SampleHeight();

      Vector3 randomPosition = new Vector3(spawnPointX, spawnPointY,
spawnPointZ);
      Vector3 sourcePostion = randomPosition;
      NavMeshHit closestHit;
      if (NavMesh.SamplePosition(sourcePostion, out closestHit, 500, 1))
      {
         return closestHit.position;
      }
      else
      {
         UnityEngine.Debug.Log("...");
         return sourcePostion;
      }
```

```
    }

    public void StopSimulation()
    {
        Debug.LogWarning("Stop");
        simulationToLLM.StopAllCoroutines();
        simulationToLLM.ReportEnd(HandleReport);

        targetInstance.GetComponent<TargetAgent>().SimulationStop();

        foreach (var officer in officerAgentsInstances)
        {
            officer.GetComponent<SearchOfficerAgent>().SimulationStop();
        }

        WriteAllCommands();
        WriteLog();
        MakeRenderOfFinalState(1920, 1080, simulationCamera, "MainRender");
        MakeRenderOfFinalState(7000, 7000, secondaryCamera,
"SecondaryRender");
    }

    private string CreateDirectory()
    {
        directoryPath = Path.Combine(Application.dataPath,
$"Simulation{currentDateTime.ToString("dd-mm")}-{SimulationGUID}");
```

```
        Directory.CreateDirectory(directoryPath);
        return directoryPath;
    }

    private void HandleReport(string response)
    {
        Debug.Log(response);

        var filePath = Path.Combine(directoryPath,
$"Simulation{currentDateTime.ToString("dd-mm")}-{SimulationGUID} LLM
Summary.txt");



        File.WriteAllText(filePath, response);
    }

    private void WriteAllCommands()
    {
        var filePath = Path.Combine(directoryPath,
$"Simulation{currentDateTime.ToString("dd-mm")}-{SimulationGUID} LLM
Commands To Simulation.txt");

        File.WriteAllText(filePath, CollectScenarioInfo(end: true));
    }

    private void WriteLog()
```

```
    {
        string jsonSerializedLog = JsonSerializer.Serialize(Log);

        var filePath = Path.Combine(directoryPath,
$"Simulation{currentDateTime.ToString("dd-mm")}-{SimulationGUID}
Log.txt");

        File.WriteAllText(filePath, jsonSerializedLog);
    }

    private void MakeRenderOfFinalState(int resolutionWidth, int
resolutionHeight, Camera camera, string name)
    {
        //var resolutionWidth = 1920;
        //var resolutionHeight = 1080;
        // Create a RenderTexture with the desired resolution
        RenderTexture renderTexture = new RenderTexture(resolutionWidth,
resolutionHeight, 24);
        camera.targetTexture = renderTexture;

        // Render the camera's view
        RenderTexture.active = renderTexture;
        camera.Render();

        // Create a Texture2D to store the rendered image
```

```
        Texture2D screenshot = new Texture2D(resolutionWidth, resolutionHeight,
TextureFormat.RGB24, false);
        screenshot.ReadPixels(new Rect(0, 0, resolutionWidth, resolutionHeight),
0, 0);
        screenshot.Apply();

        // Reset camera and RenderTexture
        camera.targetTexture = null;
        RenderTexture.active = null;
        Destroy(renderTexture);

        // Convert the Texture2D to PNG format
        byte[] bytes = screenshot.EncodeToPNG();
        Destroy(screenshot);

        // Save the PNG file
        var filePath = Path.Combine(directoryPath,
$"Simulation{SimulationGUID}-{currentDateTime.ToString("dd-mm")}
{name}.png");
        File.WriteAllBytes(filePath, bytes);

        Debug.Log($"Screenshot saved to: {filePath}");
    }

    public string CollectScenarioInfo(bool end = false)
    {
```

```
        string scenarioInfo = $"Last known approximate target position is in circle
with center {targetPotentialLocation} within
{targetZone.transform.localScale.x} radius\n";

        SortedDictionary<int, string> sortedRecievedCommands = new
SortedDictionary<int, string>();
        SortedDictionary<int, string> sortedExecutingCommands = new
SortedDictionary<int, string>();
        SortedDictionary<int, string> sortedExecutedCommands = new
SortedDictionary<int, string>();

        foreach (var recievedCommand in this.recievedCommands)
        {

sortedRecievedCommands.Add(recievedCommands.IndexOf(recievedCommand
), recievedCommand.CommandPrompt);

            if (executingCommands.Contains(recievedCommand))

sortedExecutingCommands.Add(executingCommands.IndexOf(recievedComma
nd), recievedCommand.CommandPrompt);

            if (executedCommands.Contains(recievedCommand))

sortedExecutedCommands.Add(executedCommands.IndexOf(recievedCommand
), recievedCommand.CommandPrompt);
```

```
        }
        if (end)
            foreach (var recievedCommand in sortedRecievedCommands)
            {
                //scenarioInfo += "Recieved: " + recievedCommand.Value + "\n";
                //if (sortedExecutingCommands.Contains(recievedCommand))
                //    scenarioInfo += "Executing now: " +
sortedExecutingCommands.Where(e => e.Value ==
recievedCommand.Value).FirstOrDefault().Value + "\n";

                scenarioInfo += "Executed: " + sortedExecutedCommands.Where(e =>
e.Value == recievedCommand.Value).FirstOrDefault().Value + "\n";
            }
        else
        {
            SortedDictionary<int, string> usefulCommands = new
SortedDictionary<int, string>();
            if (sortedExecutedCommands.Count > 5)
            {
                for (int i = 0; i < 5; i++)

usefulCommands.Add(sortedExecutedCommands.Reverse().Skip(i).First().Key,
sortedExecutedCommands.Reverse().Skip(i).First().Value);
            }
            else
            {
```

```
            foreach (var command in sortedExecutedCommands)
            {
                usefulCommands.Add(command.Key, command.Value);
            }
        }

        foreach (var usefulCommand in usefulCommands)
        {
            scenarioInfo += "Executed: " + usefulCommand.Value + "\n";
        }
    }

    foreach (var agent in officerAgentsInstances)
    {
        scenarioInfo +=
$"SearchOfficr[{agent.GetComponent<SearchOfficerAgent>().ID}] now locatd
at {agent.transform.position} \n";
    }
    if (end)
    {
        if (agentClues.Count > 0)
        {
            foreach (var kvp in agentClues)
            {
                scenarioInfo += $"Agent[{kvp.Key.ID}] found clues of target path
at [";
```

```
            foreach (var clue in kvp.Value)
            {
                scenarioInfo += $"Clue found at{clue.transform.position}";
            }

            scenarioInfo += "]";
        }
    }
}
else
{
    if (agentClues.Count > 0)
    {
        var usefulCluesCount = agentClues.Count % 5;
        HashSet<GameObject> usefulClues = new HashSet<GameObject>();
        float minimalDistanceToTarget = float.MaxValue;
        for (int i = 0; i < usefulCluesCount; i++)
        {
            GameObject closestClue = null;
            foreach (var kvp in agentClues)
            {
                foreach (var clue in kvp.Value)
                {
                    float distanceToTargetFromClue =
Vector3.Distance(clue.transform.position, targetInstance.transform.position);
```

```
                    if (distanceToTargetFromClue < minimalDistanceToTarget
&& !usefulClues.Contains(clue))
                    {
                        minimalDistanceToTarget = distanceToTargetFromClue;
                        closestClue = clue;
                    }
                }
            }
            if (closestClue != null)
                usefulClues.Add(closestClue);
        }
        foreach (var clue in usefulClues)
        {
            scenarioInfo += $"Clue found at{clue.transform.position}";
        }
      }
    }

    return scenarioInfo;
  }

  public bool TryAddClues(KeyValuePair<SearchAgent, List<GameObject>>
agentToClues)
  {
    if (agentClues.ContainsKey(agentToClues.Key))
    {
```

```
        agentClues[agentToClues.Key] = agentToClues.Value;
        FindPotentialLocationOfTarget();

        foreach (var clue in agentToClues.Value)
        {
            if (!clues.Contains(clue))
            {
                clues.Add(clue);
                cluePositions.Add(clue.transform.position);
            }
        }

        return true;
    }
    else
    {
        agentClues.Add(agentToClues.Key, agentToClues.Value);
        FindPotentialLocationOfTarget();

        foreach (var clue in agentToClues.Value)
        {
            if (!clues.Contains(clue))
            {
                clues.Add(clue);
                cluePositions.Add(clue.transform.position);
            }
```

```
        }
        return true;
    }

    return false;
}

public void RegisterRecievedCommand(Command command)
{
    recievedCommands.Add(command);
}

public void RegisterExecutingCommand(Command command)
{
    executingCommands.Add(command);
}

public void ConfirmExecutionOfCommand(Command command)
{
    executedCommands.Add(command);
    executingCommands.Remove(command);
}

public void RegisterREjectedCommand(string command)
{
    rejectedCommadns.Add(command);
```

```
    }

    public EnviromentSpeedSO GetEnviromentSpeed(EnviromentEnum
environmentEnum)
    {
        return enviromentSpeed.Where(e => e.GetEnviromentType() ==
environmentEnum).FirstOrDefault();
    }

    private GameObject FindPotentialLocationOfTarget()
    {
        float maxDistance = float.MinValue;
        GameObject targetLocation = targetZone;
        if (agentClues.Count > 0)
        {
            foreach (var kvp in agentClues)
            {
                foreach (var clue in kvp.Value)
                {
                    float distance = Vector3.Distance(targetZone.transform.position,
clue.transform.position);
                    if (distance > maxDistance)
                    {
                        maxDistance = distance;
                        targetLocation = clue;
                    }
```

```
            }
        }
    }
    targetPotentialLocation = targetLocation.transform.position;
    return targetLocation;
}

private void OnTriggerStay(Collider other)
{
}

// Start is called before the first frame update
private void Start()
{
}

// Update is called once per frame
private void Update()
{
}
}
```

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

public class ForestMeshGeneratorV2 : MonoBehaviour
{
    public Texture2D texture;  // The 512x512 texture
    public float meshWidth = 6500f;  // The target width for the entire mesh
```

```
    public float meshHeight = 6500f;  // The target height for the entire mesh
    public float extrusionHeight = 1000f;  // Height of the extrusion (for black
areas)

    public Color color = Color.green;
    public float opacity = 0.5f;

    void Start()
    {
        Mesh generatedMesh = GenerateMeshFromTexture();
#if UNITY_EDITOR
        SaveMesh(generatedMesh, "Assets/GeneratedMeshOptimized.asset");
#endif
    }

    Mesh GenerateMeshFromTexture()
    {
        Mesh mesh = new Mesh();
        mesh.indexFormat = IndexFormat.UInt32;
        int textureWidth = texture.width;
        int textureHeight = texture.height;

        float unitWidth = meshWidth / textureWidth;
        float unitHeight = meshHeight / textureHeight;

        List<Vector3> vertices = new List<Vector3>();
        List<int> triangles = new List<int>();

        // Traverse the texture and create box-like extrusions for black pixels
        for (int y = 0; y < textureHeight; y++)
        {
            for (int x = 0; x < textureWidth; x++)
            {
                Color pixelColor = texture.GetPixel(x, y);
                if (IsBlack(pixelColor))
                {
                    // Generate a box at this pixel position
                    GenerateMesh(vertices, triangles, x, y, unitWidth,
unitHeight);
                }
            }
        }

        mesh.vertices = vertices.ToArray();
        mesh.triangles = triangles.ToArray();
        mesh.RecalculateNormals();

        GameObject meshObject = new GameObject("GeneratedMesh",
typeof(MeshFilter), typeof(MeshRenderer));
        meshObject.GetComponent<MeshFilter>().mesh = mesh;

        Material material = new Material(Shader.Find("Standard"));
        material.color = new Color(color.r, color.g, color.b, opacity);
        material.SetFloat("_Mode", 3);
        material.SetInt("_SrcBlend", (int)BlendMode.SrcAlpha);
        material.SetInt("_DstBlend", (int)BlendMode.OneMinusSrcAlpha);
        material.SetInt("_ZWrite", 0);
```

```
        material.DisableKeyword("_ALPHATEST_ON");
        material.EnableKeyword("_ALPHABLEND_ON");
        material.DisableKeyword("_ALPHAPREMULTIPLY_ON");

        material.renderQueue = 3000;

        // Enable double-sided rendering
        material.SetInt("_Cull", (int)CullMode.Off);  // Disable backface culling
        meshObject.GetComponent<MeshRenderer>().material = material;

        meshObject.transform.position = new Vector3(-meshWidth / 2, 0, -meshHeight
/ 2);

        return mesh;
    }

    void GenerateMesh(List<Vector3> vertices, List<int> triangles, int x, int y,
float unitWidth, float unitHeight)
    {
        // Calculate quad corner positions at the bottom
        Vector3 bottomLeft = new Vector3(x * unitWidth, 0, y * unitHeight);
        Vector3 bottomRight = new Vector3((x + 1) * unitWidth, 0, y * unitHeight);
        Vector3 topLeft = new Vector3(x * unitWidth, 0, (y + 1) * unitHeight);
        Vector3 topRight = new Vector3((x + 1) * unitWidth, 0, (y + 1) *
unitHeight);

        // Top surface of the extruded mesh (box top)
        Vector3 bottomLeftTop = new Vector3(x * unitWidth, extrusionHeight, y *
unitHeight);
        Vector3 bottomRightTop = new Vector3((x + 1) * unitWidth, extrusionHeight,
y * unitHeight);
        Vector3 topLeftTop = new Vector3(x * unitWidth, extrusionHeight, (y + 1) *
unitHeight);
        Vector3 topRightTop = new Vector3((x + 1) * unitWidth, extrusionHeight, (y
+ 1) * unitHeight);

        // Add vertices for the bottom and top surfaces
        int vertexIndex = vertices.Count;

        // Bottom surface
        vertices.Add(bottomLeft);  // Bottom left
        vertices.Add(bottomRight); // Bottom right
        vertices.Add(topLeft);     // Top left
        vertices.Add(topRight);    // Top right

        // Top surface
        vertices.Add(bottomLeftTop);  // Bottom left (Top)
        vertices.Add(bottomRightTop); // Bottom right (Top)
        vertices.Add(topLeftTop);     // Top left (Top)
        vertices.Add(topRightTop);    // Top right (Top)

        // Add triangles for the bottom quad (clockwise winding)
        triangles.Add(vertexIndex);     // Bottom left
        triangles.Add(vertexIndex + 2); // Top left
        triangles.Add(vertexIndex + 1); // Bottom right
        triangles.Add(vertexIndex + 1); // Bottom right
        triangles.Add(vertexIndex + 2); // Top left
```

```
        triangles.Add(vertexIndex + 3); // Top right

        // Add triangles for the top quad (counterclockwise winding)
        triangles.Add(vertexIndex + 4); // Bottom left (Top)
        triangles.Add(vertexIndex + 6); // Top left (Top)
        triangles.Add(vertexIndex + 5); // Bottom right (Top)
        triangles.Add(vertexIndex + 5); // Bottom right (Top)
        triangles.Add(vertexIndex + 6); // Top left (Top)
        triangles.Add(vertexIndex + 7); // Top right (Top)

        // Generate outer walls around the box, checking neighboring pixels
        GenerateOuterWalls(vertices, triangles, x, y, bottomLeft, bottomRight,
topLeft, topRight, bottomLeftTop, bottomRightTop, topLeftTop, topRightTop,
unitWidth, unitHeight);
    }

    void GenerateOuterWalls(List<Vector3> vertices, List<int> triangles,
        int x, int y, Vector3 bottomLeft, Vector3 bottomRight, Vector3 topLeft,
Vector3 topRight,
        Vector3 bottomLeftTop, Vector3 bottomRightTop, Vector3 topLeftTop, Vector3
topRightTop,
        float unitWidth, float unitHeight)
    {
        int vertexIndex = vertices.Count;

        // Left wall - check if there is no black pixel to the left
        if (x == 0 || !IsBlack(texture.GetPixel(x - 1, y)))
        {
            vertices.Add(bottomLeft);
            vertices.Add(topLeft);
            vertices.Add(bottomLeftTop);
            vertices.Add(topLeftTop);
            AddQuadTriangles(triangles, vertexIndex);
            vertexIndex += 4;
        }

        // Right wall - check if there is no black pixel to the right
        if (x == texture.width - 1 || !IsBlack(texture.GetPixel(x + 1, y)))
        {
            vertices.Add(bottomRight);
            vertices.Add(topRight);
            vertices.Add(bottomRightTop);
            vertices.Add(topRightTop);
            AddQuadTriangles(triangles, vertexIndex);
            vertexIndex += 4;
        }

        // Front wall - check if there is no black pixel below
        if (y == 0 || !IsBlack(texture.GetPixel(x, y - 1)))
        {
            vertices.Add(bottomLeft);
            vertices.Add(bottomRight);
            vertices.Add(bottomLeftTop);
            vertices.Add(bottomRightTop);
            AddQuadTriangles(triangles, vertexIndex);
            vertexIndex += 4;
        }
```

```
        // Back wall - check if there is no black pixel above
        if (y == texture.height - 1 || !IsBlack(texture.GetPixel(x, y + 1)))
        {
            vertices.Add(topLeft);
            vertices.Add(topRight);
            vertices.Add(topLeftTop);
            vertices.Add(topRightTop);
            AddQuadTriangles(triangles, vertexIndex);
        }
    }

    void AddQuadTriangles(List<int> triangles, int vertexIndex)
    {
        triangles.Add(vertexIndex);     // Bottom left
        triangles.Add(vertexIndex + 2); // Top left
        triangles.Add(vertexIndex + 1); // Bottom right
        triangles.Add(vertexIndex + 1); // Bottom right
        triangles.Add(vertexIndex + 2); // Top left
        triangles.Add(vertexIndex + 3); // Top right
    }

    bool IsBlack(Color color)
    {
        return color.r < 0.1f && color.g < 0.1f && color.b < 0.1f;
    }
#if UNITY_EDITOR
    void SaveMesh(Mesh mesh, string fileName)
    {
        AssetDatabase.CreateAsset(mesh, fileName);
        AssetDatabase.SaveAssets();
        Debug.Log("Mesh saved as " + fileName);
    }
#endif
}
```

```
using ABM.Base;

using ABM.Enviroment;

using ABM.Utils;

using Sirenix.Utilities;

using System;

using System.Collections;

using System.Collections.Generic;

using System.Collections.ObjectModel;
```

```
using System.Collections.Specialized;
using System.Linq;
using Unity.VisualScripting;
using UnityEngine;
using UnityEngine.AI;
using Random = UnityEngine.Random;

public class SearchAgent : AbstractAgent
{
    [SerializeField]
    public int ID;

    [SerializeField]
    private SimulationController simulationController;

    [SerializeField]
    private NavMeshAgent navMeshAgent;

    [SerializeField]
    private GameObject targetZone;

    [SerializeField]
    private GameObject agentOfficer;

    [SerializeField]
    private float chanceToWrongDirection = 0.3f;
```

```
    [SerializeField]
    private float nextWaypointDistance = 200f;

    [SerializeField]
    private float waypointSphereDiameter = 40;

    [SerializeField]
    private float proficencySpeedMultiplier = 3.3f;

    [SerializeField]
    private Vector3 currentTargetDestinationPosition;

    [SerializeField]
    private bool isNearDestination = false;
    [SerializeField]
    private float distanceToDestinationThreshold = 5f;

    [SerializeField]
    private ObservableCollection<GameObject> enviromentObjects;
    [SerializeField]
    private List<GameObject> testEnviromentObjects = new
List<GameObject>();

    [SerializeField]
    public Command currentCommand;
```

```
[SerializeField]
private Material waypointMaterial;

[SerializeField]
private List<GameObject> reportedClues = new List<GameObject>();

[SerializeField]
private List<GameObject> foundClues = new List<GameObject>();

[SerializeField]
private Detector detector;

[SerializeField]
private GameObject detectedTarget;

[SerializeField]
public bool isRoaming;

private float roamDiameter = 400f;

public void Init(int ID, GameObject targetZone, GameObject officer,
   SimulationController simulationController)
{
   base.Init();
   this.simulationController = simulationController;
```

```
        //this.Controller = simulationController;
        this.agentOfficer = officer;
        this.targetZone = targetZone;
        this.ID = ID;

        navMeshAgent = GetComponent<NavMeshAgent>();

        detector = GetComponentInChildren<Detector>();

        //SetInitialNavMeshParameters();
        //SetTargetDirection();

        detector.observedDetectedObjects.CollectionChanged += OnDetection;

    }



    void Start()
    {
        enviromentObjects = new ObservableCollection<GameObject>();
        enviromentObjects.CollectionChanged += OnEnviromentChanged;
    }

    private void OnDetection(object sender, NotifyCollectionChangedEventArgs
e)
```

```
    {



        if (detector.observedDetectedObjects.Any(dt => dt.gameObject.tag ==
"Target"))
        {
            if (detectedTarget == detector.observedDetectedObjects.Where(dt =>
dt.gameObject.tag == "Target").FirstOrDefault())
                return;



            detectedTarget = detector.observedDetectedObjects.Where(dt =>
dt.gameObject.tag == "Target").FirstOrDefault();



agentOfficer.GetComponent<SearchOfficerAgent>().SetDetectedTarget(detected
Target.GetComponent<TargetAgent>());

            DestroyQueue("CheckDistanceToCurrentTargetDestination");
            DestroyQueue("Move");

            Vector3 position = detectedTarget.transform.position;

            Command overwriteCommand = new Command();
            overwriteCommand.Action = (position) => { };
```

```
        overwriteCommand.CommandPrompt = ("Overwritten by cause: Target
was detected");

        SetMoveDirection(detectedTarget.transform.position,
overwriteCommand);
    }
    else if (detector.observedDetectedObjects.Any(dt => dt.gameObject.tag ==
"TargetPath"))
    {
        detector.observedDetectedObjects.Where(dt => dt.gameObject.tag ==
"TargetPath").ForEach(dt =>
        {
            agentOfficer.GetComponent<SearchOfficerAgent>().ReportClue(this,
dt.gameObject);
        });
    }









  }

  private void OnEnviromentChanged(object sender,
System.Collections.Specialized.NotifyCollectionChangedEventArgs e)
```

```
    {
        HashSet<GameObject> enviromentObjectsSet = new
HashSet<GameObject>(enviromentObjects);
        enviromentObjects.Remove(null);
        SortedList<EnviromentEnum, GameObject> enviromentObjectsDictionary
= new SortedList<EnviromentEnum, GameObject>();

        enviromentObjectsDictionary.Clear();
        foreach (var item in enviromentObjectsSet)
        {
            if (item != null && Enum.IsDefined(typeof(EnviromentEnum),
item.tag))

enviromentObjectsDictionary.Add(Enum.Parse<EnviromentEnum>(item.tag),
item);
        }

        if (enviromentObjectsDictionary.Count > 0)
            navMeshAgent.speed =
simulationController.GetEnviromentSpeed(enviromentObjectsDictionary.Last().
Key).GetEnviromentSpeed() * proficencySpeedMultiplier;

    }

    public void OverideCommands(Vector3 destination, Command command)
    {
```

```
        //agentOfficer.GetComponent<SearchOfficerAgent>().Report(this,
currentCommand);
        currentCommand = command;

        SetMoveDirection(destination, command);

    }

    public void SetMoveDirection(Vector3 moveDestination, Command
command)
    {
        if (command.CommandPrompt.Contains("Send"))
            isRoaming = false;

        Transform target = this.transform;
        // Create Sphere at distance nextWaypointDistance from
target.transform.position in direction of targetZone.transform.position it must
intersect with terrain
        Vector3 direction = (moveDestination -
target.transform.position).normalized;

        if (Random.value <= chanceToWrongDirection)
        {
            // Random direction within 90 degrees from curent Vector3 direction
            Vector3 oldDdirection = direction;
```

```
            direction = GetRandomDirectionWithinAngleXZ(direction, 5);
            Debug.Log(string.Format("Wrong direction: {0}, correct was {1}",
direction, oldDdirection));
        }



        Vector3 sphereCenter = target.transform.position + direction *
Math.Min(nextWaypointDistance, Vector3.Distance(transform.position,
moveDestination));



        GameObject sphereCollider =
GameObject.CreatePrimitive(PrimitiveType.Sphere);
        sphereCollider.tag = "SearchPartyPath";

        float terrainHeight =
simulationController.simulationTerrainGameObject.GetComponent<Terrain>().
SampleHeight(sphereCenter);
        sphereCenter = new Vector3(sphereCenter.x, terrainHeight,
sphereCenter.z);

        sphereCollider.transform.position = sphereCenter;

        sphereCollider.transform.localScale = new
Vector3(waypointSphereDiameter, waypointSphereDiameter,
waypointSphereDiameter);
```

```
        sphereCollider.GetComponent<SphereCollider>().isTrigger = true;

        sphereCollider.GetComponent<MeshRenderer>().material =
waypointMaterial;

        float waypointSphereRadius = waypointSphereDiameter / 2f;

        Vector3 randomPositionInsideSphere = sphereCenter + new Vector3(
            Random.Range(-waypointSphereRadius, waypointSphereRadius),
            0,
            Random.Range(-waypointSphereRadius, waypointSphereRadius));

        terrainHeight =
simulationController.simulationTerrainGameObject.GetComponent<Terrain>().
SampleHeight(randomPositionInsideSphere);
        randomPositionInsideSphere.y = terrainHeight;
        currentTargetDestinationPosition = randomPositionInsideSphere;

        if (!command.CommandPrompt.Contains("Overwritten"))
            if (moveDestination != currentTargetDestinationPosition)
                command.CommandPrompt = $"Overwritten by limitation of
maximum distance {nextWaypointDistance} meters. New coordinates are
{currentTargetDestinationPosition}" + command.CommandPrompt;

        navMeshAgent.SetDestination(currentTargetDestinationPosition);
        if (isRoaming)
```

```
            Destroy(sphereCollider);

        StartMoveSequence(command);

    }

    private void Roam()
    {
        GameObject sphereCollider =
GameObject.CreatePrimitive(PrimitiveType.Sphere);
        sphereCollider.transform.position = agentOfficer.transform.position;

        sphereCollider.transform.localScale = new Vector3(roamDiameter,
roamDiameter, roamDiameter);

        float waypointSphereRadius = waypointSphereDiameter / 2f;

        Vector3 randomPositionInsideSphere = sphereCollider.transform.position +
new Vector3(
            Random.Range(-waypointSphereRadius, waypointSphereRadius),
            0,
            Random.Range(-waypointSphereRadius, waypointSphereRadius));

        float terrainHeight =
simulationController.simulationTerrainGameObject.GetComponent<Terrain>().
SampleHeight(randomPositionInsideSphere);
```

```
        randomPositionInsideSphere.y = terrainHeight;
        currentTargetDestinationPosition = randomPositionInsideSphere;

        var command = new Command();
        command.CommandPrompt = $"Agent[{ID}]. Roam around search
SearchOfficer[{agentOfficer.GetComponent<SearchOfficerAgent>().ID}] ";

        Destroy(sphereCollider);
        SetMoveDirection(currentTargetDestinationPosition, command);
        isRoaming = true;

    }

    private void StartMoveSequence(Command command)
    {
        navMeshAgent.isStopped = false;

        CreateQueue(Move, 1, 105);
        CreateQueue(CheckDistanceToCurrentTargetDestination, 1, 100);
        currentCommand = command;



    }

    private void Move()
    {
```

```
        navMeshAgent.velocity = Vector3.zero;
        navMeshAgent.nextPosition = this.transform.position +
navMeshAgent.desiredVelocity * 0.03f;
        transform.LookAt(navMeshAgent.nextPosition, Vector3.up);
        transform.position = navMeshAgent.nextPosition;
    }

    private void CheckDistanceToCurrentTargetDestination()
    {
        if (detectedTarget != null)
        {
            if (Vector3.Distance(transform.position,
detectedTarget.transform.position) <= 30)
            {
                simulationController.StopSimulation();
            }
        }

        if (Vector3.Distance(transform.position, agentOfficer.transform.position) >
roamDiameter)
        {
            isRoaming = false;
            DestroyQueue("CheckDistanceToCurrentTargetDestination");
            DestroyQueue("Move");
            var command = new Command();
```

```
            command.CommandPrompt = $"Agent{ID} Returning to
SearchOfficer[{agentOfficer.GetComponent<SearchOfficerAgent>().ID}]";
            SetMoveDirection(agentOfficer.transform.position, command);




        }



        float distance = Vector3.Distance(this.transform.position,
currentTargetDestinationPosition);
        if (distance < distanceToDestinationThreshold)
        { //reached target
            isNearDestination = true;

            navMeshAgent.isStopped = true;

            DestroyQueue("CheckDistanceToCurrentTargetDestination");
            DestroyQueue("Move");

            if (!currentCommand.CommandPrompt.Contains("Overwriten by
moving") && isRoaming == false)
            {
                agentOfficer.GetComponent<SearchOfficerAgent>().Report(this,
currentCommand);
                isRoaming = true;
```

```
        }

        foreach (var clue in foundClues)
        {
            if (!reportedClues.Contains(clue))
            {
                reportedClues.Add(clue);

agentOfficer.GetComponent<SearchOfficerAgent>().ReportClue(this, clue);
            }
        }

        Roam();

        //SetupStationary();
    }
    else
    {
        isNearDestination = false;
    }
  }

  private Vector3 GetRandomDirectionWithinAngleXZ(Vector3 direction, float
angle)
  {
    // Normalize the direction and project it onto the XZ plane
```

```
    direction.y = 0;
    direction.Normalize();

    // Pick a random angle within the range
    float randomAngle = Random.Range(-angle, angle);

    // Generate a quaternion for the rotation around the Y-axis (XZ plane)
    Quaternion rotation = Quaternion.AngleAxis(randomAngle, Vector3.up);

    // Rotate the direction
    return rotation * direction;
}

private void SetInitialNavMeshParameters()
{
    navMeshAgent.updatePosition = false;
    navMeshAgent.velocity = Vector3.zero;
    navMeshAgent.acceleration = 0f;
}

private void OnTriggerEnter(Collider other)
{
    enviromentObjects.Add(other.gameObject);

    if (other.gameObject.tag == "TargetPath")
    {
```

```
            if (!foundClues.Contains(other.gameObject))
                foundClues.Add(other.gameObject);
        }

    }

    private void OnTriggerExit(Collider other)
    {
        enviromentObjects.Remove(other.gameObject);
    }

    public void SimulationStop()
    {
        DestroyQueue("CheckDistanceToCurrentTargetDestination");
        DestroyQueue("Move");
    }
}
```

```
using System.Collections;
using System.Collections.Generic;
using System.Collections.ObjectModel;
using UnityEngine;
using UnityEngine.AI;
using ABM.Base;
using ABM.Enviroment;
using Random = UnityEngine.Random;
```

```
using System.Linq;
using System;
using ABM.Utils;
using System.Collections.Specialized;
using Sirenix.Utilities;



public class SearchOfficerAgent : AbstractAgent
{
    [SerializeField]
    public int ID;

    [SerializeField]
    private SimulationController simulationController;

    [SerializeField]
    private NavMeshAgent navMeshAgent;

    [SerializeField]
    private GameObject targetZone;

    [SerializeField]
    private float chanceToWrongDirection = 0.3f;

    [SerializeField]
    private float nextWaypointDistance = 1000f;
```

```
    [SerializeField]
    private float waypointSphereDiameter = 80f;

    [SerializeField]
    private float proficencySpeedMultiplier = 3f;



    [SerializeField]
    private Vector3 currentTargetDestinationPosition;

    [SerializeField]
    private bool isNearDestination = false;
    [SerializeField]
    private float distanceToDestinationThreshold = 5f;

    [SerializeField]
    private ObservableCollection<GameObject> enviromentObjects;
    [SerializeField]
    private List<GameObject> testEnviromentObjects = new
List<GameObject>();

    [SerializeField]
    private List<GameObject> agents;

    [SerializeField]
```

```
    private Command currentCommand;

    [SerializeField]
    private Dictionary<GameObject, Command> agentToCommand = new
Dictionary<GameObject, Command>();

    [SerializeField]
    private Material waypointMaterial;

    [SerializeField]
    private List<GameObject> reportedClues = new List<GameObject>();

    [SerializeField]
    private List<GameObject> foundClues = new List<GameObject>();

    [SerializeField]
    private Dictionary<SearchAgent, List<GameObject>> agentClues = new
Dictionary<SearchAgent, List<GameObject>>();

    [SerializeField]
    private Detector detector;

    [SerializeField]
    private GameObject detectedTarget;

    [SerializeField]
```

```
    private bool waitingForResponse;
    [SerializeField]
    private bool executingCommand;

    private float timer;

    private float maxTimer = 20000f;

    public void Init(int ID, GameObject targetZone, List<GameObject> agents,
SimulationController simulationController)
    {
        base.Init();
        this.ID = ID;
        this.simulationController = simulationController;
        this.targetZone = targetZone;
        timer = maxTimer;
        this.agents = agents;

        navMeshAgent = GetComponent<NavMeshAgent>();

        SetInitialNavMeshParameters();
        waitingForResponse = false;
        executingCommand = false;
        RequestCommandFromLLM();

        detector = GetComponentInChildren<Detector>();
```

```
        detector.observedDetectedObjects.CollectionChanged += OnDetection;
        //SetInitialNavMeshParameters();
        //SetTargetDirection();



    }

    public void SetDetectedTarget(TargetAgent detectedTarget)
    {
        this.detectedTarget = detectedTarget.gameObject;

        DestroyQueue("CheckDistanceToCurrentTargetDestination");
        DestroyQueue("Move");

        Vector3 position = detectedTarget.transform.position;

        Command overwriteCommand = new Command();
        overwriteCommand.Action = (position) => { };
        overwriteCommand.CommandPrompt = ("Overwritten by cause: Target
was detected");

        SetMoveDirection(detectedTarget.transform.position, overwriteCommand);
    }
```

```
    public void Update()
    {
        timer -= Time.fixedDeltaTime / 1000;
        if (timer <= 0)
        {
            timer = maxTimer;
            if (!waitingForResponse && !executingCommand)
            {
                RequestCommandFromLLM();
            }
        }
    }



    private void OnDetection(object sender, NotifyCollectionChangedEventArgs
e)
    {

        if (detector.observedDetectedObjects.Any(dt => dt.gameObject.tag ==
"Target"))
        {

            if (detectedTarget == detector.observedDetectedObjects.Where(dt =>
dt.gameObject.tag == "Target").FirstOrDefault())
                return;
```

```
        detectedTarget = detector.observedDetectedObjects.Where(dt =>
dt.gameObject.tag == "Target").FirstOrDefault();

        DestroyQueue("CheckDistanceToCurrentTargetDestination");
        DestroyQueue("Move");

        Vector3 position = detectedTarget.transform.position;

        Command overwriteCommand = new Command();
        overwriteCommand.Action = (position) => { };
        overwriteCommand.CommandPrompt = ("Overwritten by cause: Target
was detected");

        SetMoveDirection(detectedTarget.transform.position,
overwriteCommand);
    }
    else if (detector.observedDetectedObjects.Any(dt => dt.gameObject.tag ==
"TargetPath"))
    {
        detector.observedDetectedObjects.Where(dt => dt.gameObject.tag ==
"TargetPath").ForEach(dt =>
        {
            ReportClue(agents[0].GetComponent<SearchAgent>(),
dt.gameObject);
        });
```

```
        }
    }

    public void SetMoveDirection(Vector3 moveDestination, Command
command)
    {

        executingCommand = true;
        Command overideComand = new Command();
        overideComand.CommandPrompt = command.CommandPrompt;



        foreach (var agent in agents)
        {
            if (!agent.GetComponent<SearchAgent>().isRoaming)
            {
                if (agentToCommand.ContainsKey(agent))
                    agentToCommand.Remove(agent);
                Command agentCommand = overideComand;
                agentCommand.CommandPrompt =
/*agentCommand.CommandPrompt +*/ $"Overwriten by moving officer to
{moveDestination}. Agent[{agent.GetComponent<SearchAgent>().ID}] now
moving along to {moveDestination}";


                agentToCommand.Add(agent, agentCommand);
```

```
agent.GetComponent<SearchAgent>().OverideCommands(moveDestination,
agentCommand);
            }

        }

        Transform target = this.transform;
        // Create Sphere at distance nextWaypointDistance from
target.transform.position in direction of targetZone.transform.position it must
intersect with terrain
        Vector3 direction = (moveDestination -
target.transform.position).normalized;

        if (Random.value <= chanceToWrongDirection)
        {
            // Random direction within 90 degrees from curent Vector3 direction
            Vector3 oldDdirection = direction;
            direction = GetRandomDirectionWithinAngleXZ(direction, 5);
            Debug.Log(string.Format("Wrong direction: {0}, correct was {1}",
direction, oldDdirection));
        }
```

```
        Vector3 sphereCenter = target.transform.position + direction *
Math.Min(nextWaypointDistance, Vector3.Distance(transform.position,
moveDestination));

        GameObject sphereCollider =
GameObject.CreatePrimitive(PrimitiveType.Sphere);
        sphereCollider.tag = "SearchPartyPath";

        float terrainHeight =
simulationController.simulationTerrainGameObject.GetComponent<Terrain>().
SampleHeight(sphereCenter);
        sphereCenter = new Vector3(sphereCenter.x, terrainHeight,
sphereCenter.z);

        sphereCollider.transform.position = sphereCenter;

        sphereCollider.transform.localScale = new
Vector3(waypointSphereDiameter, waypointSphereDiameter,
waypointSphereDiameter);
        sphereCollider.GetComponent<SphereCollider>().isTrigger = true;

        sphereCollider.GetComponent<MeshRenderer>().material =
waypointMaterial;

        float waypointSphereRadius = waypointSphereDiameter / 2f;
```

```
        Vector3 randomPositionInsideSphere = sphereCenter + new Vector3(
            Random.Range(-waypointSphereRadius, waypointSphereRadius),
            0,
            Random.Range(-waypointSphereRadius, waypointSphereRadius));

        terrainHeight =
simulationController.simulationTerrainGameObject.GetComponent<Terrain>().
SampleHeight(randomPositionInsideSphere);
        randomPositionInsideSphere.y = terrainHeight;
        currentTargetDestinationPosition = randomPositionInsideSphere;

        if (moveDestination != currentTargetDestinationPosition)
            command.CommandPrompt = $"Overwritten by limitation of maximum
distance {nextWaypointDistance} meters. New coordinates are
{currentTargetDestinationPosition}" + command.CommandPrompt;

        navMeshAgent.SetDestination(currentTargetDestinationPosition);

        StartMoveSequence(command);
        waitingForResponse = false;

    }

    public void Scan(Vector3 locationVector, Command command)
    {
```

```
        executingCommand = true;
        float minDistance = float.MaxValue;
        GameObject closestAgent = agents[0];
        foreach (var agent in agents)
        {
            float currentAgentDistance = Vector3.Distance(agent.transform.position,
locationVector);
            if (currentAgentDistance < minDistance)
            {
                minDistance = currentAgentDistance;
                closestAgent = agent;
            }
        }



closestAgent.GetComponent<SearchAgent>().SetMoveDirection(locationVector
, command);
        simulationController.RegisterExecutingCommand(command);
        waitingForResponse = false;

        //Debug.Log("Not Implemented");
    }

    private Vector3 GetRandomDirectionWithinAngleXZ(Vector3 direction, float
angle)
```

```
{
    // Normalize the direction and project it onto the XZ plane
    direction.y = 0;
    direction.Normalize();

    // Pick a random angle within the range
    float randomAngle = Random.Range(-angle, angle);

    // Generate a quaternion for the rotation around the Y-axis (XZ plane)
    Quaternion rotation = Quaternion.AngleAxis(randomAngle, Vector3.up);

    // Rotate the direction
    return rotation * direction;
}

private void StartMoveSequence(Command command)
{
    navMeshAgent.isStopped = false;

    CreateQueue(Move, 1, 105);
    CreateQueue(CheckDistanceToCurrentTargetDestination, 1, 100);
    currentCommand = command;


}
```

```
    private void Move(){
        navMeshAgent.velocity = Vector3.zero;
        navMeshAgent.nextPosition = this.transform.position +
navMeshAgent.desiredVelocity*0.03f;
        transform.LookAt(navMeshAgent.nextPosition, Vector3.up);
        transform.position =  navMeshAgent.nextPosition;
    }

    private void CheckDistanceToCurrentTargetDestination()
    {
        if (detectedTarget != null)
        {
            if (Vector3.Distance(transform.position,
detectedTarget.transform.position) <= 30)
            {
                simulationController.StopSimulation();
            }
        }

        float distance = Vector3.Distance(this.transform.position,
currentTargetDestinationPosition);
        if(distance < distanceToDestinationThreshold){ //reached target
            isNearDestination = true;

            navMeshAgent.isStopped = true;
```

```
        DestroyQueue("CheckDistanceToCurrentTargetDestination");
        DestroyQueue("Move");

        waitingForResponse = false;
        executingCommand = false;

        if (currentCommand != null)
        {

simulationController.ConfirmExecutionOfCommand(currentCommand);
            currentCommand = null;



            RequestCommandFromLLM();
        }
        else
        {
            Debug.LogError("Current command was null! Something went
wrong!!!");
            RequestCommandFromLLM();
        }

        //SetupStationary();
    }
```

```
        else{
            isNearDestination = false;
        }
    }

    public void RejectCommand()
    {
        executingCommand = false;
        waitingForResponse = false;
        RequestCommandFromLLM();
    }


    public void Report(SearchAgent agent, Command command)
    {
        waitingForResponse = false;
        executingCommand = false;
        simulationController.ConfirmExecutionOfCommand(
            agentToCommand.Where(
                cta => cta.Key == agent && cta.Value ==
command).FirstOrDefault().Value);
        if (executingCommand == false && waitingForResponse == false)
            RequestCommandFromLLM();

    }
```

```
public void ReportClue(SearchAgent agent, GameObject clue)
{
    if (!foundClues.Contains(clue)) foundClues.Add(clue);
    if (!reportedClues.Contains(clue))
    {
        reportedClues.Add(clue);


    }

    if (agentClues.ContainsKey(agent))
    {
        agentClues[agent].Add(clue);
    }
    else
    {
        var clues = new List<GameObject>();
        clues.Add(clue);
        agentClues.Add(agent, clues);
    }

    foreach (var kvp in agentClues)
    {
        simulationController.TryAddClues(kvp);
    }
```

```
}

public Dictionary<SearchAgent, List<GameObject>> GetClues()
{
    return agentClues;
}


void Start()
{
    enviromentObjects = new ObservableCollection<GameObject>();
    enviromentObjects.CollectionChanged += OnEnviromentChanged;
}

public void SetDestination() { }

private void SetInitialNavMeshParameters()
{
    navMeshAgent.updatePosition = false;
    navMeshAgent.velocity = Vector3.zero;
    navMeshAgent.acceleration = 0f;
}
```

```
    private void OnEnviromentChanged(object sender,
System.Collections.Specialized.NotifyCollectionChangedEventArgs e)
    {
        HashSet<GameObject> enviromentObjectsSet = new
HashSet<GameObject>(enviromentObjects);
        enviromentObjects.Remove(null);
        SortedList<EnviromentEnum, GameObject> enviromentObjectsDictionary
= new SortedList<EnviromentEnum, GameObject>();

        foreach (var item in enviromentObjectsSet)
        {
            if (item != null && Enum.IsDefined(typeof(EnviromentEnum),
item.tag))

enviromentObjectsDictionary.Add(Enum.Parse<EnviromentEnum>(item.tag),
item);
        }

        if (enviromentObjectsDictionary.Count > 0)
            navMeshAgent.speed =
simulationController.GetEnviromentSpeed(enviromentObjectsDictionary.Last().
Key).GetEnviromentSpeed() * proficencySpeedMultiplier;
    }

    private void RequestCommandFromLLM()
    {
```

```
        var simulationToLLM = simulationController.simulationToLLM;
        if (waitingForResponse == false)
        {

            waitingForResponse = true;
            simulationToLLM.RequestCommandFromLLM(this);
        }
        else
        {
            //SearchOfficerAgent thisSearchOfficer = this;
            //StartCoroutine(CheckIfReadyToSendRequest(thisSearchOfficer =>
            //{
            //    simulationToLLM.RequestCommandFromLLM(thisSearchOfficer);
            //}, thisSearchOfficer));

        }

    }

    public void SendReqest(SearchOfficerAgent searchOfficer)
    {

simulationController.simulationToLLM.RequestCommandFromLLM(searchOffi
cer);
    }
```

```
    private IEnumerator
CheckIfReadyToSendRequest(Action<SearchOfficerAgent> SendRequest,
SearchOfficerAgent agent)
    {
        yield return waitingForResponse == false && executingCommand == false;

        if (executingCommand == false)
        {
            SendRequest?.Invoke(agent);
        }
    }



    private void OnTriggerEnter(Collider other)
    {
        enviromentObjects.Add(other.gameObject);
    }

    private void OnTriggerExit(Collider other)
    {
        enviromentObjects.Remove(other.gameObject);
    }

    public void SimulationStop()
    {
        DestroyQueue("CheckDistanceToCurrentTargetDestination");
```

```
        DestroyQueue("Move");


        foreach (var agent in agents)
        {
            agent.GetComponent<SearchAgent>().SimulationStop();
        }
    }
}
```

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using ABM.Base;
using ABM.Enviroment;
using UnityEngine.AI;
using System.Collections.ObjectModel;
using System;
using System.Linq;
using Random = UnityEngine.Random;
using System.Collections.Specialized;


public class TargetAgent : AbstractAgent
{
    [SerializeField]
    private SimulationController simulationController;

    [SerializeField]
    private NavMeshAgent navMeshAgent;

    [SerializeField]
    private GameObject cardinalTarget;

    [SerializeField]
    private float chanceToWrongDirection = 0.3f;

    [SerializeField]
    private float nextWaypointDistance = 120f;

    [SerializeField]
    private float waypointSphereDiameter = 80f;

    [SerializeField]
    private float proficencySpeedMultiplier = 0.9f;

    [SerializeField]
```

```
    private Vector3 currentTargetDestinationPosition;

    [SerializeField]
    private bool isNearDestination = false;
    [SerializeField]
    private float distanceToDestinationThreshold = 5f;

    [SerializeField]
    private ObservableCollection<GameObject> enviromentObjects;
    [SerializeField]
    private List<GameObject> testEnviromentObjects = new List<GameObject>();

    [SerializeField]
    private Material waypointMaterial;


    [SerializeField]
    private Detector detector;


    public void Init(GameObject cardinalTarget, SimulationController
simulationController)
    {
        base.Init();

        proficencySpeedMultiplier = 0.9f;
    }


    public void PostInit(GameObject cardinalTarget, SimulationController
simulationController)
    {
        this.simulationController = simulationController;
        this.cardinalTarget = cardinalTarget;

        navMeshAgent = GetComponent<NavMeshAgent>();

        detector = GetComponentInChildren<Detector>();

        detector.observedDetectedObjects.CollectionChanged += OnDetection;

        SetInitialNavMeshParameters();
        SetTargetDirection(null);

        Debug.Log($"{this} Initialized");
    }


    private void OnDetection(object sender, NotifyCollectionChangedEventArgs e)
    {
        GameObject detectedTarget = detector.observedDetectedObjects[0];

        DestroyQueue("CheckDistanceToCurrentTargetDestination");
        DestroyQueue("Move");

        Vector3 position = detectedTarget.transform.position;
```

```
        SetTargetDirection(detectedTarget);
    }

    private void StartMoveSequence()
    {
        navMeshAgent.isStopped = false;

        CreateQueue(Move, 1, 105);
        CreateQueue(CheckDistanceToCurrentTargetDestination, 1, 100);
    }

    // Start is called before the first frame update
    void Start()
    {
        enviromentObjects = new ObservableCollection<GameObject>();
        enviromentObjects.CollectionChanged += OnEnviromentChanged;
    }

    // Update is called once per frame
    void Update()
    {
        testEnviromentObjects.Clear();
        //testEnviromentObjects.AddRange(enviromentObjects);
    }

    private void OnEnviromentChanged(object sender,
System.Collections.Specialized.NotifyCollectionChangedEventArgs e)
    {
        HashSet<GameObject> enviromentObjectsSet = new
HashSet<GameObject>(enviromentObjects);
        enviromentObjects.Remove(null);
        SortedList<EnviromentEnum, GameObject> enviromentObjectsDictionary = new
SortedList<EnviromentEnum, GameObject>();
        enviromentObjectsDictionary.Clear();
        foreach (var item in enviromentObjectsSet)
        {
            if (item != null && Enum.IsDefined(typeof(EnviromentEnum), item.tag))
                if
(!enviromentObjectsDictionary.ContainsKey(Enum.Parse<EnviromentEnum>(item.tag)))

enviromentObjectsDictionary.Add(Enum.Parse<EnviromentEnum>(item.tag), item);
        }
        if (enviromentObjectsDictionary.Count > 0)
            navMeshAgent.speed =
simulationController.GetEnviromentSpeed(enviromentObjectsDictionary.Last().Key).Ge
tEnviromentSpeed() * proficencySpeedMultiplier;
    }

    private void Move(){
        navMeshAgent.velocity = Vector3.zero;
        navMeshAgent.nextPosition = this.transform.position +
navMeshAgent.desiredVelocity*0.03f;
        transform.LookAt(navMeshAgent.nextPosition, Vector3.up);
        transform.position =  navMeshAgent.nextPosition;
    }
```

```
    private void SetInitialNavMeshParameters()
    {
        navMeshAgent.updatePosition = false;
        navMeshAgent.velocity = Vector3.zero;
        navMeshAgent.acceleration = 0f;
    }

    private void SetTargetDirection(GameObject destination)
    {
        if (destination == null)
        {
            destination = cardinalTarget;
        }

        Transform target = this.transform;
        // Create Sphere at distance nextWaypointDistance from
target.transform.position in direction of targetZone.transform.position it must
intersect with terrain
        Vector3 direction = (destination.transform.position -
target.transform.position).normalized;

        if (Random.value <= chanceToWrongDirection)
        {
            // Random direction within 90 degrees from curent Vector3 direction
            Vector3 oldDdirection = direction;
            direction = GetRandomDirectionWithinAngleXZ(direction, 60);
            Debug.Log(string.Format("Wrong direction: {0}, correct was {1}",
direction, oldDdirection));
        }


        Vector3 sphereCenter = target.transform.position + direction *
nextWaypointDistance;

        GameObject sphereCollider =
GameObject.CreatePrimitive(PrimitiveType.Sphere);
        sphereCollider.transform.tag = "TargetPath";

        float terrainHeight =
simulationController.simulationTerrainGameObject.GetComponent<Terrain>().SampleHei
ght(sphereCenter);
        sphereCenter = new Vector3(sphereCenter.x, terrainHeight, sphereCenter.z);

        sphereCollider.transform.position = sphereCenter;

        sphereCollider.transform.localScale = new Vector3(waypointSphereDiameter,
waypointSphereDiameter, waypointSphereDiameter);
        sphereCollider.GetComponent<SphereCollider>().isTrigger = true;

        sphereCollider.GetComponent<MeshRenderer>().material = waypointMaterial;

        sphereCollider.GetComponent<MeshRenderer>().material.SetColor("color", new
Color(0.9f, .9f, .9f, .4f));
        float waypointSphereRadius = waypointSphereDiameter / 2f;

        Vector3 randomPositionInsideSphere = sphereCenter + new Vector3(
            Random.Range(-waypointSphereRadius, waypointSphereRadius),
```

```
            0,
            Random.Range(-waypointSphereRadius, waypointSphereRadius));

        terrainHeight =
simulationController.simulationTerrainGameObject.GetComponent<Terrain>().SampleHei
ght(randomPositionInsideSphere);
        randomPositionInsideSphere.y = terrainHeight;
        currentTargetDestinationPosition = randomPositionInsideSphere;


        navMeshAgent.SetDestination(currentTargetDestinationPosition);

        StartMoveSequence();

    }

    private Vector3 GetRandomDirectionWithinAngleXZ(Vector3 direction, float
angle)
    {
        // Normalize the direction and project it onto the XZ plane
        direction.y = 0;
        direction.Normalize();

        // Pick a random angle within the range
        float randomAngle = Random.Range(-angle, angle);

        // Generate a quaternion for the rotation around the Y-axis (XZ plane)
        Quaternion rotation = Quaternion.AngleAxis(randomAngle, Vector3.up);

        // Rotate the direction
        return rotation * direction;
    }


    private void CheckDistanceToCurrentTargetDestination()
    {
        float distance = Vector3.Distance(this.transform.position,
currentTargetDestinationPosition);
        if(distance < distanceToDestinationThreshold){ //reached target
            isNearDestination = true;

            navMeshAgent.isStopped = true;

            DestroyQueue("CheckDistanceToCurrentTargetDestination");
            DestroyQueue("Move");

            SetTargetDirection(null);

            //SetupStationary();
        }
        else{
            isNearDestination = false;
        }
    }

    public void SimulationStop()
    {
```

```
        DestroyQueue("CheckDistanceToCurrentTargetDestination");
        DestroyQueue("Move");
    }

    private void OnTriggerEnter(Collider other)
    {
        enviromentObjects.Add(other.gameObject);
    }

    private void OnTriggerExit(Collider other)
    {
        enviromentObjects.Remove(other.gameObject);
    }
}
```

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

public class Detector : MonoBehaviour
{

    [SerializeField]
    private List<string> tagsToDetectAndStore;

    [SerializeField]
    public List<GameObject> detectedObjects = new List<GameObject>();

    [SerializeField]
    public ObservableCollection<GameObject> observedDetectedObjects = new
ObservableCollection<GameObject>();

    [SerializeField]
    public List<GameObject> historyDetectedObjects;

    private void OnTriggerEnter(Collider other)
    {
        if (tagsToDetectAndStore.Contains(other.tag))
        {
            AddDetectedObject(other.gameObject);
        }
    }

    private void OnTriggerExit(Collider other)
    {
        if (tagsToDetectAndStore.Contains(other.tag))
        {
            RemoveDetectedObject(other.gameObject);
        }
    }

    private void AddDetectedObject(GameObject detectedObject)
    {
        if (detectedObject.gameObject.tag == "Detector")
            return;
```

```
        detectedObjects.Add(detectedObject);


        if (tagsToDetectAndStore.Contains(detectedObject.tag))
        {
            historyDetectedObjects.Add(detectedObject);
            observedDetectedObjects.Add(detectedObject);
        }


    }

    private void RemoveDetectedObject(GameObject detectedObject)
    {
        detectedObjects.Remove(detectedObject);
        observedDetectedObjects.Remove(detectedObject);

    }
}
```