ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

НАВЧАЛЬНО–НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра інженерії програмного забезпечення

Пояснювальна записка

до магістерської роботи
на ступінь вищої освіти магістр

на тему: «**Розробка системи на базі технологій штучного інтелекту для ігор в жанрі RTS**»

Виконав: студент 6 курсу, групи ПДМ–61
спеціальності
121 Інженерія програмного забезпечення
(шифр і назва спеціальності/спеціалізації)

Борук В.Ю
(прізвище та ініціали)

Керівник Дібрівний О.А.
(прізвище та ініціали)

Рецензент _____
(прізвище та ініціали)

Київ –2022

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра Інженерії програмного забезпечення
Ступінь вищої освіти -«Магістр»
Спеціальність підготовки – 121 «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Інженерії програмного забезпечення
_____Негоденко О.В.
"____" _____2022 року

З А В Д А Н Н Я
НА МАГІСТЕРСЬКУ РОБОТУ СТУДЕНТА

Борука Вадима Юрійовича
(прізвище, ім'я, по батькові)

1. Тема роботи: «Розробка системи на базі технологій штучного інтелекту для ігор в жанрі RTS»
Керівник роботи: Дібрівний О.А.
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом вищого навчального закладу від «11» жовтня 2021 року №170.
2. Строк подання студентом роботи_____
3. Вхідні дані до роботи
Розробка системи на базі технологій штучного інтелекту для ігор в жанрі RTS
Штучний інтелект, нейронні мережі.

4. Зміст розрахунково-пояснювальної записки(перелік питань, які потрібно розробити).
4.1 Тестове середовище для тестування нейронних мереж
4.2 Визначення ефективного алгоритму роботи для штучного інтелекту
4.3 Розробка системи
4.4 Аналіз та тестування розробленого середовища

5. Перелік демонстраційного матеріалу (назва основних слайдів)
1. Актуальність проблеми
2. Аналіз аналогів
3. Принцип роботи системи

4. Обраний алгоритм роботи штучного інтелекту
5. Архітектура бази даних
6. Висновки

6. Дата видачі завдання _____

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської роботи	Строк виконання етапів роботи	Примітка
1	Аналіз теоретичних матеріалів та вивчення предметної області		Виконано
2	Розробка технічного завдання		Виконано
3	Аналіз існуючих рішень та підходів до розробки штучного інтелекту		Виконано
4	Вибір технологій реалізації штучного інтелекту для мікро- та макроуправління		Виконано
5	Розробка структури тестового середовища та серверної частини		Виконано
6	Реалізація, перевірка та налагодження програми		Виконано
7	Оформлення текстових та графічних матеріалів		Виконано
8	Передзахист магістерської дисертації		
9	Подача магістерської дисертації		

Студент_____
(підпис) (прізвище та ініціали)

Керівник роботи_____
(підпис) (прізвище та ініціали)

РЕФЕРАТ

Магістерська дисертація присвячена розробці та опису штучного походження розуму з впровадженням нейронних мереж для гри в жанрі стратегія в реальному часі.

Магістерська дисертація знаходиться на 73 сторінках і включає 41 картинки, 3 таблицю та 30 бібліографічних посилань. Вона зроблена з відповідних розділів: вступ, 5 розділів провідної частини, висновки та перелік посилань.

Актуальність обраної теми полягає в підвищенні ефективності та адаптивності агентів штучного інтелекту для ігор в жанрі стратегій в реальному часі, а також використання розподіленої системи з централізованим сервером як елементу покращення ефективності модулівштучного інтелекту.

Метою роботи є створення системи агентів штучного інтелекту з використанням штучних мереж для забезпечення високої ефективності роботиботів в іграх жанру стратегій в реальному часі .

Об'єктом дослідження є штучний інтелект в іграх жанру стратегій в реальному часі.

Предметом дослідження є побудований з використанням штучних нейронних мереж штучний інтелект для гри в жанрі стратегії в реальному часі, зокрема модулі мікроуправління (тактичний) та макроуправління (стратегічний), з використанням штучних нейронних мереж в комбінації з іншими підходами, а також розподілена система навчання штучного інтелекту з використанням централізованого серверу.

Ключові слова: нейронні мережі, справжній розум, ієрархічна мережа завдань, стратегії реальному часі.

ЗМІСТ

ВСТУП

В наш час ми можемо спостерігати активний розвиток ігрової індустрії. З кожним днем з’являється все більше нових різноманітних проектів, що і підвищує інтерес користувачів. Компанії отримують все більші доходи, які дають змогу все більше покращувати і розвивати свої продукти. Всі ці проекти в яких реалізовано різноманітні цікаві та неможливі здавалося б ідеї, активно привертають увагу нових користувачів. Також постійно завдяки цим неймовірним ідеям засновуються все нові й нові жанри ігор. Певні жанри, що здавалося б давно стали забутими або неприбутковими, відновлюються та повертаються до користувачів в нових інтерпретаціях. Все це можна і сказати про стратегії в реальному часі (RTS). Це досить специфічний жанр, адже він поєднує в собі: складне стратегічне мислення, тактичне вирішення задач та швидке реагування на певні події що відбуваються під час гри.

Для стратегій в реальному часі (RTS) дуже актуальна проблема штучного інтелекту. Бо вся складність управління і прийняття рішень призводить до того, що люди що розроблюють закладають в штучний інтелект певний алгоритм що в подальшому буде виконуватись раз за разом. Як на мене, такий ґеймплей не є занадто цікавим адже гравцям потрібно постійно повторювати однаковий перебіг подій. При такій грі можна з легкістю за певний час вивчити весь алгоритм , наприклад атаки штучного інтелекту або й ходу всієї битви. Таким чином ваша перемога постає лише в питанні часу та певного бажання. Також можна виділити проблему швидкодії штучного інтелекту та рівень складності супротивника.

Актуальність теми що я обрав для опрацювання полягає в підвищенні ефективності штучного інтелекту для стратегій в реальному часі. Також розробка розподіленої системи з одним центральним сервером для більшої ефективності штучного інтелекту.

Мета роботи це створення системи агентів штучного інтелекту з використанням власноруч створених мереж для забезпечення високої ефективності роботи ігрових персонажів для стратегій в реальному часі (RTS).

Об'єктом дослідження був штучний інтелект створений для стратегій в реальному часі (RTS).

Предметом дослідження є штучний інтелект що розроблений з використанням нейронних мереж для ігор в жанрі стратегій в реальному часі (RTS). А також розроблені модулі для мікроконтролю (тактичний) та макроконтролю (стратегічний), для розподіленої системи з одним центральним сервером для більшої ефективності штучного інтелекту.

Наукова новизна заключається у вирішенні задачі, яка пов'язана з розробкою більш ефективного штучного інтелекту для стратегій (RTS). Результати даної роботи можна буде використовувати в подальшому дослідженні штучного інтелекту для будь яких інших жанрів ігор не лише для стратегій. Також можна буде впровадити їх в нові ігрові продукти або використати для покращення вже існуючих.

Для того щоб протестувати штучний інтелект нам необхідно використовувати середовище з певними правилами для гри. Для визначення його ефективності потрібно порівняти його з іншими вже існуючими технологіями. Проте окрім середовища в якому ми будемо тестувати робота, також важливим елементом є серверна частина яка буде керувати ним. Всі ці елементи нам потрібно буде розробити для визначення ефективності штучного інтелекту.

Згідно результатів роботи було опубліковано тези до XIII Міжнародної науково-технічної конференції студентства та молоді «СВІТ ІНФОРМАЦІЇ ТА ТЕЛЕКОМУНІКАЦІЙ», на теми: «Розвиток штучного інтелекту для стратегій в реальному часі» та «Використання нейронної мережі для тактичного штучного інтелекту в іграх жанру стратегії в реальному часі».

1 ОГЛЯД ІНСНУЮЧИХ РІШЕНЬ

В наш час штучний інтелект дуже активно використовується в різних сферах нашого життя, починаючи від медичного застосування закінчуючи будівництвом. Проте не дивлячись на таке розповсюджене використання, найбільш перспективною та найчастіше використовуваною сферою є комп'ютерні ігри. Серед великого різноманіття, особливо виділяється жанр стратегій в реальному часі. Тому що, в цих іграх присутня та складність задач яка відноситься не лише для гравця а й для штучного інтелекту.

Наприклад можна виокремити штучний інтелект для гри в шахи або інші настільні ігри. Його можливості досягають гри рівня людини, і це для нас є звичним явищем. Силу штучного інтелекту ми можемо побачити під час перемоги робота DeepBlue над професіоналом гри в шахи Каспаровим. Таким чином дослідники довели, що штучний інтелект в іграх з покроковим сценарієм може перемогти навіть досвідчених професіоналів. Проте зі стратегіями в реальному часі ситуація кардинально змінюється. На відміну від ігор з покроковим сценарієм, де кожен з гравців роблять свої дії один за одним і мають змогу вирахувати свої подальші дії далеко наперед, в стратегіях (RTS) ситуація змінюється щосекунди, гравці роблять свої дії одночасно. Одним з ключових моментів стратегій в реальному часі (RTS) є таке поняття, як «туман війни». Це поняття означає що так як і в реальному світі або реальній битві супротивники не можуть знати кількості ресурсів одне одного, розташування наприклад укріплень, позицій оборони та ключових споруд. В самій же грі ми не бачимо територію супротивника те буде проходити битва тому і наприклад озброєння або кількість військ нам доведеться обирати всліпу (тобто без додаткових даних). Через все це значною мірою ускладняється аналізування ситуації що відбувається в певний період часу. Один з дослідників стратегій який спеціалізується в основному на штучному інтелекті, Буро Майкл, написав: «Для того щоб зрозуміти всю складність стратегій в реальному часі, вам потрібно уявити собі гру в шахи на дошці розмірами

512 на 512 і великою кількістю ігрових фігур, які постійно рухаються і вам потрібно слідкувати за всім цим і приймати відповідні рішення».

1.1 Використання штучного інтелекту в жанрі стратегій в реальному часі (RTS)

Всі більш менш популярні проекти що підпорядковуються жанру стратегій в реальному часі мають загально прийняті правила які розповсюджуються на весь жанр. Як звичайно в грі на початкових стадіях у кожного гравця доступна лише якась певна частина ігрового світу та один або декілька ботів що виконують певну роботу (вони можуть будувати будівлі або наприклад видобувати ресурси). В стратегіях кожна виконана на початку гри дія може відобразитись на подальшому вашому ґеймплеї. В кожного об'єкту в грі є своя зона відповідальності. Наприклад будівлі, вони є різні по типу, тобто для зберігання ресурсів, технічні, оборонні споруди та інші. Залежачи від стратегії яку обрав гравець він може побудувати будь яку будівлю яка йому буде потрібна в той чи інший момент. Виходячи з цього це його рішення може нанести будь який вплив на подальший розвиток подій. В штучному інтелекті за цю частину відповідає *модуль прийняття стратегічних рішень,* також його називають *макроконтроль*.

В процесі гри коли надається можливість створювати власну армію,, гравцю відповідно потрібно буде будувати споруди в яких буде можливість наймати бойові одиниці до своєї армії. Вся армія також як зазначалося раніше поділяється на різноманітні типи (різне озброєння, різні характеристики ботів, різне призначення). Кожен тип потрібно використовувати згідно своїх можливостей та характерних рис. Наприклад, воїни що призначені для прориву повинні триматися на передовій, а групи підтримки повинні ними прикриватися. За керування всіма цими ботами (*мікроконтроль*) під час ведення війни, штучний інтелект, використовує *модуль прийняття тактичних рішень.*

На початку гри користувачам доступно лише невелика частина ігрового світу – велику її частину гравець розвідуватиме згодом під час гри. Для цього йому знадобиться *модуль розвідки* . Він також буде корисний під час підготовки до війни. Для того щоб визначитись із стратегією та планами для успішного захоплення території.

Окремо ще можна додати що окрім модулів штучного інтелекту що були згадані вище існує також система що проводить навчання (адаптацію) персонажів що підпорядковуються штучному інтелекту, для адекватної реакції на ту чи іншу ситуацію. Дуже часто розробники закладають цю систему у *модуль розвідки та прийняття рішень*.

1.2 Прийняття тактичних рішень

Для того щоб створити штучний інтелект з тактичними можливостями використовуються різні підходи. Переглянемо їх перелік:

- Баєсова модель;
- Нейронні мережі;
- Пошук по ігровому дереву;
- Навчання з підкріпленням;
- Рішення на основі прецедентів.

Баєсова модель представляє собою набір випадкових змінних та залежностей між ними. Залежність досягаються за допомогою орієнтованого ациклічного графу. За допомогою цієї моделі також можна визначати напрямки руху ботів та приймати тактичні рішення.

Нейронні мережі . Для цього методу використовуються алгоритми що навчають нейронні мережі. Називається rtNEAT. Може бути використаний для контролю кожного окремого бота. Кожен бот отримує дані зібрані з навколишнього середовища та

від оточуючих його об'єктів. Кожен об'єкт штучного інтелекту що впідпорядковується нейронним мережам визначає свою ефективність завдяки функції. Значення якої напряму залижить від значень здоров'я, його ігрового рівня та місцезнаходження. Боти що мають слабкі характеристики можуть змінюватись більш продуктивними юнітами. У всіх тестуваннях що проводились з використанням даної мережі, найкращий результат можна отримати лише коли створені ідеальні умови для штучного інтелекту (коли величина війська однакова з обох сторін). Проте для прийняття тактичних рішень використання нейронних мереж досить розповсюджене. Наприклад в свій час дослідники CNN за допомогою нейронної мережі досягли значно кращих результатів а ніж за допомогою інших алгоритмів. Навіть попри те що нейронні мережі працюють набагато довще ніж сучасні алгоритми, вони показують набагато кращі результати. Основуючись на все це дослідники штучного інтелекту вважають що якщо об'єднати разом нейронні мережі та алгоритми то можна створити непереможного супротивника для людей гравців в стратегіях в реальному часі (RTS).

Пошук по ігровому дереву є алгоритмом, що не часто використовується для прийняття стратегічних рішень, але для тактичних рішень він показує набато кращі результати. Одного разу дослідники створили систему що могла аналізувати всі відомі стратегії в реальному часі. Дана система могла визначити рівновагу при якій користувач, тобто гравець не може отримати більше ніяких переваг а також наприклад збільшити виграш змінивши своє стратегічне рішення. Для досліду обрали лише тактичні рішення а також лише мікроконтроль кожного бота. Управління велося групами і полягало лише в переміщенні юнітів по ігровому світу. Як результат перемогу отримала дана система а не штучний інтелект самої гри. Перемога полягала в тому що система використовувала набагато ефективніші стратегічні рішення.

Ще одним прикладом можна назвати гру SparCraft. Це аналог всім добре відомої гри StarCraft, але трохи спрощений. Чому аналог ? Тому що як і більшість комерційних продуктів StarCraft має закритий вихідний код. Тому дослідникам і довелось створити подібний продукт але вже з відкритим вихідним кодом. Ігровий процес був

повністю ідентичний. Тести що проводили над штучним інтелектом показали що алгоритм в 92% випадках працює ефективніше для стратегій в реальному часі.

Навчання з підкріпленнями Так названа система машинного навання де штучний інтелект повинен обирати свої кроки так щоб в подальшому вони привели його до максимального значення умовної винагороди. На відміну від роботи штучного інтелекту по заданому плану, тут немає правильних та неправильних дій, і неправильні або некоректні ді штучного інтелекту не можна виправити. Тобто в подальшому система буде працювати з вже існуючим набором дій виконаних при попередньому виконанні.

За приклад візьмемо ту ж саму гру StarCraft. Для неї біло створено модуль для прийняття тактичних рішень на основі нейронної мережі та алгоритму навчання з підкріпленням. За допомогою цієї моделі реалізували управління для кожного з воїнів. Для управління маленькими групами алгоритм показав дуже ефективним в порівнянні з вбудованими сценаріями для звичайного штучного інтелекту. Даний алгоритм ми можемо побачити на (рис.1.1)

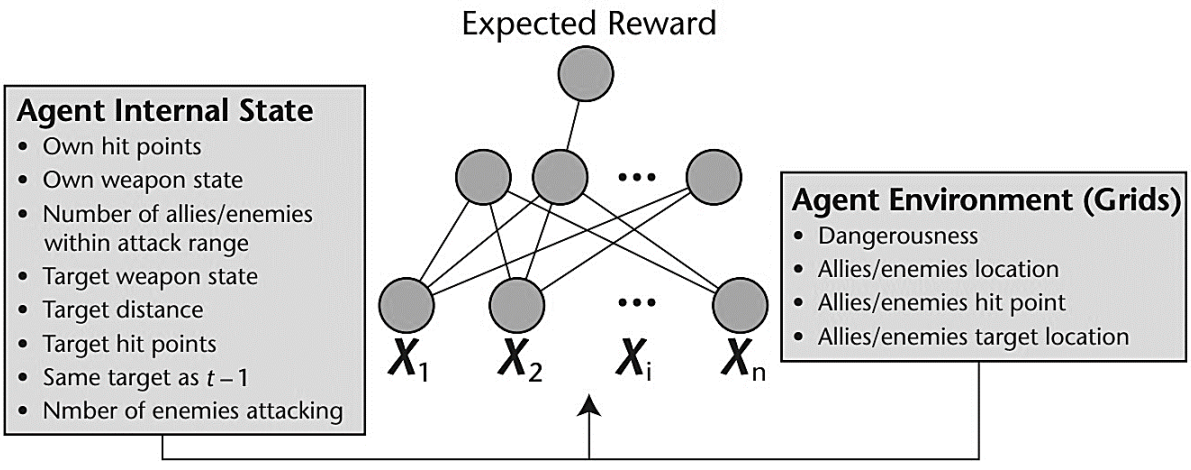


Рисунок 1.1 – Мережа для визначення очікуваної умовної нагороди для окремого бота.

Рішення на основі прецедентів використовується в стратегіях в реальному часі, де тактичні рішення важливіші аніж стратегічне планування. Тобто мікроконтроль понад усе. Простіше кажучи рішення на основі прецедентів – це вибір вже серед існуючих (тобто прописаних для штучного інтелекту) рішень. Візьмемо за приклад гру WarCraft. В ній використовується модуль управління рішеннями на основі прецедентів. Показує він себе досить ефективно в мікроконтролі під час битв, що досить ефективно допомагає гравцеві під час ведення масштабних битв. Тобто гравець надає команду певній групі військ і та діє по своєму заданому алгоритму. Дослідники за допомогою своїх досліджень змогли довести що рішення на основі прецедентів можна без проблем поєднувати з навчанням з підкріпленням . Все це надає досить ефективний результат в поєднанні одне з одним.

1.3 Прийняття стратегічних рішень

Стратегічні рішення – це частина штучного інтелекту, що працюють на висому рівні в стратегіях і надають значний вплив на гру протягом довго часу гри. Також стратегічні рішення називають макроменеджментом. Наприклад гравець побудував якусь оборонну споруду на початку гри. Розташування даного об’єкту може відіграти важливу роль наприкінці гри. Для того щоб визначити ефективні стратегічні дії використовуються планувальні системи. Ці системи показали досить ефективну роботу в ігрових продуктах. Це показали навіть дослідження які були проведені для дослідження ефективності. Вся проблема може бути лише в рішеннях супротивника. Бо ви вимушені будете змінювати свої рішення відносно його рішень.

Тут вступає в гру поняття «туман війни». Як було описано раніше ми не можемо знати нічого про нашого противника. Через це всі рішення ми повинні приймати не керуючись ніякими даними . Проте інколи ми можемо мати певну інформацію, проте не повну. В цьому випадку основними підходами для планування на основі прецедентів. Дуже рідко використовують дерева поведінки та нейронні мережі.

Зазвичай штучним інтелектом для ігор являється певний набір варіантів для досягнення цілей.

Прийняття рішень на основі прецедентів в макроменеджменті подібний до тактичного модулю. При цьому методі розробники повинні створити певний набір варіантів для певної ситуації, а штучний інтелект в свою чергу повинен знайти найбільш підходящі дії для досягнення найкращого результату. Для такого підходу не потрібно постійно навчати штучний інтелект. Він являється адаптивним до нових стратегій супротивника (гравця).

Даний штучний інтелект для прийняття рішень макроменеджменту був створений у 2005 році. Плюсом його була перш за все адаптивність до різноманітних стратегій гравця. На відміну від інших алгоритмів що показували гарні результати лише проти супротивника який діяв по чітко прописаному плані.

Ієрархічне планування – розподіл цілей для гри в сітку задач. Там можуть бути присутні як і абсолютні задачі так і мізерні задачі. До високих задач можна віднести задачі такі як наприклад перемога над противником. До мізерних задач можна віднести задачі так як: збір ресурсів, пересування з однієї точки в іншу. Ієрархічне планування наприклад можна поєднати з рішеннями на основі прецедентів. Всі ці задачі виконуються послідовно.

Ієрархічне планування було використано у відомій стратегічній грі Spring. За допомогою цієї системи можна було ефективно збирати ресурси.

Також можна виділити що такий підхід можна використовувати при створенні послідовних та паралельних дій які виконуватимуться одночасно. Всі ці дії виконуються лише від залежності стану в якому знадить юніт в даний момент.

Автономне досягнення цілей це виконання послідовності дій що можуть змінюватись в залежності від ситуації. При виникненні нових даних штучний інтелект може змінити послідовність варіантів для досягнення заданої цілі. Ця система допомагає швидко реагувати штучному інтелекту на певні зміни.

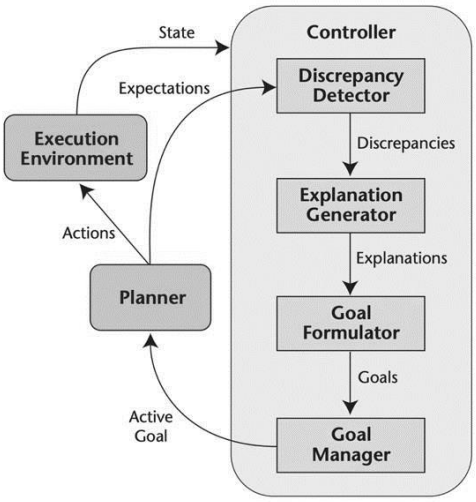


Рисунок 1.2 – Модель автономного досягнення цілей

1.4 Розвідування і визначення планів

Для стратегічного планування як окрему систему можна виокремити модель визначення планів та стратегії контр дій. Через «туман війни» плани заключаються на неповній інформації тому часто використовуються вже готові техніки:

- Дедуктивні;
- Абдуктивні;
- Ймовірні;
- Прецедентні.

Плани визначені *дедуктивним методом* працюють на визначені ймовірної ситуації на основі поточних подій . За допомогою такої техніки ми маємо змогу визначити плани противника з невеликої кількості інформації. В грі StarCraft також був присутній даний метод. Він допомагав штучному інтелекту адаптуватися до певного вибору війська або дій гравця.

Абдуктивні методи заключаються в тому що з набору певних даних ми можемо зробити певні висновки. Основуючись на ці висновки створюється певна ціль і план її

досягнення. Цей метод потребує постійної підтримки технології автоматичного досягнення цілей.

Ймовірні методи. Для них використовується певна статистика що складається з оцінки дій які відбувалися в певні проміжки часу. Для цієї техніки щоб збирати всі ці статистичні дані використовується велика кількість записів ігор реальних гравців.

Ця техніка співпрацює із статистикою що представлена набором певних даних про бота і в основному вона використовується на початкових етапах гри. При дослідженні Цю технологію використовували разом з Баєсовою моделлю. Такий зв'язок допоміг врахувати під час планування подальших дій, потенційно не розвідані території та не розвідану базу супротивника.

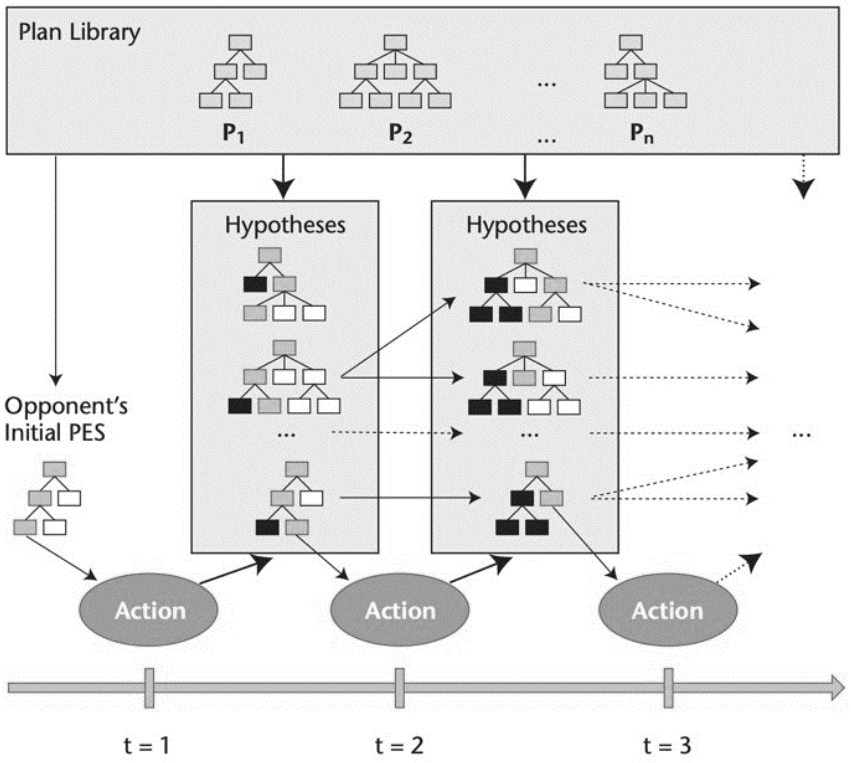


Рисунок 1.3 – UML діаграма роботи абдуктивних методів

Прецедентні техніки дозволяють отримати дані на про ймовірні дії противника через використання вже збережених випадків та ймовірних подій. Цю технологію

ефективно використовувати на початку гри. Бо саме початкове визначення тактичних та стратегічних рішень є основним в грі. Наприклад для найбільш популярих стратегій вже є багато різноманітних рішень початку гри. Вони допомагають вам обрати певну послідовність дій які в подальшому допоможуть вам досягти успіху в певній сфері. Наприклад план дій за допомогою яких можна при мінімальних затратах на побічні задачі створити найбільш ефективну армію, що в подальшому допоможе захопити велику територію.

1.5 Висновки до розділу

Виходячи з вище написаного ми зрозуміли що штучний інтелект для стратегій в реальному часі складається з декількох методів: стратегічного керування, тактичного керування та модуля визначення планів для подальших дій.

Через складність задач які постають перед штучним інтелектом немає одного алгоритму який ефективно буде працювати та приймати всі рішення.

2 ПРОЕКТУВАННЯ ШТУЧНОГО ІНТЕЛЕКТУ

2.1 Підходи та алгоритми які я використовуватиму

При підготовці до розробки самого штучного інтелекту мені довелося проаналізувати велику кількість матеріалів та досліджень що стосувались різноманітних підходів для створення штучного інтелекту. Для того щоб розробити штучний інтелект для ігор в жанрі стратегії в реальному часі (RTS) мені потрібно буде використати комбінацію що буде складатись декількох підходів. За допомогою цих підходів я буду реалізовувати різні частини штучного інтелекту.

Для розробки мікроконтролю я використовуватиму технологію нейронних мереж. Це допоможе штучному інтелекту реагувати найбільш коректно майже у всіх ситуаціях .

Багаторівневий персептрон – один із варіантів побудови штучного інтелекту. В подальшому ми використовуватимемо штучні нейронні мережі.

Штучні нейронні мережі – реалізована програмно нейронна структура, точнісінько як і в нашому мозку. Нейронні мережі можуть міняти типи отриманих сигналів не в залежності від їх виду. Види сигналів поділяються на електричні або хімічні . В людини в мозку нейрони дуже сильно пов’язані між собою і складають величезну систему нейронів, де один нейрон свій отриманий сигнал може передати тисячам іншим нейронам. Навчання штучного інтелекту відбувається за допомогою багаторазовою активацією певних нейронних з’єднань. Таким чином при виконанні певної задачі в нас є велика ймовірність виникнення правильного результату. Через те що вони отримали певний правильний сигнал який штучний інтелект навчений приймати. Таке навчання використовує зворотній зв’язок навчання. Таким чином нейронні з’вязки постійно наповнюються і таким чином ущільняються.

Тобто нейронні мережі повністю повторюють людський мозок але в більш спрощеному вигляді. Проблемою є те що вони можуть бути навчені як контрольовано так

і неконтрольовано. Тобто коли мережа навчатиметься розробником спеціально для виконання певної задачі, це добре. Але ж вона може взяти для себе певну інформацію яка потрапила до мережі не спеціально а випадково з непідконтрольної сторони . Тобто взяла для себе якусь зайву інформацію.

При навчанні штучні нейронні мережі вводяться два види інформації. Тобто Саме завдання та результат який повинен бути в кінці. Таким чином мережа навчається і в подальшому при виконанні певної задачі в нас вийде правильний результат.

Для прикладу навчання нейронної мережі можна обрати вашу електронну пошту, а саме нейронну мережу що фільтрує спам. Так само як і зазначено вище нейронний зв'язок отримує два типи інформації. Вхідна інформація завдяки якій буде визначатись чи підлягає блокуванню спам наше повідомлення чи ні. Та обов'язково вихідна це вже та інформація яка і визначить тип нашого повідомлення (спам чи ні). Таким чином поступово з кожним зверненням до спаму з певними повідомленнями тобто з вхідною інформацією наша нейронна мережа буде все більше наповнюватися.

Тепер для того щоб щось виконати за допомогою нейронної мережі нам потрібно його «ввімкнути». Активується він за допомогою функції. Зазвичай для цього використовують сигмоїдну функцію:

$$f(x)\frac{1}{1+\exp(-z)} \tag{2.1}$$

Графік цієї функції зображений нижче на (рисунку 2.1).

По даному графіку можна побачити що функція зростаюча, тобто вона активує нашу нейронну мережу. Сигмоїдна функція неперервна, вона зростає від 0 до 1 при кожному збільшені значення x. Це все також означає що дана функція також має і похідну, що є невід'ємним аргументом для того щоб навчати нашу нейронну мережу.

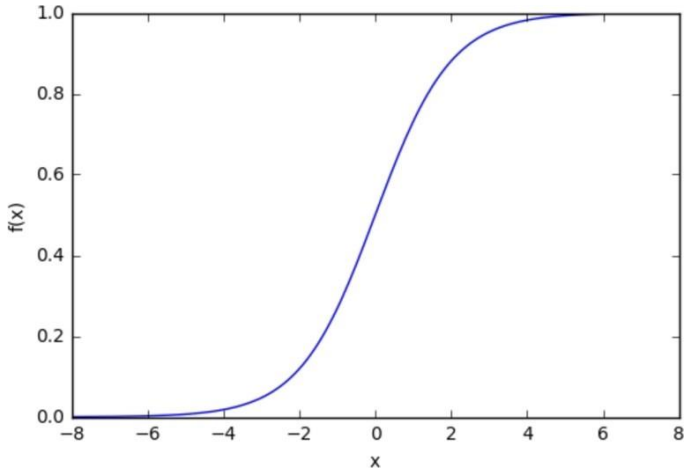


Рисунок 2.1 – Графік сигмоїдної функції

Як вже було написано до цього, звичайні нейрони з мозку людини з’єднані між собою певними мережами. В цих мережах вихід одних нейронів зв’язаний з іншими нейронами. Тобто вони можуть не тільки виходити з цього виходу а й заходити через нього.

Таку мережу можна представити за допомогою намиста. Кожна намистина має вхід через який проходить нитка.

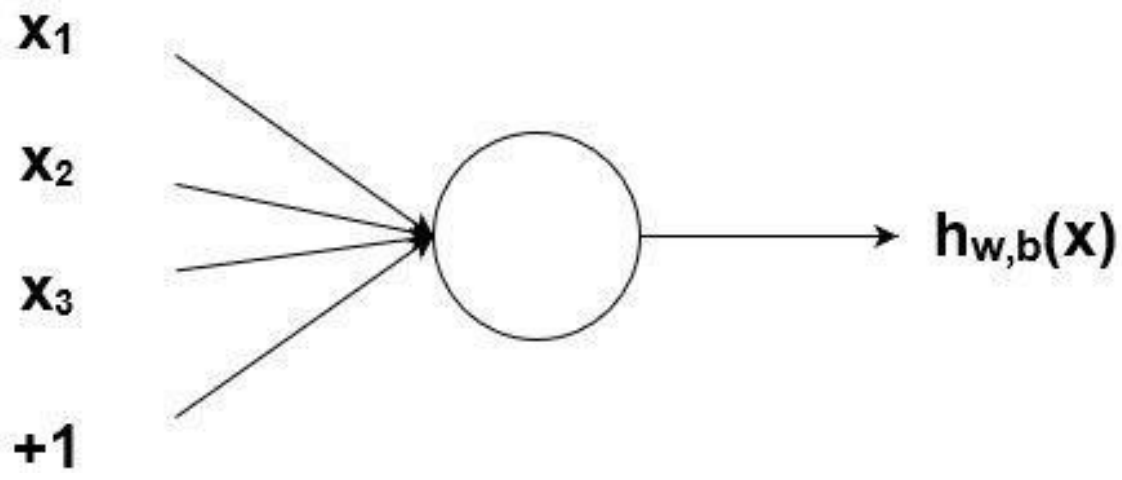


Рисунок 2.2 – Зображення штучного нейрону

В самій намистині відбувається активація функції, та в кінці кінців виходить через вихід, таким чином генерується вихід.

Коло на даній схемі зображує певний вузол. Вузол це так зване місце де знаходиться наша сигмоїдна функція, що і активує певний процес. Даний вузол приймає в себе декілька сигналів, об'єднує їх в один напрям, а потім активує саму функцію. Дана функція представлена за допомогою функції h. Вузол також можуть називати перцептроном. Для ваги беруться небінарні числа. Ці числа в подальшому помножуються на початку і вже на самому виході додаються. Все це відбувається у так званому вузлі, який на схематичному зображені позначається колом.

$$w_1 x_1 + w_2 x_2 + x_3 w_3 + b \tag{2.2}$$

Ось такий вигляд має у вузла зважений вхід. Де wn – числове значення ваги. Вага вузла це значення яке буд змінюватися протягом всього навчання нейронної мережі. b – вага переміщення елементу на 1. Нам обов'язково потрібно включати цю змінну, адже таким чином наш вузол стає гнучкішим.

До цього ми розглядали як працює окремий елемент працює. Тобто як працює один нейрон або вузол. Проте Повне нейронна мережа складається з багатьох пов'язаних між собою вузлів. Такі мережі можуть мати різноманітні форми та вигляди. Не дивлячись на все це найбільш поширеною є вигляд що складається з: вхідних сигналів, прихованого елементу та елементу виходу. Приклад такого елементу ми можемо побачити на схемі нижче.

На зображеній схемі ми бачимо нейронну мережу що складається з трьох шарів:

- Перший, вхідний. Тут мережа в себе вхідні зовнішні сигнали (дані);
- Другий, прихований. Ця частина нейронної мережі не являється вхідною і вихідною. Цих шарів може бути декілька;
- Третій, вихідний.

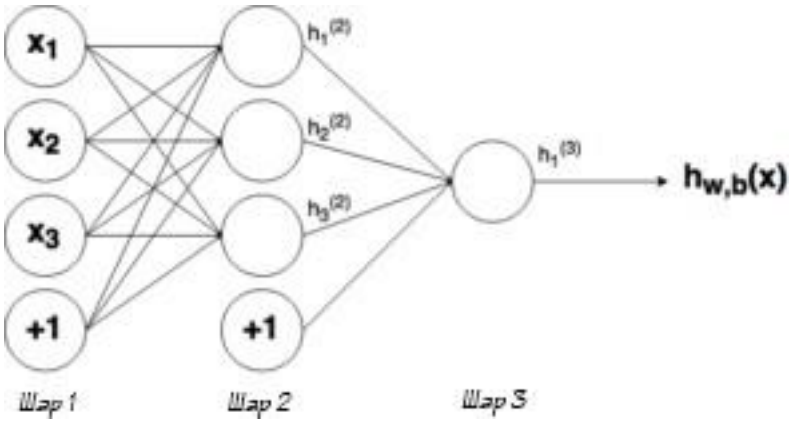


Рисунок 2.3 – Трирівнева штучна нейронна мережа

Також ми можемо побачити великі кількість зв'язків між всіма цими частинами. А особливо між першою та другою. Можна сказати що перший та другий шари згруповують дані і надають їх вже до останнього шару який видає лише один результат. Всі ці зв'язки мають свою вагу.

Наприклад, для того щоб показати роботу прямолінійного поширення для нейронних мереж ми можемо взяти ту ж саму схему що зображена вище . З трьома частинами. Ми використаємо систему рівнянь для демонстрації:

$$h^{(2)} = f(w^{(1)}x_1 + w^{(1)}x_2 + w^{(1)}x_3 + b^{(1)}) \quad (2.3)$$

$$h^{(2)} = f(w^{(1)}x_1 + w^{(1)}x_2 + w^{(1)}x_3 + b^{(1)}) \quad (2.4)$$

$$h^{(2)} = f(w^{(1)}x_1 + w^{(1)}x_2 + w^{(1)}x_3 + b^{(1)}) \quad (2.5)$$

$$h^{(3)} = f(w^{(2)}h^{(2)} + w^{(2)}h^{(2)} + w^{(2)}h^{(2)} + b^{(2)}) \quad (2.6)$$

В результаті останнього рядка ми отримуємо вихід, як показано на схематичному зображенні. Схематичне зображення ми розглядали раніше. Результат також можна назвати результатом нейронної мережі. Для виконання використовуються різні дані, такі як ваги виходів вузлів що знаходяться в другій частині та заміщення. Завдяки

даній системі рівнянь ми можемо наглядно зрозуміти та побачити ієрархічну структуру нейронної мережі.

Далі ми можемо підставити значення до системи. Значеннння ваги та заміщення відповідно. Давайте розрахуємо результат.

$$h_1^{(2)} = f(0.2 \times 1.5 + 0.2 \times 2.0 + 0.2 \times 3.0 + 0.8) = 0.8909 \quad (2.7)$$

$$h_2^{(2)} = f(0.4 \times 1.5 + 0.4 \times 2.0 + 0.4 \times 3.0 + 0.8) = 0.9677 \quad (2.8)$$

$$h_3^{(2)} = f(0.6 \times 1.5 + 0.6 \times 2.0 + 0.6 \times 3.0 + 0.8) = 0.9909 \quad (2.9)$$

$$h_1^{(3)} = f(0.5 \times 0.8909 + 0.5 \times 0.9677 + 0.5 \times 0.9909 + 0.2) = 0.8354 \quad (2.10)$$

Таким чином при визначенні значень ваги вузлів що з’єднують частини нейронних мереж, ми і отримуємо те так назване навчання нейронної мережі. Тобто не отримуємо а це воно і є. Суттю спеціалізованого навчання є те що на потрібно зменшити невідповідність між входом до системи та виходом із неї.

Наприклад в нас є певна нейронна мережа з одним виходом та певний вхід. В результаті навчання ми хочемо щоб наша нейронна мережа показала значення 3. Але через певні обставини мережа видає 7. То нам потрібно знайти розбіжність точніше похибку. Для того щоб знайти похибку нам потрібно від бажаного результату 3 відняти отриманий 7. В результаті ми отримаємо 4. На математичній мові це значення нам дає розуміння того що ціна помилки в нас дорівнює 4.

Контрольоване навчання – навчання нейронної мережі коли всі дані вносяться до системи вручну. Для ефективності потрібно постійно змінювати значення ваги вузлів що зв’язують різні часини нейронної мережі, для того щоб зменшити помилку.

Всі значення що стосуються значень входу та виходу векторні. Тобто значення можуть постійно змінюватись, а саме збільшуватись та зменшуватись відповідно. І для кожного з них може бути велика кількість різних значень одночасно.

2.2 Різновиди наказів та подій відносно

Стратегії в реальному часі можуть похвалитись своїм неперевершеним а й одночасно дуже складним ґеймплеєм. Для стратегічних ігор навіть коли гравець один, може використовуватись велика кількість ігрових об’єктів, роботів та штучних інтелектів. Щосекунди між гравцями або гравцем та апаратною частиною гри відбувається обмін даними. Ці дані постійно оновлюються для підтримки ігрового процесу. Всі ці дані несуть в собі невід’ємний вклад і в нейронні мережі . За допомогою них нейронні мережі оновлюються та постійно навчаються. Простіше кажучи все що відбувається в програмній частині гри і те що не бачить сам гравець можна розділити на декілька різновидів.

Перший, накази. Це основна частина керування не тільки в стратегічних іграх а й у всіх інших. Наприклад якщо розглядати стратегії в реальному часі до наказів можна віднести: атаку, переміщення, побудова будівель. Ці накази можуть роздавати не тільки гравці а й штучний інтелект.

Другий, події. Це різновид який охоплює в себе все що виконується не завдяки наказам гравця або штучного інтелекту. Наприклад знищилася певна оборонна споруда, це є досить важливою подією яка в майбутньому може нанести чимало шкоди всьому ігровому процесу.

Третій – синхронізовані події. Наприклад бот що стояв в обороні замітив ворожого бота і почав атакувати.

2.3 Мікроконтроль та його штучний інтелект

Для того щоб створити штучний інтелект я використовуватиму нейронні мережі. Проте для коректної роботи мені потрібно буде нейронні мережі трохи модернізувати. Адже ми отримуватиму дані які трохи не типові для використання них нейронних мережах.

По-перше ми використовуватимемо дані що будуть братися з локації на якій зараз знаходиться гравець . Тобто ми вокористовуватимемо те що бачить гравець. Даними можуть бути дані про місцезнаходження про війська ворога та оборонні споруди . Для того щоб провести певний аналіз даних які ми отримали їх обов'язково потрібно поділити.

Для зручності обробки цих даним нам потрібно наприклад область яку ми отримали розбити на невеликі зони. Накази які ми віддаватимемо або які буде віддавати штучний інтелект будуть діяти саме по цим зонам .

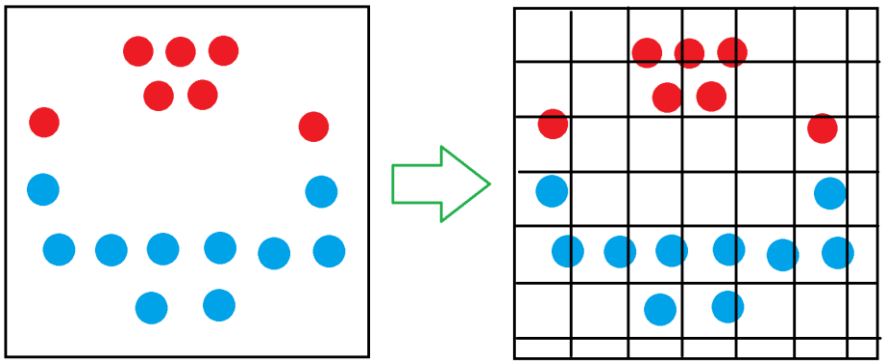


Рисунок 2.4 – Розділення локації на зони

Наступним кроком після поділу на зони нам потрібно кожній зоні надати порядковий номер. Кожна ця зона повинна мати свій порядковий номер.

Після всієї цієї обробки ми вже можемо починати аналізувати ту ситуацію яка в нас вимальовується. Важливим фактором є те що штучний інтелект виконуватиме накази або реагуватиме на них лише після того як він виконає попередню задачу. Ще одним важливим фактором є те що керувати певними групами військ ми будемо лише у віддаленості в одну клітинку від клітинки де знаходиться ця група військ.

Підсумовуючи все те що написано раніше можна виділити певну досить чітку мету роботи для штучного інтелекту. Вона полягає в тому що нам потрібно досягти максимальної ефективності елементів якими керуватиме штучний інтелект які знаходяться в зоні його дії.

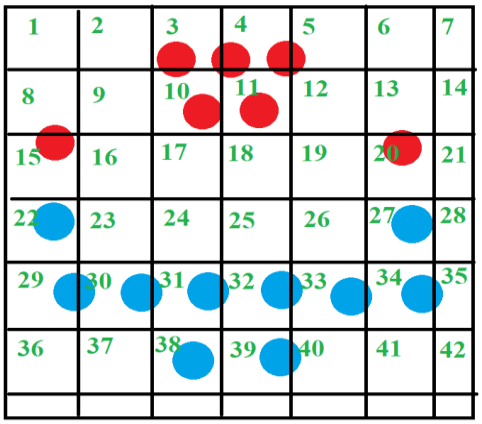


Рисунок 2.5 – Надання кожній зоні свого порядкового номеру

Важливим фактором є те що штучний інтелект виконуватиме накази або реагуватиме на них лише після того як він виконає попередню задачу. Ще одним важливим фактором є те що керувати певними групами військ.

Підсумовуючи все те що написано раніше можна виділити певну досить чітку мету роботи для штучного інтелекту. Вона полягає в тому що нам потрібно досягти максимальної ефективності елементів якими керуватиме штучний інтелект які знаходяться в зоні його дії. Також для аналізу ефективності для штучного інтелекту я використовуватиму систему що буде мати змогу вираховувати втрати ботів , як і власних так і противника. Основним значенням використовуватиметься вага або вартість кожного бота. Тобто яку він несе користь. Це значення буде отримуватись з кожного бота за допомогою функції.

Для того щоб прийняти рішення про подальші дії в рамках обраної зони ми проводимо по два дослідження (аналіз самої зони та аналіз ситуації в якій на даний момент знаходяться елементи якими ми керуємо).

Таким чином можна сказати що для керування ми маємо два типи даних. Перший це те що ми суто перед нашими ботами та те що ми вже маємо аналіз про певну кількість ворожих військ та їх силу.

Також важливим поняттям є тещо в нас в різних видів ботів якими керуємо є різна швидкість руху. То ж для тестування у нашому середовищі ми зробимо швидкість руху однаковою для всіх видів воїнів.

Якщо зібрати всі ці дані до купи то людина може вже прийняти правильне рішення як і коли йому краще діяти .Тож наша задача полягає в тому, що нам потрібно зробити так щоб наша нейронна мережа також при отриманні відповідних даних реагувала так само. Адже якщо поділяти локацію на зони керування то для нейронної мережі потрібно прийняти лише одне правильне рішення з дев’яти.

Також для точнішого аналізу нам потрібно ввести додаткові зони. То му що для аналізу потрібно щоб у кожної зони були сусідні зони для того щоб можна було розробляти різні варіанти.

Тож як ми бачимо після того як ми вже отримали аналіз зон тобто локації на якій ми знаходимося. Всі проаналізовані дані передаються до нашої нейронної мережі. Цей перехід ми можемо спостерігати на рисунку 2.6. Тут зображено два етапи аналізу для певної локації та подальший перехід проаналізованих даних до нейронної мережі.

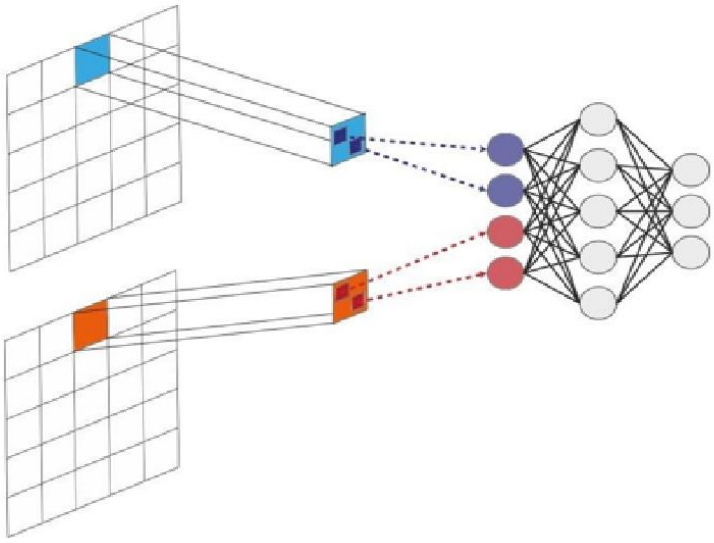


Рисунок 2.6 – Система що об’єднує проаналізовані ані з нейронною мережею

2.4 Стратегічний штучний інтелект

Для того щоб створити систему глобального керування або ж макроконтролю для ігор використовуються різні методи. Як писалося раніше часто використовуються підходи під час яких ми повинні розділити певні задачі на певну послідовність дій які в подальшому будуть виконуватись.

Послідовне виконання певних дій для досягнення певного результату це один з найпростіших варіантів. Тому лише його ми не можемо використовувати. Адже такий підхід найкраще підходить на початку гри коли гравець або штучний інтелект не має великої кількості другорядних задач.

Для стратегій в реальному часі початок гри має невід'ємний вплив на подальшу гру. Адже на початку гри гравці обирають свій подальший шлях розвитку. Всі шляхи розвитку можна поділити на три етапи:

- Економічний;
- Військовий;
- Технічний.

Всі ці варіанти мають певний порядок дій для досягнення найкращих результатів.

Завдяки такому варіанту розвитку можна наприклад якщо використовувати військовий розвиток , розпочати воювати можна майже одразу після освоєння азів гри. Адже після цього у гравця вже буде певна ресурсна база для підтримки та розвитку бойових одиниць.

Також для економічного варіанту при використанні даної моделі ми можемо наростити певну кількість ресурсів для розростання своєї бази для того щоб ворогу було складніше вас завоювати.

Якщо ми обрали технічний варіант розвитку і використовуємо дану модель то ми розвиваємо і економічну складову та військову одночасно або майже порівну.

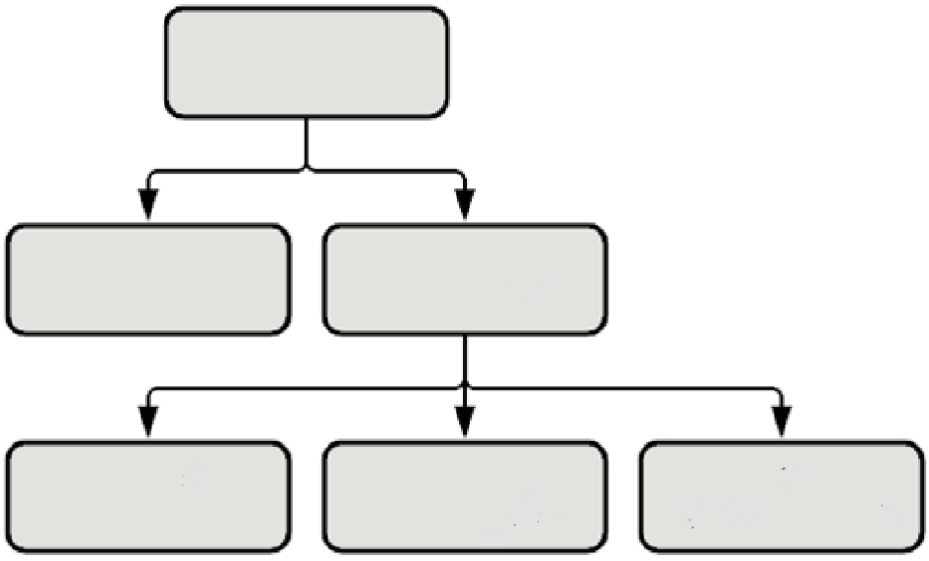


Рисунок 2.7 – Структура розвитку на початку гри для будь якого з етапів

Всі ці варіанти розвитку мають як і свої плюси так і мінуси. Для економічного варіанту основним недоліком є те що в нас присутня лише невелика кількість воїнів та відсутність оборони відповідно. Військовий варіанти основним недоліком вважає те що перспектива може бути не довгою. Адже при поразках нам потрібно використати велику кількість ресурсів а так як економічна складова в нас не розвинута ми не можемо нічого відновити.

Для того щоб адаптувати всі початкові етапи розвитку для ефективного розвитку в подальшому можна використати мережу задач . Точніше кажучи це ієрархічна мережа задач, схематично вона зображена на рисунку 2.8.

Така система складається наступним чином. Перше це ми отримуємо нашу ціль яку ми повинні виконати. Наприклад перемогти в битві.

Наступним етапом є розділення основної задачі на декілька менших. Для кожного з етапів розвитку на початку гри всі ці задачі можуть відрізнятися. Адже ми можемо захищатися або нападати. Тож задачі і змінюються відповідно.

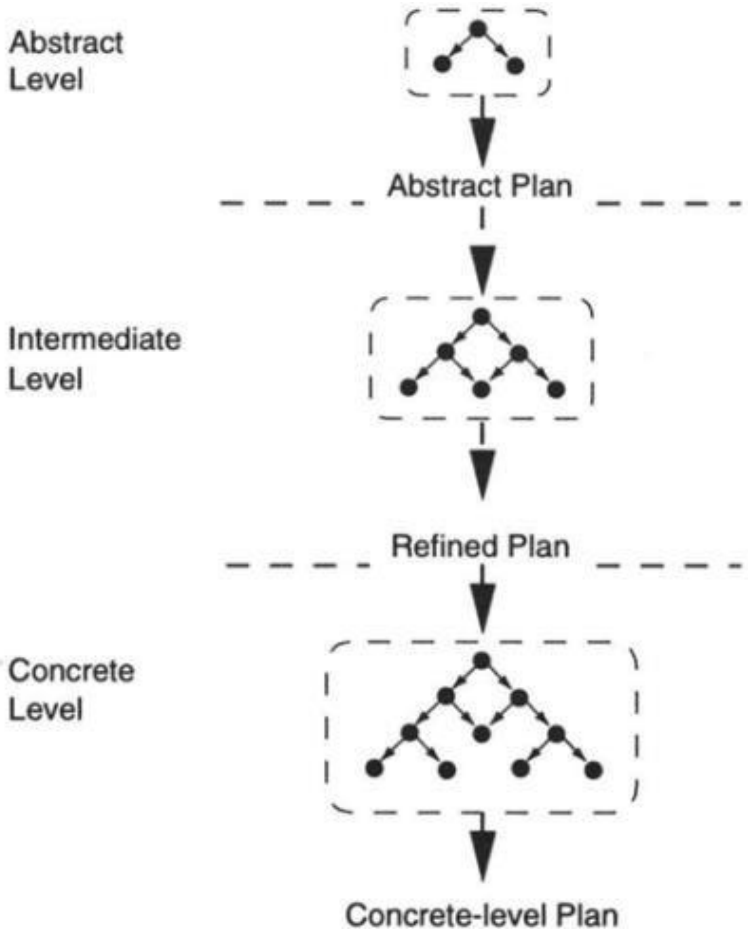


Рисунок 2.8 – Ієрархічна мережа задач

Така система складається наступним чином. Перше це ми отримуємо нашу ціль яку ми повинні виконати. Наприклад перемогти в битві.

Наступним етапом є розділення основної задачі на декілька менших. Для кожного з етапів розвитку на початку гри всі ці задачі можуть відрізнятися. Адже ми можемо захищатися або нападати. Тож задачі і змінюються відповідно.

Останнім етапом є постановка кінцевих завдань які потрібно виконати. Наприклад якщо ми розглядатимемо основну ціль перемогти то ми повинні виконати ряд наступних задач :

- Провести розвідку певними ботами що для цього пристосовані;

- Захоплення менш захищених територій;
- Якщо не вдається захопити території то потрібно збільшити кількість військ;
- Якщо не вистачає ресурсів для збільшення війська то потрібно покращити економічну складову;
- Якщо у противника є значна перевага у розвитку потрібно підвищити свій рівень розвитку технологій.

Всі ці задачі не є кінцевими. Кожна з них поділяється на безліч менші, які в свою чергу можна розділити на ще менші. І ці найменші задачі можна вже назвати наказами які ми роздаємо нашим юнітам.

Таким чином ми можемо побачити що використання ієрархічної мережі задач є досить ефективним. Бо ми найбільшу найголовнішу задачу розбиваємо на безліч інших менш складних. Така модель виконання задач використовується у різних сферах не тільки при створенні стратегій у реальному часі а й у повсякденному житті. Ця технологія може похвалитись високою результативністю в порівнянні з іншими методами.

Наприклад як видно на рисунку 2.8, перед системою ставиться завдання транспортувати один або кілька товарів з однієї точки до іншої. Постановка кожного із завдань розділена на набір умовних рівнів, які відповідають за різні елементи задачі:

- Перевірка чи виконується певне завдання;
- Розподіл на більш простіші задачі;
- Перетворення маленьких задач на накази.

Як ми можемо побачити з рисунка наведеного нижче, ієрархічна мережа задач використовується навіть у виконанні такої простої задачі як доставка двох посилок з однієї точки в іншу.

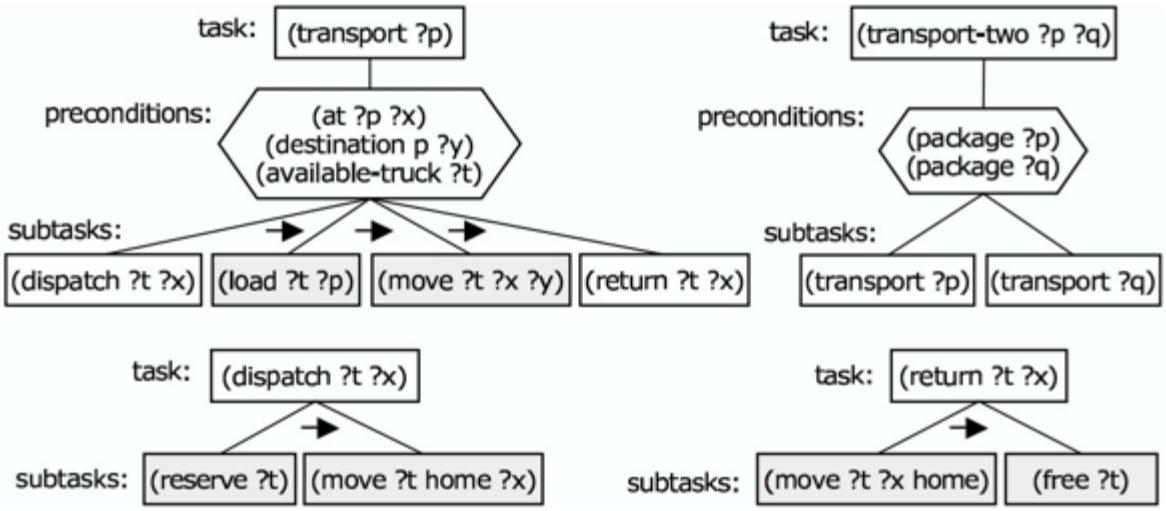


Рисунок 2.8 – Розділення задачі переміщення товарів на конкретні кроки (p, q – пакунки для доставки; t – транспорт; x, y – точки транспортування)

1. Завдання перевіряється на актуальність, у разі завдання доставки двох посилок перевіряється наявність товару у першій точці;

2. Залежно від тесту завдання ділиться на дві менші задачі, кожна з яких є завданням переміщення однієї посилки;

3. Кожна з невеликих задач по переміщенню однієї посилки є типовою, тому ми розглянемо переміщення першої посилки. Слід зазначити, що зараз логістична система враховує правило переміщення однієї посилки одним транспортом. Тобто під однією посилкою у цій системі розробники же знають проаналізувавши дані кількість чи обсяг товару, що повністю завантажує транспортний засіб. В інших системах для логістики можлива складніша структура, коли для переміщення потрібна певна кількість товару, а також важлива логістична складова така як маршрут та кількість транспортних засобів залежить від завдання. У цьому випадку перед відправкою формується завдання об'єднання товарів у мінімально можливу кількість упаковок для більш ефективного руху;

4. Під час переміщення товару на склад потрібний не зайнятий транспорт. Тому перше завдання перед переміщенням товарів буде знаходження транспорту що не задіяний;

5. Після того, як ми знайшли транспорт та відправили посилку на склад, де зберігатиметься посилка , транспорт необхідно розвантажити, щоб звільнити місце, цей етап також можна назвати підготовкою до транспортування;

6. Розвантаження та підготовка автомобіля до перевезення нової партії товару, завантаження пакета;

7. Після завантаження транспортного пакета дається наказ рухатись до місця призначення;

8. Після прибуття на місце та розвантаження пакету транспортного засобу дається завдання повернутися додому.

При створені стратегічних ігор такий порядок завдань є звичним, оскільки більш складні завдання просто потрібно розбити на простіші, що в кінці кінців перетворюється в звичайні накази.

Зокрема, цей метод показує роботу між гравцями під час гри. Для того щоб створити додаткової техніки в стратегіях в реальному часі вони часто використовують механізм взаємодії продавців з іншими гравцями. Такий обмін дає змогу можливості впливати на економічний розвиток .

Така система може ефективно виконати більшість завдань. Крім розподілу завдань по порядку, проблема в стратегіях заключається лише в необхідності правильного вибору найкращого способу дій. Наприклад, економічний розвиток зараз може відкласти створення нової армії, але в подальшому збільшить потоки ресурсів. Таким чином можна сказати що правильна постановка проритетних завдань в ієрархічній задачі залишається важливим фактором, за допомогою якого можна визначити кінцеву ефективність.

2.5 Підсумок

Для створення мікроконтролю я обрав нейронні мережі. У порівнянні з всіма існуючими методами, новизною є використання першочергового аналізу для визначення зон впливу та ризиків.

Для макроконтролю система заключається у послідовності дій і в подальшому використання ієрархічної мережі задач.

Використовуючи такі методи ми можемо створить доволі гнучкий, ефективний штучний інтелект, який зможе ефективно працювати на будь яких платформах.

Зокрема, цей метод показує роботу між гравцями під час гри. Для того щоб створити додаткової техніки в стратегіях в реальному часі вони часто використовують механізм взаємодії продавців з іншими гравцями. Такий обмін дає змогу можливості впливати на економічний розвиток .

Така система може ефективно виконати більшість завдань. Крім розподілу завдань по порядку, проблема в стратегіях заключається лише в необхідності правильного вибору найкращого способу дій. Наприклад, економічний розвиток зараз може відкласти створення нової армії, але в подальшому збільшить потоки ресурсів. Таким чином можна сказати що правильна постановка проритетних завдань в ієрархічній задачі залишається важливим фактором, за допомогою якого можна визначити кінцеву ефективність.

3 СЕРЕДОВИЩЕ ТЕСТУВАННЯ ШТУЧНОГО ІНТЕЛЕКТУ

3.1 Вибір ігрового движка

Для того щоб отримати реальні результати та для визначення ефективності нашого штучного інтелекту нам потрібно створити середовище тестування. Я вважаю що для ігор стратегій в реальному часі тестування є дуже важливою частиною створення штучного інтелекту.

Особливою частиною таких ігор це використання різноманітних підходів та технологій. Наприклад, для деяких представників стратегічних ігор основною технологією є захист власних територій. База ресурсів залежить від контрольованих областей на локації. Це вимагає агресивного стилю гри , адже втрата території може означати лише програш .

Можна знайти ще велику кількість різних особливостей стратегій в реальному часі. Для деяких ігор основою є керування кожним ботом. А для інших навпаки.

При всіх цих факторах будуть залежати і результати які в кінці покажуть ефективність штучного інтелекту.

Для того щоб протестувати штучний інтелект найкраще нам буде створити стратегічну гру. Тобто не повністю гру а лише її прототип з основними механіками стратегії в реальному часі. Система повинна мати специфіку гри щоб нею можна було користуватися і на мобільних пристроях і на комп’ютері. Це можна виділити як дуже важливий факт адже якщо продукт буде гарно працювати на мобільних пристроях, то і на комп’ютерах з цим не буде ніяких проблем. Адже всі ми добре знаємо що мобільні пристрої поступаються в потужностях до персональних комп’ютерів.

Для того щоб протестувати штучний інтелект зазвичай використовують ігровий движок що називається microRTS. Він є спрощеним варіантом стратегії в реальному часі. Хоча і можливості в нього значно обмежені. Зазвичай в іграх використовує кілька

видів бойових воїнів, вони відрізняються кількістю одиниць здоров'я, силою і типом розподіл на дальність бою). Список доступних ігрових одиниць буде виглядати так:

- Звичайний робітник;
- Легкий бот для простого легкого бою;
- Важкий бот для ближнього бою;
- Бот підтримки з дальніх точок;
- Місця дислокації;
- Ресурси які добуваються.

Ігрові можливості в застосунку microRTS обмежені, але це не заважає йому бути певною альтернативо тестового середовища для тестування штучного інтелекту. Проте складністю є те, що в готовому середовищі важко вносити зміни. Також важливою відмінністю від звичайних стратегій є відсутність технічного прогресу. Це в подальшому тестуванні може бути великою проблемою.

Зваживши всі за і проти я вирішив що застосунок microRTS не підійде до нашого тестування нашого штучного інтелекту. Тому нам потрібно використовувати щось інше. А саме потрібно використовувати такі популярні ігрові рушії як : Unreal Engine, Unity. Адже написання власного ігрового рушія я вважаю не досить вдалою ідеєю.

Основні моменти для розробки прототиму стратегії яку ми в подальшому будемо тестувати я вважаю:

- Швидкість розробки;
- Популярність ігрового рушія;
- Наповненість самої гри;
- Можливість адаптації для мобільних пристроїв;
- Підтримки Популярних протоколів для вдалого з’єднання з сервером.

Для того щоб обрати найбільш підходящий ігровий рушій треба оцінити плюси та мінуси ігрових движків: Unreal Engine та Unity. Для цього я використаю таблицю:

Таблиця 3.1 – Порівняння Unreal Engine та Unity

Ігровий рушій / Параметри	Unreal Engine	Unity
Визначення	Рушій з відкритим кодом	Кросплатформенний ігровий рушій.
Розробник	Epic Games	Unity Technologies
Мови програмування	C++	C#
Сфера активного використання	Використовується для розробки ігор для ПК, мобільних телефонів. консолей тощо.	Використовується для розробки ігор для ПК, мобільних телефонів. консолей тощо.
Найбільш успішні функції	Надійна багатокористувацька структура, VFX та моделювання частинок.	Підтримка 2D, якісні компоненти для створення анімації.
Вихідний код рушія	Вихідний код відкритий.	Вихідний код закритий
Ціноутворення	Безкоштовний до публікації комерційного продукту.	Базова версія безкоштовна.

Після того як я проаналізував дані що отримав після порівняння цих двох движків я можу надати свої висновки:

- За допомогою Unreal Engine можна створити більш графічно досконаліший продукт аніж за допомогою рушія Unity;

- Unreal Engine потребує більшого обсягу знань аніж Unity;
- Кожен з них може використовуватись для розробки продуктів і на мобільні пристрої але в Unity це простіше через те що цей інструмент вже вбудований до самої програми на відміну від Unreal Engine;
- Зазвичай Unity вважають кращим при створенні тестувальних систем.

Тож проаналізувавши все це я вирішив що використовуватиму ігровий движок Unity.

3.2 Створення прототипу гри

Гру я створюватиму в режимі 3D на ігровому движку Unity. Чому саме 3D ? Тому що в майбутньому якщо я захочу я зможу з легкістю його вдосконалювати та розвивати як повноцінний продукт.

Для початку нам потрібно створити новий проект. В новому проекті мені довелося провести різноманітні налаштування які будуть корисні для нашої гри.

При розробці даного проекту я вважаю що графічна складова не сильно важлива на відміну від механіки. Наш ігровий додаток матиме будову класичної гри, яка складається з декількох основних частин. Кожна частина виконує свою певну функцію.

- перша частина, яка нам потрібна – це управління;
- друга частина це рендеринг об’єктів на сцені:
- звук, також важлива частина гри;
- остання частина яка відповідатиме за процес гри та її механіку (ця частина управляє взаємодією ігрових об’єктів, перемикати інтерфейс і т.д.).

Повний опис вимог здійснюється на етапі проектування програми. Для того щоб уточнити деталі нам потрібно виділити процеси, що відбувається в певній предметній області. Описати такі процеси можна завдяки UML діаграми прецедентів. Ця діаграма є так би мовити сценарієм або іншими словами варіантом використання додатку. Тут описується функціонування гри з точки зору користувача.

В програмі потрібно реалізувати систему меню, яка дозволятиме користувачу розпочинати гру, завершати, ставити на паузу, і розпочинати знову по завершенню. Для проектування даної частини зручно використовувати діаграми станів.

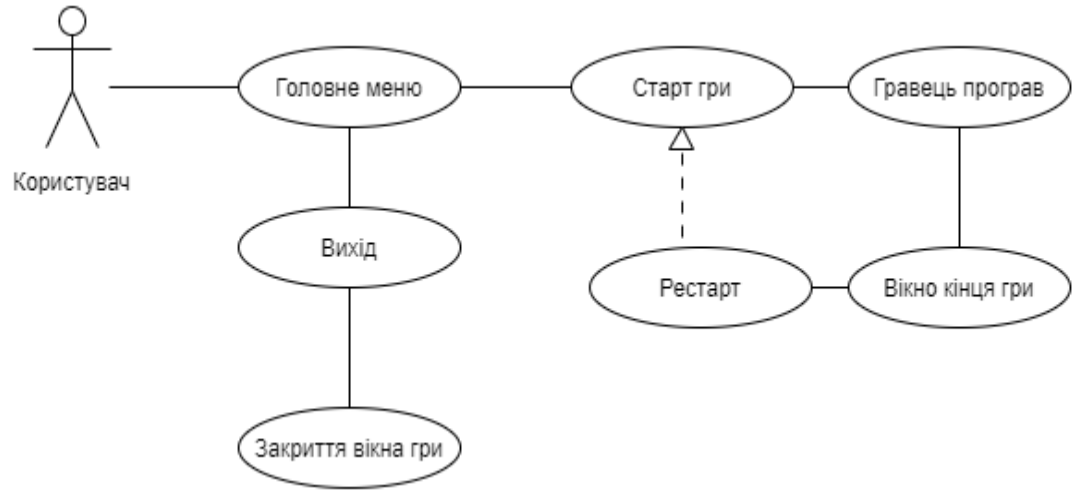


Рисунок 3.1 - Діаграма прецедентів

Я вирішив зробити головне меню максимально простим тому воно складатиметься з наступних пунктів:

- старт гри, вибравши даний пункт розпочнеться гра;
- вихід , коли користувач натиснув на дану кнопку вікно гра закривається.

Також під час самої гри в нас доступне меню паузи в якому доступні пункти:

- продовжити гру, при виборі даного пункту гравець повертається назад в гру і продовжує рівно з того моменту коли він натиснув на кнопку паузи;
- завершити гру, користувач повернеться до головного меню.

І третє меню це меню коли користувач завершив гру, або програв і воно складається з таких пунктів:

- розпочати гру заново;
- завершити гру.

На мою думку на стадії розробки ігрового додатку головному меню слід надати теж чимало уваги. Адже це перша річ яку бачить користувач і по ній складається перше враження від гри.

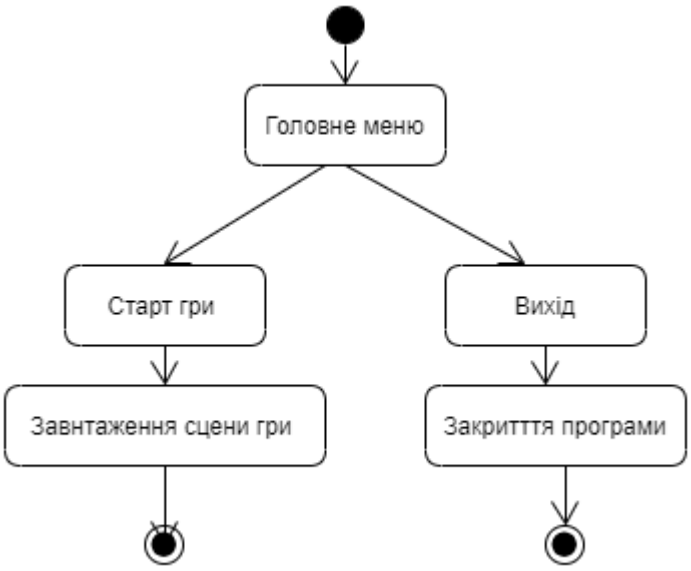


Рисунок 3.2 - Діаграма станів головного меню

Також важливим моментом механіки гри буде камера. Вона буде мати два режими показу ігрового поля. Перший це вона матиме змогу рухатись по межах зони активності а другий це прив’язка на самому активному об’єкті або на певному персонажі. Також камера матиме дуже важливу функцію для стратегічних ігор це мати змогу маштабувати зображення . Адже дії можуть проходити на великих локаціях і для для зручності контролю це просто необхідна можливість.

3.3 Ігрові об’єкти та навколишній світ

Всі дії будуть відбуватися на ігровому полі яке амтиме певний розмір. Також ми можемо назвати це поле землею. Проте для нашого варіанту прототипу гри велика

територія нам не потрібна тому ми використаємо невелику зону, яку в майбутньому можна буде розширювати.

Також важливими елементами в грі є його наповненість. Тобто різноманітні будівлі рослинність інколи навіть тварини. Хоч і для тестування все це нам не дуже потрібно але я все ж таки використаю їх для того щоб можна було протестувати як штучний інтелект в майбутньому буде оминати всі ці перешкоди при прокладанні свого маршруду то свої точок призначення.

Рисунок 3.3 – Наповнення ігрового поля

3.4 Опис технік та методів середовища

Як ми вже визначилися раніше дана гра не є повноцінною грою а лише набагато спрощеним її аналогом. Також можна назвати концептом. Тут будуть відображені лише основні механіки які потрібні для того щоб після розробки штучного інтелекту ми могли його так би мовити протестувати. Тобто ми можемо називати це тестовим середовищем для тестування штучного інтелекту.

При розробці інтерфейсу обов’язково потрібно врахувати всі специфіки ігор в жанрі стратегій в реальному часі.

В грі можна додати велику кількість об’єктів. Головними об’єктами в таких іграх вважають ботів. Адже вони роблять весь ігровий процес. Ботів можна поділити на декілька видів це воїна та звичайні працівники. Для кожного з цих видів ми повинні спеціально налаштувати штучний інтелект та обучити його коректним діям для кожного з них.

Рисунок 3.3 – Воїни

Кожен з юнітів, що представлені , мають певні значення захисту. Інші бойові підрозділи можуть покращити свої значення після вивчення певних знань. Для кожного юніта в таблицях зберігається набір даних, що показує кількість пошкоджених одиниць, які блок можна застосувати до певного рівня броні залежно від рівня досвіду. Значення досвіду накопичується, коли загін бере участь у бою, кожен піхотинець отримує певну кількість балів за зроблену задачу – пошкодження або ліквідацію противника.

Смертельна шкода залежить від здатності оговтатися від удару. Кожен воїн і споруда мають декілька Для того щоб відновити здоров'я. Після того як зникла одна одиниця здоров’я ніяких додаткових операцій не потрібно, а відновитися можна лише

до останнього «пошкодженого» рівня. Розмір одиниці здоров’я — 10. Якщо максимальний рівень життя бойового підрозділу — 50, а поточний — 34, то після виходу з бою та відпочинку протягом 30 секунд життя бойового підрозділу почне відновлюватися, але воно обмежено. до 40 одиниць. На графіку показано розмір сектора охорони здоров’я.

Для простоти користувачі побачать рівень життя кожного блоку. Оскільки на маленькому екрані важко відразу проаналізувати велику кількість чисел, які постійно рухаються разом з бойовим підрозділом, для спрощення інтерфейсу кожного підрозділу показник життя виділить одиницю кольором, що відповідає фактичному здоров’ю. значення. Для будівель система подібна, але кожна будівля має в рази більше одиниць HP, тому розмір будівлі збільшився в 4 рази (одна зона -40 одиниць здоров'я).

3.5 Підсумок

В даному розділі ми розглядали основні технології які ми використовуватиме під час створення нашого тестового середовища для стратегії в реальному часі.

Також я розглянув популярні ігрові рушії та обрав найбільш підходящий. Мій вибір припав на Unity.

Ще я створив ігрову локацію яка в подальшому використовуватиметься при розробці.

4 ПРОЕКТУВАННЯ СЕРВЕРУ ІГРРОВОГО ЗАСТОСУНКУ

4.1 Аналіз варіантів.

При створенні сучасних ігор робота із сервером гри вважається звичайною практикою. Для ігор цього жанру, тобто стратегій в реальному часі найпопулярнішими видом гри є гра гравців один між одним адже гра проти штучного інтелекту швидко набридає. Тому що під час гри можна вивчити весь алгоритм дій штучного інтелекту і всі його дії стають передбачуваними.

Як правило, багатокористувацьку гру або режим багатокористувацької гри в грі можна розділити на кілька категорій. Перший і найпростіший з них - це діаграма успіху та певні маркери досягнень. Більшість ігрових платформ дозволяють розробникам використовувати власну систему зберігання даних. Наприклад, мобільна платформа PlayMarket дозволяє розробникам використовувати власний API, який надає повні функції та можливості для вищезгаданих функцій. Варто зазначити, що покупки, які можна зробити в додатку. Цю важливу функцію зазвичай потрібно пов’язувати лише з платформою продажів програми, щоб гравці могли її завантажити.

Далі при розробці ігрових додатків для взаємодії із сервером є збереження даних гравця, прогресу та інших важливих елементів гри. Це дозволяє краще взаємодії з програмою, легко перемикатися на інших пристроях і відновлювати гру, навіть якщо ви втартили гаджет на якому ви грали або видаляли гру на якийсь час. Цей режим ігри не потребує постійної взаємодії між людьми під час ігрового сеансу.

На мою думку правильного підходу при створенні серверу для стратегій в реальному часі не існує. Різноманітні компанії що займаються розробкою ігор в жанрі стратегій в реальному часі користуються власними розробками. Адже кожен по своєму бачить свій напрямок розвитку штучного інтелекту і можливості свого продукту.

Раніше при створені перших стратегій в реальному часі використовували пряме з’єднання між користувачами. Накази що відправляв користувач своїм воїнам дублювалися і противнику . Це досить проста та й водночас важка технологія. Бо при великій кількості гравців потрібно було обробляти та фільтрувати велику кількість однакових команд що спричиняло перебої та тугість програми. І лише через певний час почали використовувати багатокористувацькі режими гри.

Одні з перших в розробили якісноу систему здатну згуртувати багатокористувацької гри для стратегії в реальному часі була командування що розробила всесвітньо відодому стратегію (Age Of Empires). Для обох частин гри використовували однакові серверні технології. Вся синхронізація проходила між клієнтами тобто хостами , інакші клієнти не мали прямого підключення як самотні один до одного. На той час перед розробниками поставали доволі складні проблеми – враховуючи слабкий інтернет у більшості користувачів та потенційні затримки синхронізації. Ще однією проблемою була синхронізація з швидкістю відбиток змін ось графічному пристрої гравця понад чим слабкі процесори і (частіше в медицині) недостатність відеокарти (від замовця) замовлення з графікою була на низькому рівні, а в іграх (зберігання) умови тривимірності симулювався – у дійсності (пояснити) щ куди й до чого обраховувалось тобі) ось двовимірній мапі.

Для синхронізації розробники Age Of Empires використали широко вживаний до сьогодні метод синхронізація з фіксованими точками часу . Значимість цього підходу полягала у синхронізації даних проміж клієнтами всього-на-всього в певні рівні моменти часу. Це дозволяло обрати оптимальний час для синхронізації проміж різними клієнтами. За деякий час цього всі накази, які гравці віддавали, синхронізувались з основною симуляцією дуже швидко а з іншими користувачами тривалість моменту синхронізації в визначений час. Для спрощення позначка розробники використовували крок комуннікації, з кожного такого кроку починали виконуватись всього-на-всього прості задачі,

Проте ця технологія не протрималась досить довго. Ігрова індустрія почала розвиватись дуже стрімко. Ігровий процес більше не заключається в грі лише пари людей. В одну гру можуть грати зараз тисячі людей одночасно з будь яких куточків світу. Саме тому розробникам довелося впровалити нову техологію, яка стала називатись клієнт серверною.

4.2 Аналіз технології розробки серверної частини гри

Підхід до вибору архітектури серевру напряму залежить від технічного завдання і результату якого очікують від нього. Саме через це немає сталого алгоритму створення серверної частини програмного продукту. Чому свме? Бо кожен програмний продукт має свою специфіку використання свою сферу використання та клієнто орієнтованість. Є пений алгоритм який вказує на те який вигляд повинна мати архітектура і його потрібно дотримкватись. Нижче розлянемо види технологій розробки серверної частини.

Монолітна архутектура також може називатись традиційною. Використовується для того щоб побудувати серверну частину для веб-застосунків. Тут притаманні такі риси: основна частина серверної частини досить громіздка вона має доступ до великої кількості різноманітних даних. А всі інші компоненти тісно співпрацюють між собою . В свою чергу це можна вважати як і плюсом так і мінусом такого методу.

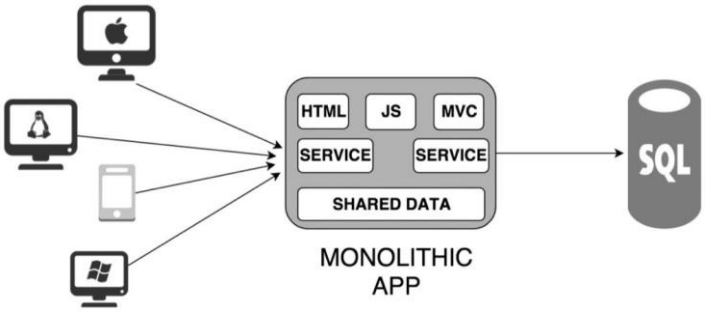


Рисунок 4.1 – Монолітна архутектура

Перевагами даного типу є спільне використання ресурсів у додатку що дозволяє ефективно використовувати апаратні ресурси системи. Дані при такому типу створення передаються досить швидко. Оскільки функцію яку ми хочемо виконати ми викликаємо напряму не використовуючи для цього сторонні методи та модулі.

Недоліками можна назвати те що це є вже досить застарілим форматом і більшість розробників не рекомдовують використовувати його при роботі з комерційними проектами. Адже технології зараз в наш час дуже активно розвиваються. А для того щоб використовувати монолітну архітектуру потрібно в свій час її було активно підтримувати та постійно оновлювати.

Сервісна архітектура – Певна кількість елементів які відповідають на певні функції відповідно. За допомогою такого методу можна чітко виокремити всі модулі системи, роділивши для цього сервер на декілька менших технологіій, які в свою чергу мають достить таки поширену зону роботи але з обмеженим функціоналом.

Якщо дану архітектру порівняти з монолітною то ми знайдемо більше позитивних якастей. З розвитком серверної частини розвивалась і функція поділу серверної частини на сервіси, які ставали все меншими меншими. Постійна підтримка ефективності та постійне оновлення призвело до створення нової архітектури мікросервісної.

При мікросервісній архітектурі набір сервісів поділяється на велику кількість незалежних один від одного елементів із своїми мін базами даних.

Таблиця 4.1 – Аналіз мікросервісної та сервісно-орієнтованих архітектур

	Сервісно-орієнтована	Мікросервісна
Рівень зберігання даних	Модель SOA має єдиний рівень зберігання даних, який поділяють усі сервіси в застосунку.	Додатки мікросервісів в основному присвячують базу даних або інший тип пам'яті службам, які цього потребують.
Основне завдання	Орієнтований на максимальне багаторазове використання сервісів.	Основний акцент на декомпозиції системи та її функціоналу.

Продовження таблиця 4.1

Наслідки необхідності розширення	Вимагає модифікації моноліта.	Вимагають створення нового сервісу
Можливості CI/CD	DevOps і безперервна доставка можливі, але не є стандартом	Акцент на DevOps та безперервній доставці
Призначення серверу	Призначений для спільного використання сервісів у різних службах.	Призначений для розміщення сервісів, які можуть функціонувати незалежно.
Рівень доступу	Часто включає спільний доступ до компонентів	Як правило, він не включає спільний доступ до компонентів
Сховища даних	Передбачає обмін сховищем даних між службами	Кожен сервіс може мати незалежне зберігання даних.
Обмін даними	Спілкується через ESB	Обмін даними через шар API
Основна перевага	Покладається на спільне використання ресурсів	Базується на обмеженому контексті зв'язку.
Розгортання	Менша гнучкість у розгортанні	Швидке та просте розгортання.
Розміри технологічного стеку	Технологічний стек SOA нижчий у порівнянні з Мікросервісом.	Стек технологій мікросервісу може бути дуже великим.

При мікросервісній архітектурі набір сервісів поділяється на велику кількість незалежних один від одного елементів із своїми мін базами даних.

Мікросервісний метод в при ідеальних показниках відображає комунікацію проміж різними сервісами та компаніями , для якої ми створюємо продукт.

Окремим пунктом, що додає мікросервісному архітектурному підходу переваг, є перспектива постійноих оновлень для найкращого кожного окремого мікросервісу за винятком зміни інших. Це дозволяє частинам команди робити максимально незалежно і творити зміни виключно у ту частину продукту, за яку вони несуть відповідальність.

При використанні мікросервісів так само є шанс масштабувати окремі компоненти (мікросервіси) в системі, при цьому не змінюючи потужності серверів для всіх інших. Це просто від неможливий створити в моноліті, адже критерій використовує спільний фізичний сервер з його процесорними потужностями та об'ємом оперативної пам'яті.

Важливою функціональною особливістю мікросервісів є повна непідпорядкованість та автономність. Це дозволяє вдосталь по-простому виконувати свою справу, при цьому не змінюючи структуру взаємодії з уже існуючим мікросервісом, та таким чином розширення системи не потребує фактичних дій виправдовує інших команд розробки. Необхідність в такому поділі одного мікросервісу на декілька менших може бути велике розширення функціоналу, і походженні у мікросервісі певних проблем.

Для прикладу дозволено розглянути розділення мікросервісу повідомлень (Notifications) на невелику сервісну систему. інформування збирати докупи користувачів є фактично в будь-якій системі. Розглянемо мобільний ігровий продукт притаманний для стратегій. Реалізованими функціями є перспектива автономної гри проти ботів, гри з використанням мережі межи гравцями, а також функціонал турнірних битв і, від відповідний таблиці успіхів. Під час неігрових можливостей, є функціонал покупок певних елементів гри. крім того через те межи гравцями є шанс використовувати чат, створювати власні матчі та запрошувати своїх друзів на матч. Невідповідний функціонал є доволі типовим здатний згуртувати всіх ігор, не тільки в жанрі стратегій в реальному часі. Основна риса цього продукту потребує функціоналу повідомлень користувачів: Перед початком турнірним матчем гравцю потрібно прийти за запрошенням через електронну пошту користувача має прийти електронний лист. За деякий час отримання інформація виправдовує іншого користувача на телефон отримувача належімо прийти пуш повідомлення. При надані користувачем номеру телефона найважливіші інформація не заборонено висилає тобі на номер, у свою чергу інформація здатний згуртувати телефону можуть використовуватись у який спосіб зберігатись для безпеки.

Для серверної частини використана мікросервісна архітектура з великою вірогідністю здатний згуртувати надсилання повідомлень користувачам було створено відокремлений послуги Notifications Service. За допомогою брокера повідомлень сервіси отримують запит на надсилання різнотипних нагадувань для користувачів виправдовує інших мікросервісів системи.

Загальноприйнятим підходом покищо є розділення мікросервісів на максимальну число сервісів для спрощеного адмініструванні і подальшої розробки. Коли розглянути мікросервіс повідомлень, то можна розділити його на умовні частини. Обробка вхідних повідомлень і певна додаткова послідовність думок необхідна під час відправкою, є спільною частиною щоб, всі з них – відправлялись кожним з описаних вище способів.

У більшості випадків в системах існує чітке розділення засобів надсилання різних типів повідомлень. Для електронної пошти може бути існує використаний побічний сервіс на зразок MailGun, для SMS повідомлень існують інакші сервіси (наприклад, Twilio), Push-повідомлення обробляються самим додатком у смартфоні тобто на коп'ютері користувача, проте для їх надсилання потрібен зв’язок з використанням веб- сокетів. У кожного з цих сервісів є власна документація і API для взаємодії. монолітній взаємозв'язок поміж ними – це послідовний ступінь що обробляє сповіщення з брокера та викликає функціонал для надсилання. Проаналізувавши ці думка можливо зміркувати що висновок, як мікросервіс може бути існувати розділений на менші частини, кожна з яких відповідатиме за ясний лімітований ряд функцій.

Таке розділення зовсім ніяк не впливає при взаємодії з іншими елементами системи та ніяким чином не змінює взаємодію інших мікросервісів з надсилання повідомлень.

За винятком мікросервісів з розвитком хмарних технологій з’являється більш сильніші можливостей для управління системою з тисячами. Це призвело до появи підходу, що назвали мікро-сервісним. Специфікою цього підходу є розділення системи на окремі функції – «мікро-сервіси», які працюють за допомогою технологій

Azure Functions або AWS Functions. Такі уміння поводитися з ким дозволяє орудувати ресурсами і масштабувати дані системи на найнижчому рівні, проте ускладнює розгортання системи та відслідковування проблем.

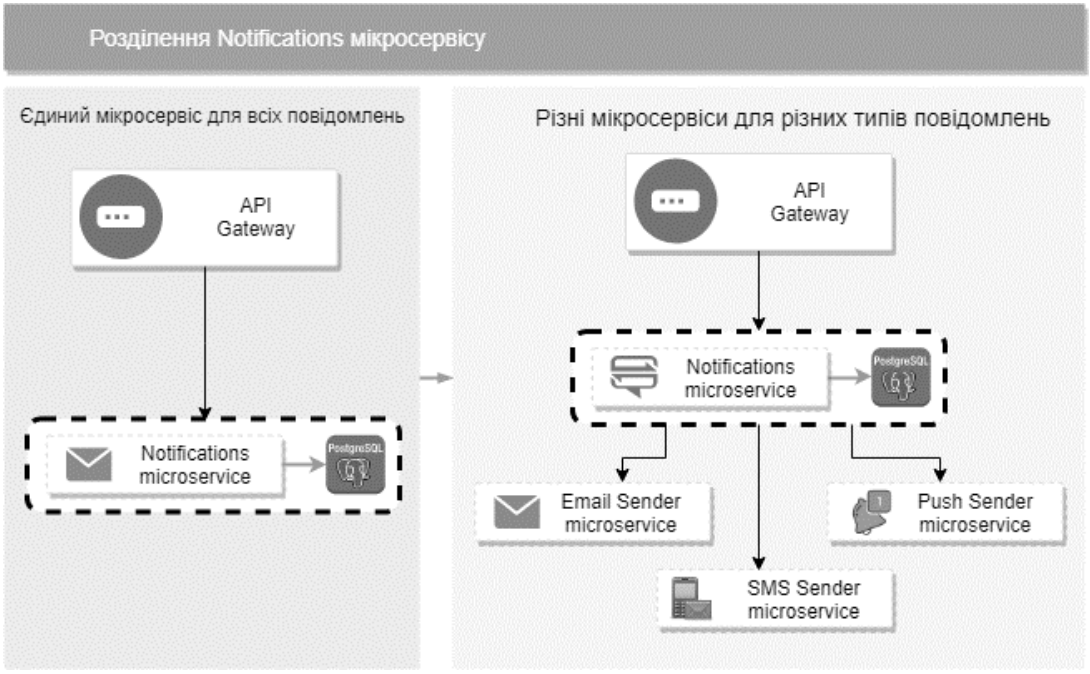


Рисунок 4.2 – Розподіл на мікросервіси

Таке розділення зовсім ніяк не впливає при взаємодії з іншими елементами системи та ніяким чином не змінює взаємодію інших мікросервісів з надсилання повідомлень.

За винятком мікросервісів з розвитком хмарних технологій з’являється більш сильніші можливостей для управління системою з тисячами. Це призвело до появи підходу, що назвали мікро-сервісним. Специфікою цього підходу є розділення системи на окремі функції – «мікро-сервіси», які працюють за допомогою технологій Azure Functions або AWS Functions. Такі уміння поводитися з ким дозволяє орудувати ресурсами і масштабувати дані системи на найнижчому рівні, проте ускладнює розгортання системи та відслідковування проблем.

4.3 Аналіз застосунку та серверної частини

Для того щоб ефективно обрати архітектуру серверної частини нам потрібно чітко визначити критерії за якими працюватимете наш застосунок.
Адже для кожної системи можуть знадобитись специфічні модулі та технології.

По-перше для ефективності нашої системи нам потрібно розробити систему в якій дані передаватимуться за допомогою найшвидшого алгоритму. Для того щоб використовувати це у багатокористувацькому режимі.Адже швидкість виконання та передачі даних є найважливішою проблемою в дамґній частині. Хоч і в більшій мірі все це залежить від користувачів. Тобто швидкості з'єднання з інтернетом та апаратних можливостей їх гаджетів. Проте не дивлячись на все це серверна частина нашого застосунку повинна витримувати навантаження не зменшуючи швидкість обробки та передачі даних. Також важливо щоб у користувачів в незалежності від навантажень на сервер не було жодних знижень ефективності.

Особливо важливо при створенні нашого застосунку або як вже зазначалося раніше тестового середовища для тестування штучного інтелекту важлива ефективність обробки та передачі даних. Також важливо збереження цих даних в цілісному вигляді. Адже в подальшому ці дані використовуватимуться для аналізу та обробки. Для того щоб зробити наш штучний інтелект більш ефективним. Ці дані нам знадобляться також під час навчання нейронної мережі. Гравцям також можуть знадобитися ці дані . Наприклад для аналізу поразки. Для того щоб проаналізувати свої помилки зробити певні корегування в тактиці та перемогти в результаті.

Так значну увагу треба приділити базі даних в якій зберігатимуться всі наші дані. Вже і саме сховище може значно сповільнювати процес обміну даними.

Ще одним значним модулем є сервіс що збирає дані про проведені дії. За допомогою його встановлюється зв'язок штучного інтелекту з гравцем. Таким чином штучний інтелект підлаштовується під гравця. Цей сервіс зберігає всі дані кожного користувача. Дані ці можна класифікувати як накази. Тобто все що гравець наказав

робити ботам записується і в подальшому використовується при навчанні нейронної мережі. Також цей сервіс зберігає дані про кожну бойову одиницю а точніше про їх стан рівень розвитку та рівень здоров'я. Всі ці дані зберігаються на сервері. І під час того як гравець логіниться в грі ці дані повинні завантажитись на пристрій користувача. Таким чином навіть можна провести певну застереження про доброчесність гравців. Адже всі їх досягнення зберігаються на сторонньому ресурсі які при кожному заводі в гру підтягуються і в разі завершення гри вони автоматично оновлюються на сервері.

Також можна виділити сервіт який пов'язаний з авторизацією користувачів.важливти елементом для цього модуля потрібно виділити безпеку даних. Тобто безпеку даних користувачів що збереженні для авторизації. Тут ми повинні використовувати шифрування даних для того щоб дані не потрапили до сторонніх осіб. Це дуже важливий момент, хоча він і призводить до зменшення ефективності роботи при передачі даних.

Далі в нас знаходиться сервіс що займається навчанням нейронних мереж. Головною проблемою є затримка роботи через використання і обробку великих обсягів інформації. В подальшому якщо ми захочемо модернізувати нашу тестову систему то нам потрібно значно покращити ефективність роботи даного сервісу. На цьому етапі ми можемо налаштувати роботу так щоб штучний інтелект який ми навчаємо міг аналізувати певні ситуації в різних комбінаціях вирішення певних завдань.проте ця система потребує великих потужностей процесора та немалого обсягу оперативної пам'яті. Бо під час даного процесу обробляється велика кількість інформації. І штучному інтелекту доводиться прораховувати велику кількість вирішень і щ всього різноманіття обрати найбільш ефективний варіант. Що допоможе найшвидше досягти цілі.

Проаналізувавши всю інформацію вказану вище ми можемо зробити висновок що монолітну архітектуру не ефективно використовувати через складність використання певних сервісів. А підхід з використанням сервісів можна назвати більш

оптимальним для використання в нашому завданні. Адже робота нашого тестового середовища поділяється на різні сервіси для того щоб вони могли виконувати потрібні ним завдання. Проте таких сервісів потрібно буде створити дуже велику кількість. Сервісно-орієнтовану архітектуру ми використовувати також не можемо. Бо вона використовує загальну базу даних. Що в подальшому може викликати слабку виробничу потужність і зменшити швидкодію нашого середовища.

Тому найбільш ефективним я вважаю буде використання мікросервісної архітектури в якій всі сервіси або ж частини будуть повністю незалежними один від одного. Схему роботи даної архітектури можна спостерігати нижче.

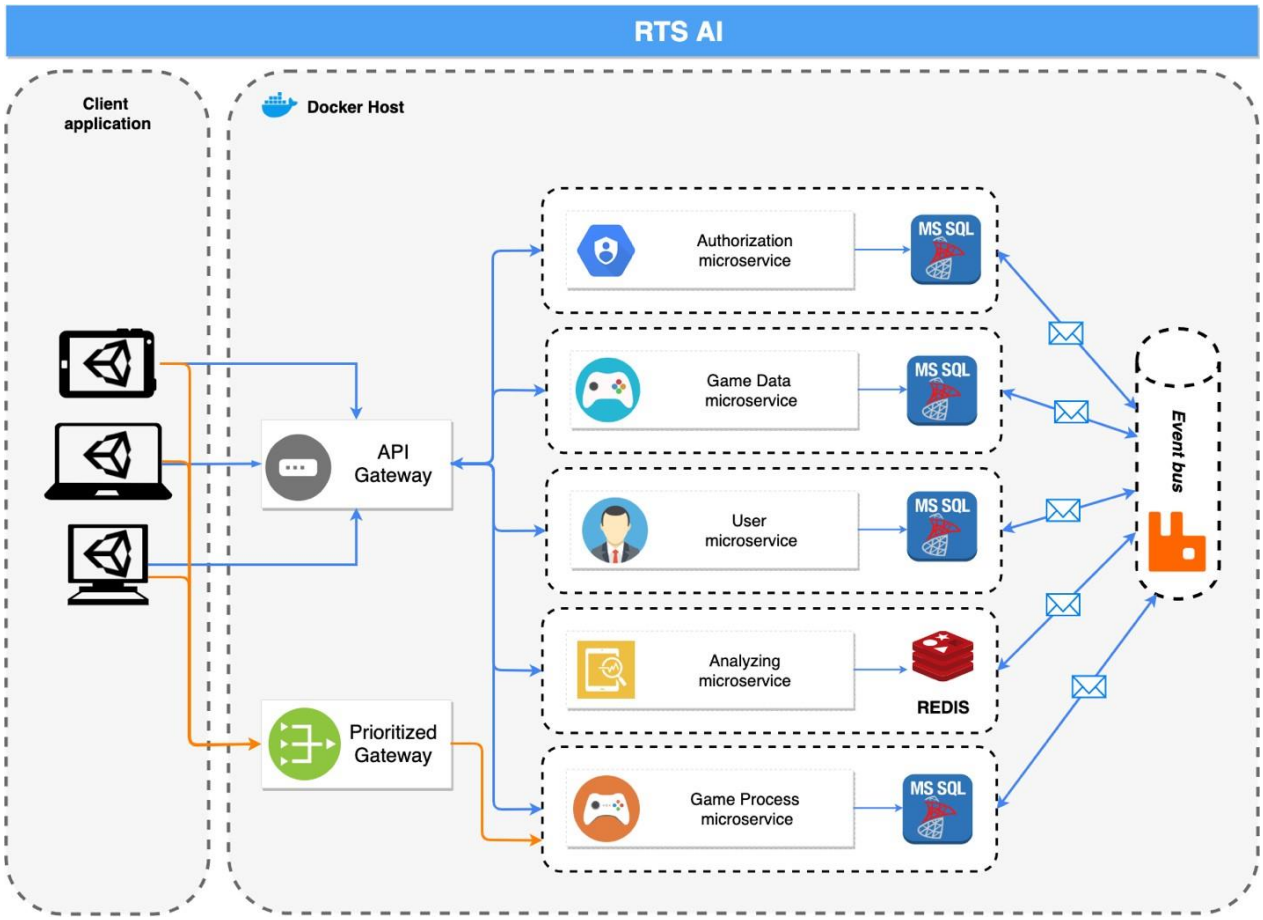


Рисунок 4.6 – Архітектура системи

Також важливо виділити систему яка буде якісно взаємодіяти між сервісами. Адже всі запити починають одразу виконуватись при потраплянні в сервісну частину. І таким чином ми можемо злегкістю перевантажити наш сервіс що може позначитись на швидкості роботи всього продукту. Тож для того що всього цього уникнути нам потрібно буде розробити ще систему порядку повідомлень що буде зберігати всі запити і таким чином сервіс виконуваме їх поступово один за одним по мірі їх надходження.

Для упорядкування запитів ми можемо використовувати технологію RabbitMQ. Ця технологія може використовуватись з більшістю популярниз мов програмування. Також є можливість паралельно обробляти декілька запитів що ніяк не відображається на швидкості роботи системи.

4.4 Сервіс попереднього аналізу та обробки запитів користувачів

Для того щоб мати змогу нормально взаємодіяти з клієнтами існує декілька прийомів:

- Прямолінійне з'єднання з сервером;
- Сервіс що є так би мовити посередником при з'єднанні між самою програмою та сервером.

Прямолінійне з'єднання заключається в тому що запити або дані прямують одразу на сервер. В же на сервері вони розходяться по своїм сервісам відповідно. Недостатком такого методу можна виділити те що система не може бути адаптивною або гнучкою. Виконання програмного продукту ускладнюється. Адже при надсиланні запиту використовують один канал передачі даних. А якщо нам потрібно надіслати тисячу запитів в нас створиться одразу ж і тисяча різних каналів що направляються до своїм сервісів які їм потрібні. Саме це і навантажуватиме систему. Також такий підхід потребує більш потужний комп'ютер та стабільне інтернет з'єднання.

також можна виділити таку проблему як постійний вільний прямий доступ до сервісів. Це може призвести до втрати особистої інформації.

Другий спосіб це використання модуля посередника між сервісами та користувачем.

В роботі програми може бути декілька таких програм посередників вони працюють при передачі запитів на сервер. Мінусом можна вважати те що програма посередник не може зберігати дані. В них зберігається лише певна послідовність запитів.

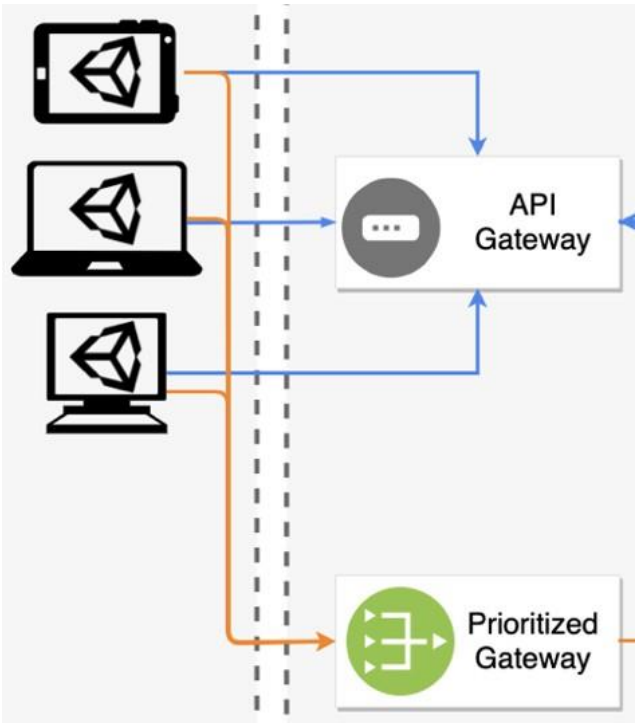


Рисунок 4.7 – Технологія передачі даних через програму посередника

На рисунку ми можемо побачити як використовують такий метод. При такому підході створюються два сервіси вони відповідали за обробку запитів користувачів.

Один працює з простими запитами або ще можна назвати їх загальними. Наприклад це звичайний запит що відповідає за запуск гри пошук багатокористувацькому лоббі.

Другий використовують щоб обробити запити внутрішньоігрові. Тобто це ті запити які відповідають за процес гри.

SignalR - це бібліотека яку використовують для того щоб створити з'єднання які допоможуть обмінюватись інформацією між сервером та користувачем. Цю бібліотеку розробила компанія Microsoft. Я обрав дану бібліотеку лише через її розповсюдженість ті легке встановлення до мого тестового середовища. За допомогою цієї бібліотеки ми можемо обирати велику кількість найоптимальніших протоколів для забезпечення передачі даних між сервером та користувачем.

4.5 Мікросервіси для обробки даних

Тут ми розглянемо мікросервіси що відповідають за обробку та аналіз ігрових даних. За всі мікропроцесор ігрового процесу відповідають лише два мікропроцеси: мікросервіс ігрового процесу та мікросервіс ігрових даних. Розділення грає дуже важливу роль. Адже при передачі даних важлива швидкість передачі. Якщо б все було на одному процесі то при визнанні наприклад якихось важких процесів в нас би падала швидкість передачі даних.

А рамках сервісу процесу гри відбувається аналіз ігрових даних та поточного ігрового стану.

Використання такого методу дозволяє нормалізувати навантаження на сервер та клієнтський пристрій.

Самі ж дані зберігаються у базі даних. Для цього я вирішив використати не реляційну базу даних MongoDB. Чому саме її ? Тому що передача даних буде проводитись по протоколу JSON. Передавані дані можуть відрізнятись відповідно до типу гри або ж релізної версії.

Як же працюватиме ця система ? Адже для нормального розуміння роботи та правильної розробки тестового середовища нам потріібно розуміти як працюватиме наше середовище. Нижче наведено алгоритм роботи програми:

- Під час ігрового процесу на сервері зберігаються всі дії які виконав користувач. Таким чином вони потрапляють до мікросервісу ігрових даних де і саме зберігаються.
- Дані оновлюються з однаковою періодичністю, за допомогою таймеру.
- Далі в самій грі створюється набір даних в який включено інформація про військо оборонні споруди якими керує користувач. Це потрібно для того щоб синхронізувати ігрові клієнти обох гравців що грають один проти одного.

За допомогою такого методу можна швидко синхронізувати ігрові дані для кожного з гравців та в разі їх розбіжностей швидко надіслати інформацію і виправити все.

Сервіс ігрових даних також зберігає таку інформацію:

- Стан ігрової сесії;
- Загальна інформація про гравців;
- Дані про військо та споруди;
- Інформація про стан кожного ж юнітів.

Тобто сервіс ігрових даних зберігає в собі всі дані про гру.

4.6 Штучний інтелект та мікросервіс для його аналізу

Також важливим мікросервісної можна назвати мікросервіс що відповідатиме за авторизацію гравців та збереження їх власних даних. Для штучного інтелекту цей мікросервіс дозволить індивідуальне навчання для кожного з гравців. Тобто кожен гравець грає по своєму. І завдяки базі даних з інформацією про користувача до ней буде додаватись інформація яка допоможе штучному інтелекту ефективно реагувати на тип гри для кожного з гравців. Тобто для ефективності штучного інтелекту зберігаються всі дані з гри кожного користувача. Таким чином для кожного користувача

буде підбиратись система ботів або супротивники відповідного рівня за яким вони грають.

Показниками для аналізу в даному мікросервісі використовують показники ефективності виконань задача при тирі гри кожного з користувачів. Але варто зауважити що окрім підбору рівня гри за допомогою штучного інтелекту для кожного з користувачів цей сервіт також відповідає за навчання нейронних мереж.

Причини через які я обрав цю систему при розробці свого тестового середовища:

- З кінцевих пристроїв користувачів знімаються процеси що займаються навчанням та аналізом роботи штучного інтелекту. Ці процеси відбуватимуться на сервері. Таким чином і гру можуть грати і користувачі із слабкими пристроями;
- Для ефективного навчання штучного інтелекту використовується інформація всіх користувачів та темпу їх гри;
- Також можна використовувати додаткові типи навчання для штучного інтелекту.

4.7 Мікросервіс для авторизації користувачів

Це один з найбільш важливих сервісів. Тому що в ньому зберігаються особисті дані всі користувачів. Всі інформація що з зберігається в даному сервісі повинна бути максимально захищена від третіх осіб. Для того щоб авторизуватись в грі ми може використати різні варіанти:

- Електронна пошта та пароль;
- Логін та пароль;
- Обліковий запис Google;
- Обліковий запис Facebook.

Цей мікросервіс відповідає за цілісність даних користувачів. Вся інформація зберігається у зашифрованому вигляді.

Наприклад якщо ми використовуватимемо реферальну систему для запрошення користувачів. То в цьому сервісі також зберігатиметься інформація щодо зв'язку користувачів один з одним.

Також тут буде зберігатись інформація щодо заблокованих користувачів які підозрюються або були помітні при нечесній грі.

4.8 Підсумок до розділу

В даному розділі я розглянув можливості для розробки серверної частини нашого тестового середовища. Для свого проекту я обрав мікросервісу архітектуру .

Також я окремо розглянув кожен мікросервіс як і чому саме він буде використовуватися в моєму тестовому середовищі.

В ньому буде зберігатись інформація щодо заблокованих користувачів які підозрюються або були помітні при нечесній грі.

Показниками для аналізу в даному мікросервісі використовують показники ефективності виконань задача при тирі гри кожного з користувачів. Але варто зауважити що окрім підбору рівня гри за допомогою штучного інтелекту для кожного з користувачів цей сервіт також відповідає за навчання нейронних мереж.

5 РОЗРОБКА ТА ТЕСТУВАННЯ

5.1 Створення тактичного штучного інтелекту.

Для розробки штучного інтелекту було прийнято використовувати платформу .NET. Це пов’язано з тим, що нейронну мережу та інші елементи штучного інтелекту необхідно вбудовувати в програму користувача, це дозволить гравцям використовувати гру з ботом навіть за відсутності Інтернету, а також прискорить і стабільність, оскільки всі розрахунки та рішення будуть доступні користувачеві.

При роботі тестового середовища я створив окремий модуль який відповідатиме за штучний інтелект. В ньому будуватиметься структура нейронних мереж та аналіз ваг та переміщень. Також в цьому модулі буде реалізовано метод попереднього аналізу даних.

Як же саме проходить попередня обробка? Давайте розглянемо. В нас є перелік певних методів: - SplitTerritory, HandleZones, GetUnitsPower, IsOwnUnitsExists, CountPower, CountInfluence. Всі ці методи працюватимуть один за одним. Спочатку вони визначають територію на якг ми знаходимося, потім вони поділяють цю територію на певні зони. Далі відбувається аналіз кожної з зон. Там певний алгоритм дізнається всю інформацію про ворожі об'єкти що будуть знаходитися в зоні.

Для порожніх зон аналіз не відбувається адже аналізувати там нічого.

Після завершення так званої первинної обробки даних для зон в яких було проведено аналіз ми використовуватимемо нейронні мережу. Для того щоб використовувати нейронні мережі і нашому застосунку було прийнято рішення розробки власної моделі .

Таке рішення довелось застосувати через проблеми які могли б виникнути з движком Unity, ми б не змогли зручно додати пакети з Nuget.

Nuget – це основний менеджер для проектів, написаних з використанням платформи .NET і є стандартом для розробки. Окрім завантаження пакетів з спеціальних

репозиторії в є можливість прямого додавання бібліотек класів у вихідний код Unity проекту.

Для того щоб створити нейронну мережу нам потрібні нейрони. Вони є основними елементами нейронних мереж. В обраній схемі класів Neuron відповідає за даний елемент мережі, він реалізує інтерфейс INeuron. Кожен з нейронів розроблюваної мережі належить класу NeuronLayer, що відповідає за кожен окремий шар мережі, незалежно від типу шару та його характеристик.

Кожен з шарів нейронів в свою чергу належить класу NeuralNetwork, що є основним класом для нейронної мережі.

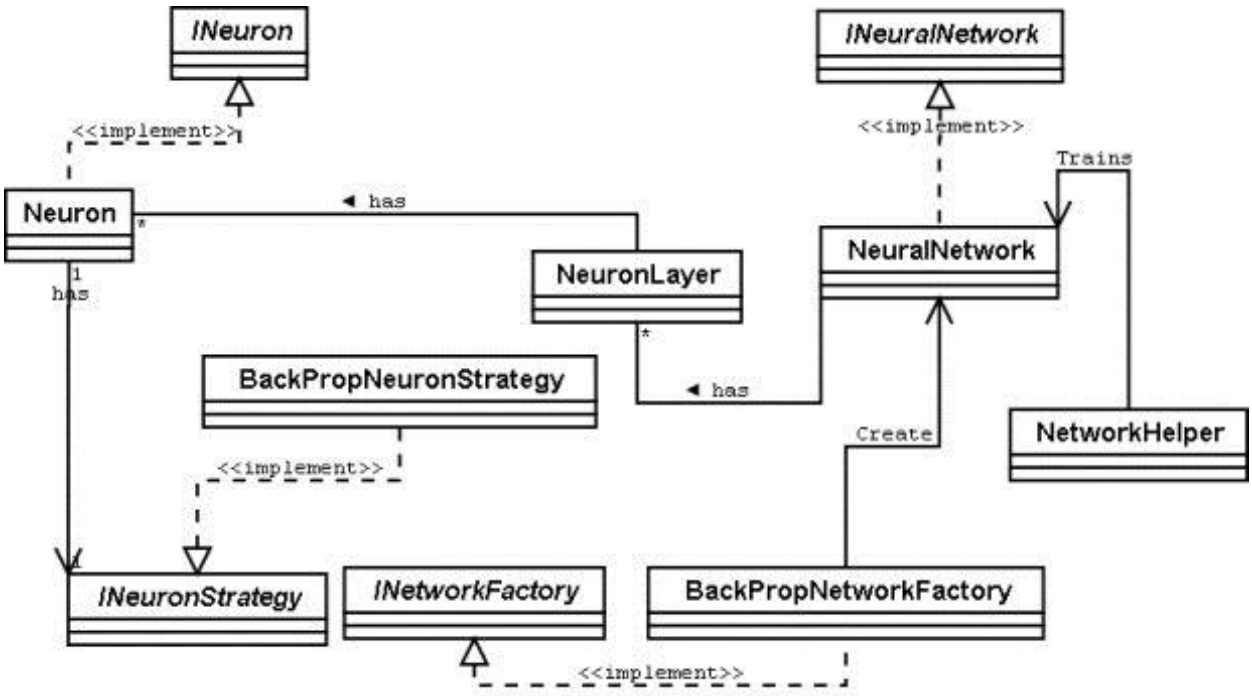


Рисунок 5.1 – Структура класів для розробки нейронної мережі

Оскільки структуру нейронної мережі можна змінювати і коефіцієнти На рівні клієнта це буде залежати від сервера, тому було вирішено призначити окремий клас BackPropNetworkFactory. Назва цього класу пояснюється тим, що в системі є 2 мережеві методи для різних елементів системи. Оскільки створювана мережа є нейронною мережею, використовуйте алгоритм зворотного поширення для її навчання. Цей клас

відповідатиме за створення нейронної мережі після завантаження необхідних коефіцієнтів на початковому етапі гри. NetworkHelper, відповідальний за навчання нейронної мережі, реалізований на стороні сервера і, по суті, є інтерфейсом, який надає інформацію про навчання нейронної мережі.

З схеми функції навчання мережі (Додаток G) видно, що мережу можна навчати двома способами – автоматичне моделювання та налаштування інструктора.

Метод автоматичного моделювання є складнішим для сервера, але він є ефективним способом визначення миттєвої винагороди за рішення групи при виборі дії.

З огляду на те, що вихідний шар нейронної мережі складається з 3 нейронів, значення, що повертається цих нейронів, знаходиться в діапазоні від -1 до 1, два з яких визначають напрямок, а один задає режим перетворення, можна визначити що для групи (тобто для регіону Серія бойових одиниць) можна визначити 27 рішень.

Оскільки загальна площа розділена на 25 квадратів, кількість необхідних симуляцій пропорційна збільшенню кількості одиниць, зайнятих бойовою одиницею.

Необхідна кількість симуляцій завантажить сервер і завадить ефективному використанню функцій моделювання для повного навчання моделі.

Для того, щоб навчити систему, користувач використовує веб-інтерфейс, який дозволяє розмістити торії, передати для аналізу та переглянути результати обробки алгоритму та покроково. Цей метод нейронної мережі набагато швидший, оскільки правила вже підготовлені і не потрібні додаткові ресурси. Проблема цього методу в тому, що люди не можуть гарантувати правильний результат.

Навчання самої мережі — власне зміна коефіцієнтів — відбувається за типовими алгоритмами. На відміну від людського сприйняття, нейронні мережі, по суті, являють собою набір значень, виражених у формі матриці, і всі дії над цими значеннями також відбуваються в рамках лінійної алгебри.

Однак платформа .NET не має вбудованого алгоритму обробки матриць. Бібліотека Math.NET Numerics використовується для забезпечення ефективної та швидкої обробки матриць.

Це простий і потужний інструмент, який може ефективно обробляти матрицю. Щоб підключити її до проекту, потрібно завантажити пакет Nuget або додати файл бібліотеки .dll.

5.2 Розробка середовища тестування

У процесі проектування тестового середовища було визначено багато моментів, які є особливо важливими при розробці тестового середовища. По-перше, це структура керування камерою.

Створено окремий скрипт CameraMovement.cs для керування камерою, він відповідає за всі рухи камери. Однією з особливостей цього скрипту є адаптація руху до пристроїв із сенсорними екранами. Це дозволяє ефективно використовувати цю модель камери навіть під час гри на мобільному пристрої.

Щоб спростити керування камерою на кількох площинах, кожна площина розділена на управління певним об’єктом.

Крім керування камерою, особлива увага була приділена створенню зручного інтерфейсу в дизайні системи.

Для його створення Unity використовує елементи інтерфейсу, вбудовані в ігровий движок. Інтерфейс поділяється на два типи взаємодії - інтерфейс і ігрове поле. Це показує можливість зміни стану та впливу на ігрові об'єкти через інтерфейс користувача та ігрові об'єкти.

З діаграми видно, що багато елементів інтерфейсу тісно перетинаються з елементами арени Вибір ігрових об’єктів впливає на інтерфейс, а використання інтерфейсу також впливає на арену. Це досить складна система для реалізації зручних функцій і

забезпечен ня комфортного ігрового процесу, тому що ви можете використовувати різні і більш зручні варіанти для надання команд при необхідності.

Усі команди, які можна подавати в грі, мають чітку структуру. Це необхідно, оскільки здібності гравця і робота повинні бути збалансовані, тому що якщо у гравця буде більше інструментів для управління роботом, робот не зможе йому протистояти через неможливість подавати певні команди. Кожне замовлення має свій код, який дозволяє зручно синхронізувати дані з сервером і передавати їх між гравцями.

Модуль контролера використовується для отримання та передачі даних, зв'язку з модулем запитів веб-сторінки та інших функцій керування в грі.

За своєю структурою контролери можна розділити на два типи – той що відповідає за управління ботами та той що керує будівлями. Для нормальної роботи обидва цих контролери працюють один з одним. На рисунку 5.2 ми можемо побачити повний цикл роботи всіх компонентів що працюватимуть в нашому тестовому середовищі на движку Unity.

На даному рисунку зеленими точками зображені методи що можуть працювати в скриптах. А червоними стрілками позначені методи які постійно обов'язково створюються в усіх класах.

В створених нами контролерах всі дії які виконує користувач напряму несуть важливий вплив на ігрову логіку. Саме тому після виконання будь-якої дії користувача в нас викликається метод Update.

Суть роботи контролера полягає в тому що в кожному фреймі відбувається перевірка дій які були виконані користувачем і що потрібно змінити перед тим як відображати сцену. Саме тому метод Update найбільш навантажений метод в програмі.

Взагалі робота з бібліотеками на движку Unity така ж як і на виконанні у веб-застосунках. Тому що вона дозволяє використовувати наш ігровий клієнт для роботи з сервером.

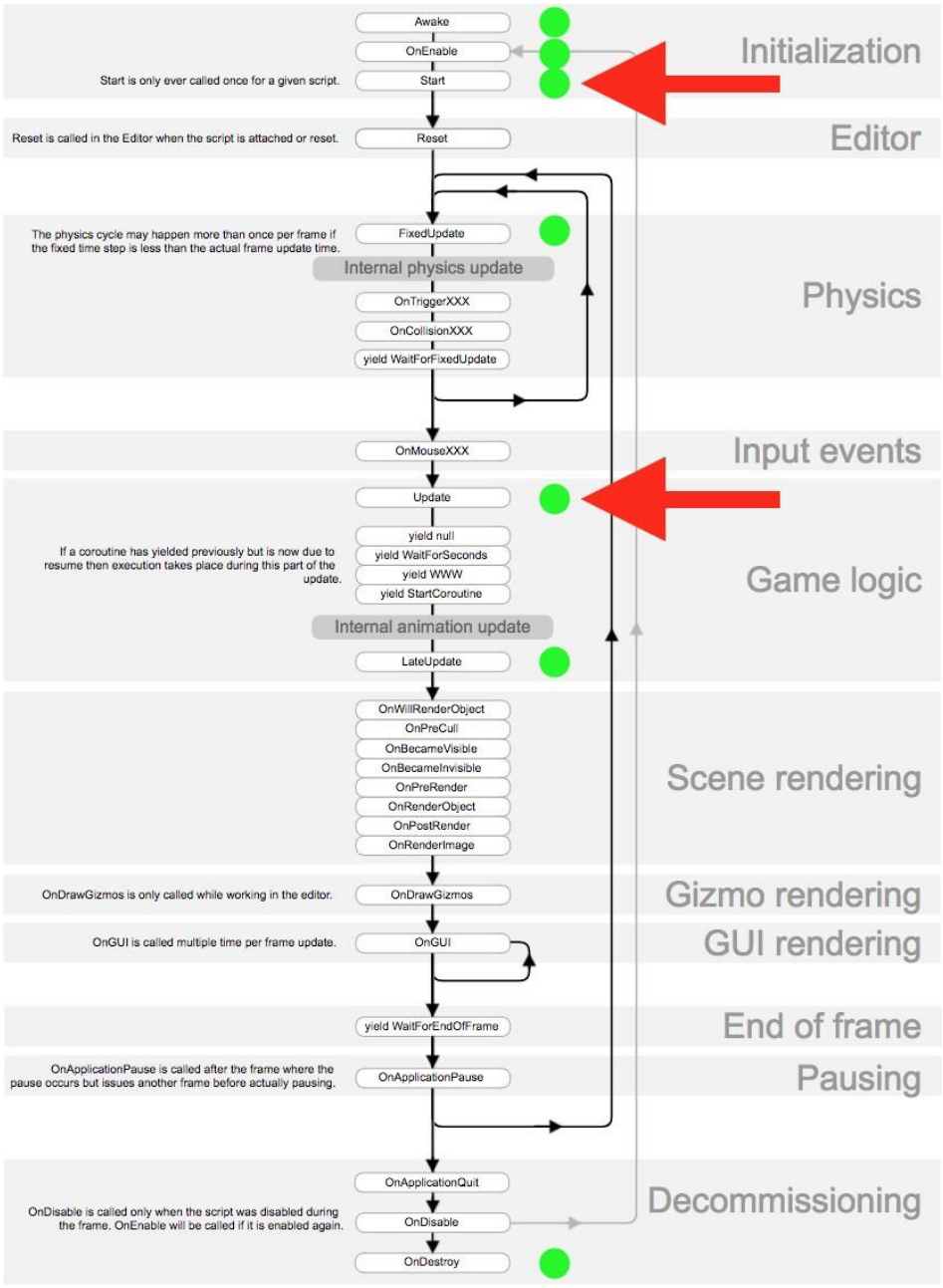


Рисунок 5.2 – Цикл роботи компонентів програми

5.3 Створення серверної частини

Під час створення робочого варіанту нашого тестового середовища для кожного мікросервісу я створював схеми бази даних. По стуктурі ці бази досить громіздкі тому що в них повинен зберігатись весь ігровий процес. Основною таблицею є

GameHistory. В ній ми зберігаємо дані про початок нової гри До цієї таблиці прив’язані майже всі таблиці нашої бази даних. Помилки що виникають під час ігрового процесу записуються також в окрему таблицю.

Всі дані про гру та об’єкти зберігаються в таблицях WarriorLibrary та BuildingLibrary відповідно. Також є таблиці для всіх подій та наказів ActionType та EventType.

Якщо враховувати особливості функціоналу мікросервісної архітектури, деякі дані, наприклад PalyerID, які на перший погляд повинні би бути зовнішніми ключами, не посилаються на таблиці, оскільки дані збережені в іншому сервісі. Зв’язування даних відбувається в компоненті превинної обробки даних (API Gateway), а синхронізація – через програму посередника повідомлень (Service bus).

Для того щоб розробити базу даних було використано підхід Code-first. Робота цього підходу заключається в тому, що для створення таблиць і загалом бази даних використовуються моделі, що були написані мовою C#. Перевагою цього підходу вважається використання для розробки тільки C#. Моделі для отримання даних нам обов’язково доведеться створювати. Але якщо б ми використовували інші підходи то нам би довелося робити одну і ту саму роботу двічі тобто це створення моделей та створення таблиць в базі даних.

Проте даний підхід має певні недоліки. Поперше при такому підході швидкість роботи з бази даних набагато менша, оскільки запити до сховища даних виконуються з використання LINQ to Database. LINQ (Language-Integrated Query) – це частина мови програмування C#, яка додає можливості зручного підходу в створенні запитів до будь яких колекції даних. Також LINQ допомагає аналізувати та обробляти дані. В певній мірі це схоже на розширену версію SQL, що інтегрована в C#.

Для того щоб використовувати LINQ потрібно перш за все зв’язати дані моделі з таблицями в базі даних. Не дивлячись на те, що таблиці створюються змоделей, для їх підключення потрібно додати їх до файлу налаштувань. Для створення будь яких

налаштувань потрібно підключити базу даних до проекту. Додавання будь яких сервісів та модулів виконується в Startup файлі,що використовується для створення вебхосту в файлі Programm. Така структура вважається стандартно для проектів, розроблених з використанням .NET Core. В Startup файлі виконується підключення всіх middleware, websocket та обробників запитів. При небхідності додається авторизація та авторизація користувача. Якщо веб-застосунок потребує використання бази даних або стороннього модуля, він має бути доданим до колекції сервісів, що ініціалізується перед початковим запуском хосту.

5.4 Результати тестування штучного інтелекту

Основною проблемою яка виникла перед початком тестування виникло налаштування для навчання системи. Адже ми використовуємо адаптивну систему навчання. Аде система яка надає дані про поточні військові сили, дає хмогу моделювати ситуації з різними варіантами і давати системі правильний вибір з максимальною кількістю балів. Все це ускладнює навчання мережі. Також ми можемо коригувати дії для за допомогою правильного результату в систему.

Основним показником за допомогою якого оцінювалось ефективність використання бойових одиниць. Під час зіткнення бойових одиниць розраховувалась загальна оцінка групи військ, щовступає в територію контакту. Після зміни території зіткнення – переходу фокусу на іншу точку карти або знищення всіх бойових одиниць однієї з сторін – підраховується загальна оцінка. Загалом ця оцінка є різницею втрат гравців в поточній зоні, якщо втрати менші – дії в зоні були успішними. Ефективне використання військ в разі використання таких метрик можливо врахувати навіть при повній втраті бойових одиниць в межах території, оскільки вони можуть нанести пошкоджень ворожим бойовим одиницям більше, ніж мають. Ефективність дій також можна визначити величиною оцінки – чим вона вища – тим вища різниця між втратами гравця та супротивника.

Для тестування я використав достатньо типові попередньо прописані алгоритми. Вони часто вкориистовуються в звичайній моделі стратегічних ігор. Нижче в таблиці 1 ми можемо побачити результати та порівняння поєдинків бота що керувався нейронною мережею та бота що діяв по загальноприйнятому алгоритму.

Для тримання більш ефективних результатів я використовував три різних моделі навчання нейронних мереж:

- Перший працював під управлінням нейронної мережі де вріний результат задавався вручну;
- Другий працював під управліннм нейронної мережі з використанням власних даних після їх обробки;
- Загально прийнятий алгоритм дій для ботів.

Таблиця 5.1 – Оцінки ефективності

	Атака	Захист	Баланс
1	67%	61%	64%
2	72%	65%	71%
3	75%	66%	73%

Аналізуючи результати, можна побачити, що ефективність нейронної мережі проти стандартними ботами досить висока навіть за невеликій кількості тестових даних. Ефективність проти агресивного агента вища, тому що в бою набагато легше отримати перевагу - бот не чекає на підкріплення і втрачає підрозділи невеликими групами. Нижчі результати із захисним ботом обумовлені частим виведенням останнього межі зони конфлікту через необхідність більшої маси військ чи відсутності стратегічного пункту оборони. Відступ із зони не змінює цінності бойових підрозділів, бо перемоги немає.

Усього було проведено 10 матчів. Кількість боїв у кожному з матчів була різною. Цікаві результати – це також залежність кількості битв у матчі від тренування нейронної мережі (рис. 2). 6). Форма, як видно. 6, спочатку нейронна мережа заповнена випадковими коефіцієнтами, результатом є швидке ураження через відповідні контрольні помилки. Зі збільшенням підготовки нейронної мережі кількість зустрічей в одному матчі – активації тактичного штучного інтелекту – збільшилася, оскільки після закінчення бою пройшло більше часу через більш високу ефективність. У другій частині графіка показана частина мережевого навчання, в якій штучний інтелект стає сильнішим за вбудований інтелект. Кількість зіткнень за гру почала знижуватися, тому що через високу ефективність використання зіткнення призвели до величезних втрат бойових підрозділів і на потрібен час.

Розбираючи табл. 1 можна зробити висновок, що композиція дослідження за підсумками симуляції та дослідження за прикладами зробленим людиною вважається кращим варіантом. Проблема вивчення лише з симуляції у цьому, що приклади людини мають найбільш узагальнене бачення картини і прорахування кроків тривалий час.

Ефективність штучного походження розуму може збільшити навчанням мережі на більшій кількості даних, а також поліпшити метод симуляції для автоматизації процесу.

5.5 Підсумок розділу

Розроблено тестове середовище, яке вважається полегшеною стратегією в реальному часі з обмеженою чисельністю типів військ та кілька полегшених ігрових правил.

До роботи у випробувальному середовищі створено справжній розум до ухвалення тактичних і стратегічних висновків під управлінням. У моделі роботи штучного походження розуму активно застосовувалися розклади, подібні до використовуваних

у згорткових нейронних мережах.

За підсумками випробування можна кваліфікувати, що справжній розум досить ефективний навпроти роботів і є гнучким ворогом. Для подальшого вдосконалення результатів необхідно збільшити розміри тестових даних і змінювати їх.

Таким чином композиція дослідження за підсумками симуляції та дослідження за прикладами зробленим людиною вважається кращим варіантом. Проблема вивчення лише з симуляції у цьому, що приклади людини мають найбільш узагальнене бачення картини і прорахування кроків тривалий час.

Ефективність штучного походження розуму може збільшити навчанням мережі на більшій кількості даних, а також поліпшити метод симуляції для автоматизації процесу.

Аналізуючи результати, можна побачити, що ефективність нейронної мережі проти стандартними ботами досить висока навіть за невеликій кількості тестових даних. Ефективність проти агресивного агента вища, тому що в бою набагато легше отримати перевагу - бот не чекає на підкріплення і втрачає підрозділи невеликими групами. Основним показником за допомогою якого оцінювалось ефективність використання бойових одиниць. Під час зіткнення бойових одиниць розраховувалась загальна оцінка групи військ, щовступає в територію контакту. Після зміни території зіткнення – переходу фокусу на іншу точку карти або знищення всіх бойових одиниць однієї з сторін – підраховується загальна оцінка.

ВИСНОВКИ

У результаті виконання роботи було досліджено актуальність обраної теми. Розглянуто та проаналізовано існуючі рішення та підходи до розробки штучного інтелекту для стратегій в реальному часі, виділено основні компоненти в стратегічному ШІ та досліджено особливості розробки макро- та мікро-стратегічного інтелекту, а також технології прогнозування дій супротивника.

Опрацьовано велику кількість інформації від передових видань у сфері розробок штучного інтелекту, надано загальну інформацію щодо розробки штучного інтелекту для ігрових продуктів та наукових проектів, пов’язаних з даним жанром ігор.

На основі проаналізованої інформації обрано оптимальний варіант комбінації агентів для створення наближеного до людини та потужного штучного інтелекту. Спроектовано реалізації обраних методів, дано розгорнуте порівняння та опис переваг саме такого підходу для створення інтелекту.

Обрано платформу та технології для створення тестового середовища, необхідного для перевірки штучного інтелекту на реальному продукті. Розроблено розподілену систему для розгортання застосунку та комунікації з сервером. Розроблено структуру серверної частини, обґрунтовано використання мікросервісного підходу для забезпечення стійкості системи.

Проаналізовано сучасні методи для створення та роботи з нейронними мережами, обґрунтовано використання нейронної мережі для додатку. Розглянуто можливості для серверного навчання моделі з подальшим використанням у користувацькому додатку результатів навчання моделі.

Розглянуто переваги побудови штучного інтелекту для стратегій з використанням ієрархічної мережі постановки задач. Створено ієрархічну мережу для гри в рамках тестового середовища, проаналізовано результати роботи мережі.

ПЕРЕЛІК ПОСИЛАНЬ

1. Morris R. Workshop Summary Kasparov vs. Big Blue: The Significance for Artificial Intelligence. *ICGA Journa*l. 1997. Vol. 20.
2. Buro M. Call for AI Research in RTS Games. *Proceedings of the AAAI Workshop on AI in Games*. AAAI Press, 2004.
3. Shantia A., Begue E., Wiering, M Connectionist Reinforcement Learning for Intelligent Unit Micro Management in StarCraft. *2011 International Joint Conference on Neural Networks (IJCNN)*. San Jose, CA USA, 31 July–5 August, 2011.
4. Amado, L. Reinforcement learning applied to RTS games. 2017.
5. Niel R., Krebbers J., Drugan M., Wiering M. Hierarchical Reinforcement Learning for Real-Time Strategy Games. *ICAART 2018 - 10th International Conference on Agents and Artificial Intelligenc.* 2018.
6. Szczepanski T., Aamodt, A. Case-Based Reasoning for Improved Micromanagement in Real-Time Strategy Games. *Case-Based Reasoning for Computer Games* at the 8th International Conference on Case-Based Reasoning. Seattle, WA, USA, 20–23 July, 2009.
7. Molineaux M., Aha D., Moore P. Learning Continuous Action Models in a Real-Time Strategy Environment. *Twenty-First International Florida Artificial Intelligence Research Society (FLAIRS) Conference*, pp. 257–262. Palo Alto, CA: AAAI Press, 2008.
8. Sailer F., Buro M., Lanctot M. Adversarial Planning Through Strategy Simulation. *IEEE Conference on Computational Intelligence and Games*, pp 80–87. Piscataway, NJ: Institute for Electrical and Electronics Engineers, 2007.
9. Churchill D., Saffidine A., Buro M. Fast Heuristic Search for RTS Game Combat Scenarios. *Eighth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, 112–117. Palo Alto, CA: AAAI Press, 2012.
10. Synnaeve G., Bessière P. A Bayesian Model for RTS Units Control Applied to StarCraft. *2011 IEEE Conference on Computational Intelligence and Games,* 190–

196. Piscataway, NJ: Institute for Electrical and Electronics Engineers, 2011b.

11. Gabriel I., Negru V., Zaharie D. Neuroevolution Based MultiAgent System forMicromanagement in Real-Time Strategy Games. *Fifth Balkan Conference inInformatics*, 32–39. New York: Association for Computing Machinery, 2012..
12. Stanescu M., Barriga N., Hess A., Burom M. Evaluating Real-Time Strategy Game States Using Convolutional Neural Networks, 2016.
13. Jie H., Weilong Y. A multi-size convolution neural network for RTS games winner prediction. *MATEC Web of Conferences*, 2018.
14. Aha D., Molineaux M., Ponsen M. Learning to Win: Case-Based Plan Selection in a Real-Time Cadena P., Garrido L. Fuzzy Case-Based Reasoning for Managing Strategic and Tactical Reasoning in Star-Craft. *Advances in Artificial Intelligence*, 2011.
15. Muñoz-Avila H., Aha D. 2004. On the Role of Explanation for Hierarchical Case-Based Planning in Real-Time Strategy Games. *Advances in Case-Based Reasoning, 7th European Conference, ECCBR 2004*. Lecture Notes in Computer Science. Berlin: Springer, 2004.
16. Laagland, J. A HTN Planner for a Real-Time Strategy Game, 2008.
17. Weber B., Ontañón S. Using Automated Replay Annotation for Case-Based Planning in Games. *Case-Based Reasoning for Computer Games at the 8th International Conference on Case-Based Reasoning*, Seattle, WA, USA, 20– 23 July, 2010.
18. Molineaux M., Klenk M., Aha D. Goal-Driven Autonomy in a Navy Strategy Simulation. *Twenty-Fourth AAAI Conference on Artificial Intelligence*. Palo Alto, CA: AAAI Press, 2010.
19. Weber B., Mateas M., Jhala, A. Applying Goal- Driven Autonomy to StarCraft.*Sixth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, 101–106. Palo Alto, CA: AAAI Press, 2010.
20. Kabanza F., Bellefeuille P., Bisson F., Benaskeur A., Irandoust H. Opponent Behaviour Recognition for Real-Time Strategy Games. *Plan, Activity, and Intent Recognition*: Papers from the AAAI Workshop. Technical Report WS- 10-15. Palo Alto, CA:

AAAI Press, 2010.

21. Weber B., Mateas M. A Data Mining Approach to Strategy Prediction. *2009 IEEE Symposium on Computational Intelligence and Games*, 140–147. Piscataway, NJ: Institute for Electrical and Electronics Engi- neers, 2009.
22. Dereszynski E., Hostetler J., Fern A., Dietterich T., Hoang, T., Udarbe, M. Learning Probabilistic Behavior Models in Real-Time Strategy Games. *Seventh AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, 20–25. Palo Alto, CA: AAAI Press, 2011.
23. Hostetler J., Dereszynski E., Dietterich T., Fern A. Inferring Strategies from Limited Reconnaissance in Real-Time Strategy Games. *Conference on Unceertainty in Artificial Intelligence,* Avalon, Catalina Island, CA, 15–17 August, 2012.
24. Zhu J., Villareale J., Javvaji N., Risi S., Löwe M., Weigelt R., Harteveld C. Player-AI Interaction: What Neural Network Games Reveal About AI as Play.2021.
25. Нейронні мережі - шлях до глибинного навчання. URL: https://codeguida.com/post/739 (дата звернення: 10.07.2021).
26. SimpleHierarchicalOrderedPlanner. URL: http://www.cs.umd.edu/projects/shop/index.html (дата звернення: 15.07.2021).
27. Unity vs. Unreal – Choosing a Game Engine. URL: https://gamedevacademy.org/unity-vs-unreal/ (дата звернення: 06.08.2021).
28. Свириденко О.А. Веб-застосунок для пошуку репетитора на базімікросервісної архітектури URL: https://ela.kpi.ua/handle/123456789/30233 (дата звернення: 21.10.2021).
29. Designing And Implementing A Neural Network Library. URL: https://www.codeproject.com/Articles/14342/Designing-And-Implementing-A-Neural-Network-Librar#_articleTop .
30. Strategy Game. *Case-Based Reasoning: Research and Development*, 2005.

ДОДАТОК А

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

НАВЧАЛЬНО–НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра інженерії програмного забезпечення

МАГІСТЕРСЬКА РОБОТА
«РОЗРОБКА СИСТЕМИ НА БАЗІ ТЕХНОЛОГІЙ ШТУЧНОГО ІНТЕЛЕКТУ ДЛЯ ІГОР В ЖАНРІ RTS»

Виконав: студент групи ПДМ-61 Борук В.Ю.

Керівник: доцент кафедри ІПЗ Дібрівний О.А.

Київ - 2021

МЕТА, ОБ'ЄКТА ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

Мета роботи: створення системи агентів штучного інтелекту з використанням власноруч створених мереж для забезпечення високої ефективності роботи ігрових персонажів для стратегій в реальному часі (RTS).

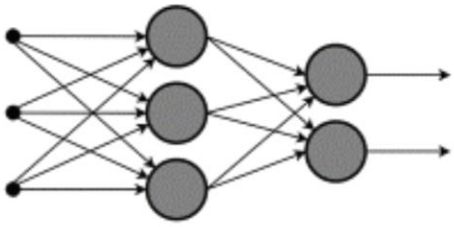
Об'єкт дослідження: був штучний інтелект створений для стратегій в реальному часі (RTS).

Предмет дослідження: штучний інтелект що розроблений з використанням нейронних мереж для ігор в жанрі стратегій в реальному часі (RTS). А також розроблені модулі для мікроконтролю (тактичний) та макроконтролю (стратегічний), для розподіленої системи з одним центральним сервером для більшої ефективності штучного інтелекту.

2

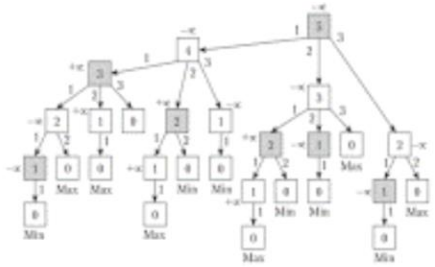
МІКРОКОНТРОЛЬ (ТАКТИЧНИЙ ШТУЧНИЙ ІНТЕЛЕКТ)

Навчання з підкріпленням



Навчання з підкріпленням це галузь машинного навчання, де агент повинен обирати дії, які приведуть до максимізації умовної винагороди. На відміну від навчання з учителем, агенту не задаються правильні варіанти вхідних та вихідних даних, а неоптимальні рішення агента не виправляються.

Пошук по ігровому дереву



Пошук по ігровому дереву є одним з алгоритмів, які нечасто використовують для стратегічних рішень, однак він може гарно себе показати для прийняття тактичних рішень.

4

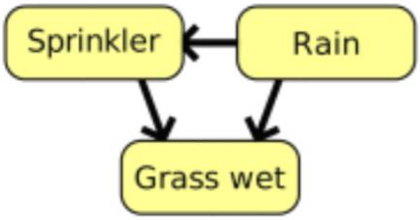
АКТУАЛЬНІСТЬ РОБОТИ

Актуальність даної роботи полягає в підвищенні ефективності штучного інтелекту для стратегій в реальному часі. Також розробка розподіленої системи з одним центральним сервером для більшої ефективності штучного інтелекту.

Наукова новизна полягає у вирішенні науково-практичного завдання розроблення ефективного штучного інтелекту для стратегій в реальному часі. Результати роботи можуть бути використані у подальших дослідженнях штучного інтелекту для ігор інших жанрів, а також можуть бути впроваджені в нові або вже існуючі ігрові продукти як ефективніша альтернатива існуючому штучному інтелекту.

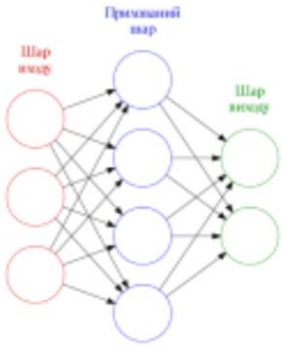
3

Баєсова модель



Баєсова модель, яка представляє собою набір випадкових змінних та залежностей між ними за допомогою орієнтованого ациклічного графу, може бути використана також в тактичному прийнятті рішень, для визначення напрямку руху юнітів

Штучна нейронна мережа

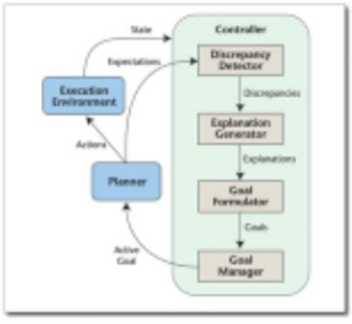


Може бути використана для впливу як на коефіцієнти, так і на топологію моделі нейронної мережі для контролю кожного окремого юніта. На вхід до моделі потрапляють дані, які юніт отримує з навколишнього середовищі, а також від сусідніх юнітів. Кожен носій нейромережі мав визначену функцію поточної ефективності, яка залежала від позиції, здоров'я, рівня та інших параметрів.

5

МАКРОКОНТРОЛЬ (СТРАТЕГІЧНИЙ ШТУЧНИЙ ІНТЕЛЕКТ)

Ієрархічне планування – це система розподілення цілей в грі в ієрархічну мережу задач, де присутні як основні високо рівневі задачі (перемога над супротивником), так і задачі низького рівня, які є підзадачами для високорівневих (найняти робочого). Ієрархічне планування часто поєднують з іншими алгоритмами для стратегічного планування, наприклад з рішеннями на основі прецедентів.



Автономне досягнення цілей є технікою вибору послідовностей дій, які змінюються в залежності від ситуації. При отриманні нових даних агент може змінити послідовність дій, які позначатимуть виконання цілі. Це дозволяє боту ефективно відповідати на різкі зміни у грі та бути гнучким у досягненні цілей, при цьому не переключаючись з них. Цей підхід часто комбінують з іншими, наприклад з прийняттям рішень на основі прецедентів.

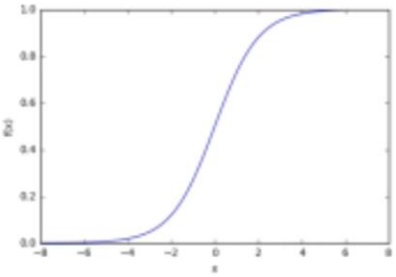
6

Для створення тактичного штучного інтелекту було обрано рішення з використанням нейронних мереж що дозволить йому реагувати правильно навіть у цілком незнайомій ситуації. В порівнянні з попередніми дослідженнями новими елементами є застосування попереднього аналізу для визначення зон впливу та реальної кількості одиниць, використання підходу, що ефективно використовується в згорткових нейронних мережах.

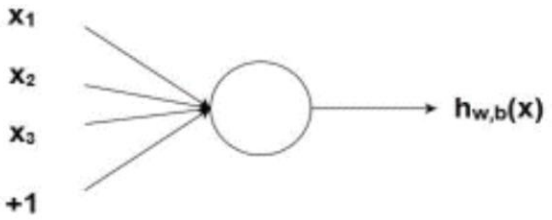
Біологічний нейрон імітується у ШНМ через активаційну функцію.

$$f(z) = \frac{1}{1+\exp(-z)}$$

З графіку можна побачити, що функція "активаційна" – вона росте з 0 до 1 з кожним збільшенням значення x. Сигмоїдна функція є гладкою і неперервною. Це означає, що функція має похідну, що у свою чергу є дуже важливим фактором для навчання алгоритму.



Коло на картинці зображує вузол. Вузол є "місцерозташуванням" активаційної функції, він приймає зважені входи, сумує їх, а потім вводить їх в активаційну функцію. Вивід активаційної функції представлений через h.



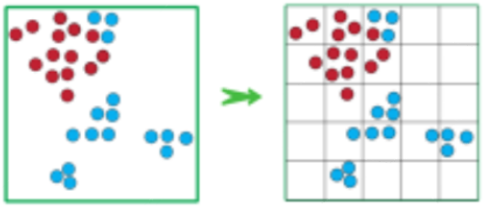
За вагу беруться числа (не бінарні), які потім множаться на вході і сумуються у вузлі. Зважений вхід у вузол має вигляд:

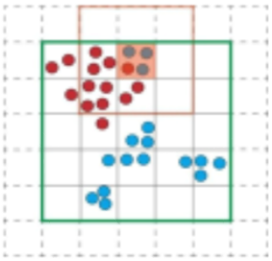
$$x_1w_1 + x_2w_2 + x_3w_3 + b$$

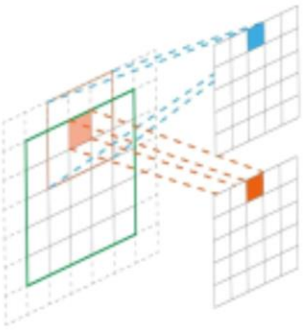
„де w_i - числові значення ваги. Ваги є значеннями, які будуть змінюватись протягом процесу навчання. b є вагою елемента зміщення на +1, включення ваги b робить вузол гнучкішим.

Для розробки штучного інтелекту буде застосовано штучні нейронні мережі, з певною специфікою, оскільки дані, які потрібно обробляти, мають комплексну структуру, та є нетиповими для використання в мережах.

Перш за все, для штучного інтелекту важливо враховувати правила тестового середовища. Тому агент буде використовувати доступні дані з певної зони – по суті те, що бачив би гравець. Отримані дані є набором координат і характеристик військ. Для аналізу дані потрібно обробити і класифікувати

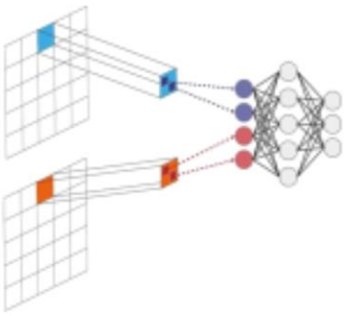




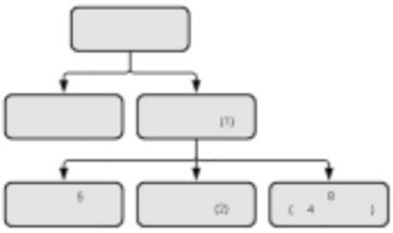


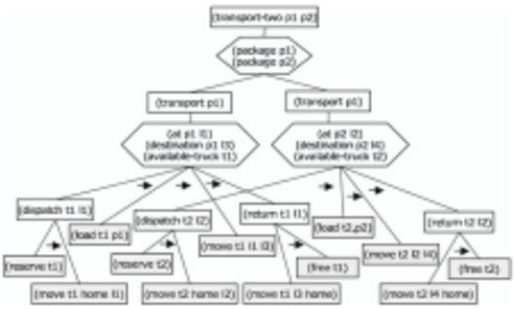
Згортка карти для зон впливу (верхній, позначено синім кольором) та ситуації в поточній зоні (нижній, позначено оранжевим кольором)

Після аналізу дані про оточуючі зони та поточний стан у конкретній зоні передається в нейронну мережу, яка в свою чергу надає на вихід дані про напрям руху (або утримання поточної зон) і режим руху.



Для стратегічного штучного інтелекту була обрана комбінація з «типовою» стартовою послідовністю дій і подальшим використанням ієрархічної мережі завдань для досягнення ефективного стратегічного управління в пізніх етапах гри. Ми використовуватимемо метод для розділення задачі на під-задачі.





Для стратегічного штучного інтелекту була обрана комбінація з розподіленням задач на підзадачі і подальшим використанням ієрархічної мережі завдань для досягнення ефективного стратегічного управління в пізніх етапах гри.

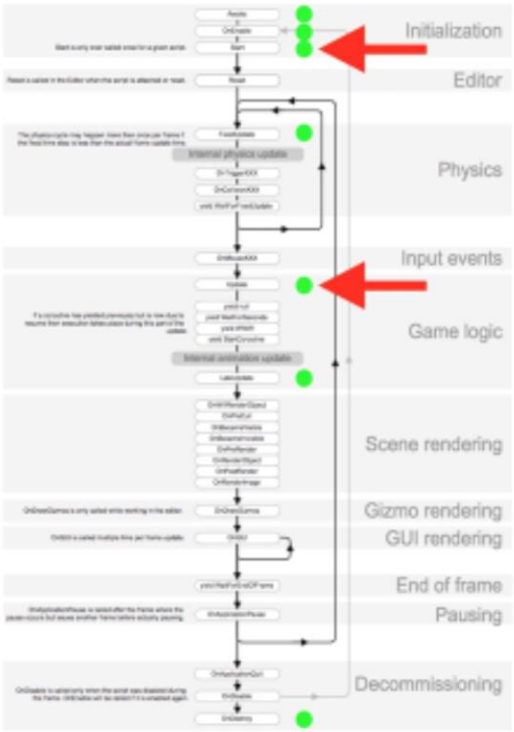
11

ПРАКТИЧНИЙ РЕЗУЛЬТАТ

Розроблено тестове середовище, яке є спрощеною стратегією в реальному часі з обмеженою кількістю типів військ та дещо спрощеними ігровими правилами.

Для роботи в тестовому середовищі створено штучний інтелект для прийняття тактичних та стратегічних рішень в управлінні. В моделі роботи штучного інтелекту активно використовувались підходи, аналогічні тим, що використовуються в згорткових нейронних мережах.

На рисунку зеленим позначено методи, доступні для використання у скриптах. Якщо додати логіку в ці методи Unity виконає її на етапі життєвого циклу, до якого відноситься конкретний метод. Червоними стрілками позначено методи, які по замовчуванню створюються у всіх класах, які наслідують MonoBehaviour.



12

РЕЗУЛЬТАТИ МОДЕЛЮВАННЯ

За результатами тестування можна визначити, що штучний інтелект досить ефективний проти ботів та є гнучким супротивником. Для подальшого покращення результатів потрібно збільшити обсяги тестових даних та урізноманітнити їх.

Таблиця 5.1 – Оцінки ефективності

	Атака	Захист	Баланс
1	67%	61%	64%
2	72%	65%	71%
3	75%	66%	73%

Основною метрикою, з допомогою якої вимірювалась ефективність системи, була оцінка ефективності використання бойових одиниць.

Для тримання більш ефективних результатів я використовував три різних моделі навчання нейронних мереж:

- Перший працював під управлінням нейронної мережі де вріний результат задавався вручну;
- Другий працював під управлінням нейронної мережі з використанням власних даних після їх обробки;
- Загально прийнятий алгоритм дій для ботів.

13

ВИСНОВКИ

У результаті виконання роботи було досліджено актуальність обраної теми. Розглянуто та проаналізовано існуючі рішення та підходи до розробки штучного інтелекту для стратегій в реальному часі. На основі проаналізованої інформації обрано оптимальний варіант комбінації агентів для створення наближеного до людини та потужного штучного інтелекту. Спроектовано реалізації обраних методів, дано розгорнуте порівняння та опис переваг саме такого підходу для створення інтелекту.

Обрано платформу та технології для створення тестового середовища, необхідного для перевірки штучного інтелекту на реальному продукті. Розроблено структуру серверної частини, обгрунтовано використання мікросервісного підходу для забезпечення стійкості системи.

Проаналізовано сучасні методи для створення та роботи з нейронними мережами, обгрунтовано використання нейронної мережі для додатку

Розглянуто переваги побудови штучного інтелекту для стратегій з використанням ієрархічної мережі постановки задач. Створено ієрархічну мережу для гри в рамках тестового середовища, проаналізовано результати роботи мережі.

14

АПРОБАЦІЯ РОБОТИ

Статті:

1. Борук В.Ю. **«Прийняття стратегічних рішень»**

Тези доповідей:

1. Борук В.Ю.**Розвиток штучного інтелекту для стратегій в реальному часі** // XIII Міжнародна науково-технічна конференція студентства та молоді «СВІТ ІНФОРМАЦІЇ ТА ТЕЛЕКОМУНІКАЦІЙ» – Київ: ДУТ, 2021.
2. 1. Борук В.Ю. **Використання нейронної мережі для тактичного штучного інтелекту в іграх жанру стратегії в реальному часі** // XIII Міжнародна науково-технічна конференція студентства та молоді «СВІТ ІНФОРМАЦІЇ ТА ТЕЛЕКОМУНІКАЦІЙ» – Київ: ДУТ, 2021.

15

ДЯКУЮ ЗА УВАГУ!

16

ДОДАТОК Б

Публікації

1. Борук В.Ю.Розвиток штучного інтелекту для стратегій в реальному часі // XIII Міжнародна науково-технічна конференція студентства та молоді «СВІТ ІНФОРМАЦІЇ ТА ТЕЛЕКОМУНІКАЦІЙ» – Київ: ДУТ, 2021.

2. Борук В.Ю. Використання нейронної мережі для тактичного штучного інтелекту в іграх жанру стратегії в реальному часі // XIII Міжнародна науково-технічна конференція студентства та молоді «СВІТ ІНФОРМАЦІЇ ТА ТЕЛЕКОМУНІКАЦІЙ» – Київ: ДУТ, 2021.

ДОДАТОК В

Вихідний код застосунку

```
using System.Collections.Generic; using TestEntertainment;
using Test Entertainment; using UnityEngine.AI;

public abstract class Person : MonoBehaviour, IUnit
{
[SerializeField] public NavMeshAgent agent; [SerializeField] public GameObject pointer; [SerializeField] public float baseMoveSpeed; [SerializeField] public float baseRotation-Speed = 10; public Guid Id { get; private set; }
public float Health { get; set; }
public GameObject Owner { get; set; } public float Speed
{
get => baseMoveSpeed; set { }
}

public UnitState CurrentState { get; set; }
public abstract bool SetState(UnitState newState); public abstract string GetTypeName();

protected Vector3? targetPos; protected Animator animator;

private void Awake()
{
Id = Guid.NewGuid();
animator = GetComponent<Animator>();
}

// Start is called before the first frame update void Start()
{
pointer.SetActive(false);
}

// Update is called once per frame

void Update()
{
var distance = Vector3.Distance(targetPos.GetValueOrDefault(), transform.position);
if (targetPos != null && distance > 0.7f)
{
```

```
var position = gameObject.transform.position;
var simpleVector = targetPos.GetValueOrDefault() - position;

if (false && RotateToTarget(simpleVector))
{
SetState(UnitState.FastMove);
var moveVector = simpleVector / distance * (Time.deltaTime * Speed); transform.position
+= moveVector;
}

SetState(UnitState.FastMove); agent.SetDestination(targetPos.Value);
}
else
{
targetPos = null; SetState(UnitState.Stay);
}
}

public bool RotateToTarget(Vector3 simpleVector)
{
var targetAngle = Vector3.Angle(Vector3.forward, simpleVector.normalized); if (targetPos
!= null && targetPos.Value.x < transform.position.x)
{
targetAngle = 360 - targetAngle;
}

var delta = Mathf.DeltaAngle(transform.eulerAngles.y, targetAngle);

if (Mathf.Abs(delta) > 3)
{
SetState(UnitState.Move);

var rotation = Time.deltaTime * baseRotationSpeed; var isLeftMove = delta < 0;

var eulerAngles = transform.eulerAngles;

eulerAngles = new Vector3(eulerAngles.x, eulerAngles.y + (isLeftMove ? - rotation : rota-
tion),
eulerAngles.z); transform.eulerAngles = eulerAngles;

return false;
```

```
}

return true;
}


public void Select()
{
pointer.SetActive(true);
}

public void Unselect()
{
pointer.SetActive(false);
}

public void MoveToPoint(Vector3 point)
{
targetPos = point;
}

public void Attack()
{
throw new NotImplementedException();
}

public void ReceiveDamage()
{
throw new NotImplementedException();
}

public void SetMaterial()
{
throw new NotImplementedException();
}

}

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

```
using Common; using Controls.Units; using UnityEngine;
using UnityEngine.EventSystems; using UnityEngine.SceneManagement; using Object = UnityEngine.Object;

public class UnitController : MonoBehaviour
{
[SerializeField] private new Camera camera; [SerializeField] private LayerMask unitLayer-Mask; [SerializeField] private LayerMask terrainLayerMask;

private Player _owner; private float _lastClickTime;
private const float CatchTime = .25f;

// private (GameObject, IUnit)? _selectedUnit = null;
private readonly List<ControlledUnit> _controlledUnits = new List<ControlledUnit>();
private readonly List<ControlledUnit> _allUnits = new List<ControlledUnit>(); #region MultiSelect
private bool _multiSelect = false;
public Vector3? MultiSelectPoint { get; set; }

public bool MultiSelect
{
get => _multiSelect; set
{
Debug.Log($"Multi set to: {value}"); if (value && !_multiSelect)
{
ClickUnselect();
}

_multiSelect = value;
}
}


#endregion

#region UIHandlers

public void ClickMultiSelect()
{
MultiSelect = !MultiSelect;
}
```

```
public void ClickUnselect()
{
// TODO: UI need to be updated UnselectAll();
}

#endregion

private void Awake()
{
_allUnits.AddRange( Utils.FindGameObjectsWithLayer(8)
.Where(x => x.TryGetComponent(out IUnit unit))
.Select(x => new ControlledUnit(x.GetComponent<IUnit>(), x, x.GetComponent<IUnit>().Id))
.ToList());
}

private void Start()
{
_owner = Player.players.FirstOrDefault(x => x.IsCurrentPlayer);
}

private void Update()
{
if (!EventSystem.current.IsPointerOverGameObject() && Input.GetMouseButtonUp(0) && Input.touchCount == 0)
{
var isDoubleClick = Time.time - _lastClickTime < CatchTime; Debug.Log("Catch Double Click: " + isDoubleClick);
var ray = camera.ScreenPointToRay(Input.mousePosition);
if (!MultiSelect && Physics.Raycast(ray, out var hit, Mathf.Infinity, unitLayerMask))
{
HandleUnitClick(hit, isDoubleClick);
}

else if (!MultiSelect && Physics.Raycast(ray, out var hitGround, Mathf.Infinity, terrainLayerMask))
{
HandleMoveToPoint(hitGround);
}
```

```
else if (MultiSelect && Physics.Raycast(ray, out var selectGround, Mathf.Infinity, terrain-
LayerMask))
{
HandleMultiSelection(selectGround);
}
_lastClickTime = Time.time;
}
}

private void HandleUnitClick(RaycastHit hit, bool isDoubleClick)
{
var unit = hit.transform.gameObject.GetComponent<IUnit>(); if (unit != null)
{
var cUnit = _allUnits.Find(x => unit.Id == x.UnitId); var alreadySelected = _controlledUnits
.FirstOrDefault(x => x.UnitId == cUnit.UnitId) != null; if (!_controlledUnits.Any())
{
AddUnitToSelection(cUnit);
}
else if (alreadySelected)
{
UnselectAll();
Debug.Log(("cUnit.LastSelectedTime; " + cUnit.LastSelectedTime));
if (isDoubleClick && Time.time - cUnit.LastSelectedTime <= CatchTime)
{
AddUnitsToSelection(_allUnits.Where(x => x.Unit.GetTypeName() == unit.GetType-
Name() && Vector3.Distance(x.Object.transform.position,
cUnit.Object.transform.position) < 20));
}
else
{
AddUnitToSelection(cUnit);
}
}
else
{
UnselectAll();

AddUnitToSelection(cUnit);
}
}
}
```

```
#region Helpers

private void AddUnitToSelection(ControlledUnit cUnit)
{
_controlledUnits.Add(cUnit); cUnit.LastSelectedTime = Time.time; cUnit.Unit.Select();
}

private void AddUnitsToSelection(IEnumerable<ControlledUnit> cUnits)
{
var controlledUnits = cUnits.ToList();
_controlledUnits.AddRange(controlledUnits); foreach (var cUnit in controlledUnits)
{
cUnit.LastSelectedTime = Time.time; cUnit.Unit.Select();
}
}

private void UnselectAll()
{
_controlledUnits.ForEach(x => { x.Unit.Unselect(); });
_controlledUnits.Clear();
}

private void UnselectById(Guid id)
{
// TODO: probably can remove null-check and improve performance for this method
var unitToUnselect = _controlledUnits.FirstOrDefault(x => x.UnitId == id); if (unitToUnse-
lect == null)
{
Debug.Log("Check UnselectById method!");
}
else
{
unitToUnselect.Unit.Unselect();
_controlledUnits.Remove(unitToUnselect);
}

}

private Vector3 GetMultiMovePositionModifier(int index)
{
```

```
if (index < 5)
{
return MultiMoveCombinator(index);
}

switch (index)
{
case 5: return MultiMoveCombinator(1, 2);
case 6: return MultiMoveCombinator(2, 3);
case 7: return MultiMoveCombinator(3, 4);
case 8: return MultiMoveCombinator(4, 1);
case 9: return MultiMoveCombinator(1, 1);
case 10: return MultiMoveCombinator(2, 2);
case 11: return MultiMoveCombinator(3, 3);
case 12: return MultiMoveCombinator(4, 4);
case 13: return MultiMoveCombinator(1, 1, 2, 2);
case 14: return MultiMoveCombinator(2, 2, 3, 3);
case 15: return MultiMoveCombinator(3, 3, 4, 4);
case 16: return MultiMoveCombinator(4, 4, 1, 1);
case 17: return MultiMoveCombinator(1, 1, 2);
case 18: return MultiMoveCombinator(1, 2, 2);
case 19: return MultiMoveCombinator(2, 2, 3);
case 20: return MultiMoveCombinator(3, 3, 2);
case 21: return MultiMoveCombinator(3, 3, 4);
case 22: return MultiMoveCombinator(4, 4, 3);
case 23: return MultiMoveCombinator(4, 4, 1);
case 24: return MultiMoveCombinator(1, 1, 4); default: throw new NotImplementedExcep-
tion();
}
}

private Vector3 MultiMoveCombinator(params int[] ar)
{
Vector3 Directions(int i)
{
switch (i)
{
case 0: return Vector3.zero; case 1: return Vector3.left; case 2: return Vector3.forward; case
3: return Vector3.right;

case 4: return Vector3.back;
```

```
default: throw new NotImplementedException();
}
}
return ar.Aggregate(Vector3.zero, (a1, a2) => a1 + Directions(a2));
}

#endregion

private void HandleMultiSelection(RaycastHit selectGround)
{
if (MultiSelectPoint is null)
{
MultiSelectPoint = selectGround.point;
}
else
{
// select all units
var xs = new[] {MultiSelectPoint.Value.x, selectGround.point.x}.OrderBy(x => x).ToArray();
var zs = new[] {MultiSelectPoint.Value.z, selectGround.point.z}.OrderBy(z => z).ToArray();

AddUnitsToSelection(_allUnits
.Where(unit =>
{
var position = unit.Object.transform.position;
return position.x >= xs[0] && position.x <= xs[1] && position.z >= zs[0] && position.z <= zs[1];
}));

MultiSelectPoint = null; MultiSelect = false;
}
}

private void HandleMoveToPoint(RaycastHit hitGround)
{
var pointToMove = Vector3Int.CeilToInt(hitGround.point); pointToMove.y = 0;
const int unitSize = 3;
for (var i = 0; i < _controlledUnits.Count; i++)
{
_controlledUnits[i].Unit.MoveToPoint( pointToMove +
```

```
GetMultiMovePositionModifier(i % 25) * unitSize + MultiMoveCombinator(i / 25) * (5 *
unitSize));
}
}
}

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

public class Building : MonoBehaviour
{
public Vector2Int size = Vector2Int.one;

public GameObject state1; public GameObject state2; public GameObject state3;

private Renderer _renderer; private protected bool Ready;

private void Awake()
{
_renderer = GetComponentInChildren<Renderer>();
}

public void SetTransparent(bool available)
{
_renderer.material.color = available ? Color.green : Color.red;
}

public void OnBuildingSet()
{
_renderer.material.color = Color.white; StartCoroutine(Build());
}

private void OnDrawGizmos()
{
for (int x = 0; x < size.x; x++)
{
for (int y = 0; y < size.y; y++)
{
Gizmos.color = (x + y) % 2 == 0 ? Color.cyan : Color.gray;
```

```
Gizmos.DrawCube(transform.position + new Vector3(x, 0, y),new Vector3(1, .2f ,1));
}
}
}

public IEnumerator Build()
{
state3.SetActive(false); state1.SetActive(true);

yield return new WaitForSeconds(3); state2.SetActive(true); state1.SetActive(false);

yield return new WaitForSeconds(3); state3.SetActive(true); state2.SetActive(false);

Ready = true;
}
}
```