МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

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

КАФЕДРА ЕЛЕКТРОНІКИ, ІНФОРМАЦІЙНИХ СИСТЕМ ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Кваліфікаційна робота

другий (магістерський)
(рівень вищої освіти)

на тему **Штучний інтелект в ігрових програмах з використанням машинного навчання**

Виконав: студент 2 курсу, групи 8.1211-2іпз
спеціальності 121 Інженерія програмного забезпечення
(код і назва спеціальності)
освітньої програми Інженерія програмного забезпечення
(код і назва освітньої програми)
С.В. Островецький
(підпис, ініціали та прізвище)
Керівник доцент А.І. Безверхий
(посада, вчене звання, науковий ступінь, підпис, ініціали та прізвище)
Рецензент директор ТОВ Дискус П.О. Лютий
(посада, вчене звання, науковий ступінь, підпис, ініціали та прізвище)

Запоріжжя
2022

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

Кафедра Електроніки, інформаційних систем та програмного забезпечення

Рівень вищої освіти _____другий (магістерський)_____

Спеціальність 121 Інженерія програмного забезпечення_____
(код та назва)

Освітня програма ___Інженерія програмного забезпечення
(код та назва)

ЗАТВЕРДЖУЮ

Завідувач кафедри_____Т.В. Критська_____

“_12_” __вересня_____2022_року

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

Островецькому Сергію Володимировичу
(прізвище, ім’я, по батькові)

1.Тема роботи __Штучний інтелект в ігрових програмах з використанням машинного навчання_____

керівник роботи ___Безверхий Анатолій Ігорович, доцент
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від 02.06.2022 р. №597-с

2. Строк подання студентом кваліфікаційної роботи 1 грудня 2022 р.

3. Вихідні дані магістерської роботи

- комплект нормативних документів ;
- технічне завдання до роботи.

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

- огляд та збір літератури стосовно теми кваліфікаційної роботи;
- огляд та аналіз існуючих рішень та аналогів;
- дослідження проблеми використання штучного інтелекту в ігрових застосунках з використанням машинного навчання;
- створення програмного продукту та його опис;
- перелік вимог для роботи програми;
- дослідження поставленої проблеми та розробка висновків та пропозицій.

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

слайдів презентації

6. Консультанти розділів магістерської роботи

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

7. Дата видачі завдання 12.09.2022

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів магістерської роботи	Строк виконання етапів магістерської роботи	Примітка
1	Аналіз предметної області	02.06-17.06.2022	виконано
2	Формулювання основної задачі дипломної роботи та узгодження її з науковим керівником	18.06-19.06.2022	виконано
3	Аналіз існуючих рішень	20.06-8.07.2022	виконано
4	Дослідження сучасних методів реалізації ігрового штучного інтелекту та методів машинного навчання	09.07-20.07.2022	виконано
5	Розробка ігрового застосунку	21.07-21.09.2022	виконано
6	Розробка модулів прикладів з різними методами навчання та їх аналіз на предмет інтеграції в ігровий застосунок	22.09-07.10.2022	виконано
7	Узгодження подальших дій з науковим керівником	08.10-9.10.2022	виконано
8	Розробка системи навчання за допомогою методів Q-навчання	10.10-28.10.2022	виконано
9	Навчання системи керування юнітами за допомогою методів Q-навчання та тестування навченої системи в різних умовах	29.10-07.11.2022	виконано
10	Представлення отриманих результатів науковому керівнику та узгодження плану подальшого дослідження	08.11-09.11.2022	виконано
11	Аналіз отриманих результатів дослідження	10.11-17.11.2022	виконано
12	Оцінка доцільності використання машинного навчання при розробці штучного інтелекту для гри	18.11-22.11.2022	виконано
13	Оформлення звіту	22.11-30.11.2022	виконано

Студент _____ С.В. Островецький
(підпис) (ініціали та прізвище)

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

Нормоконтроль пройдено

Нормоконтролер _____ І.А. Скрипник
(підпис) (ініціали та прізвище)

АНОТАЦІЯ

Сторінок: 97

Рисунків: 46

Джерел: 30

Островецький С. В. Штучний інтелект в ігрових програмах з використанням машинного навчання: кваліфікаційна робота магістра спеціальності 121 «Інженерія програмного забезпечення» / наук. керівник А. І. Безверхий. Запоріжжя: ЗНУ, 2022. 97 с.

Мета і завдання дослідження полягають у розробці ігрового застосунку з використанням методів штучного інтелекту та навчання з підкріпленням. Визначити переваги та недоліки використання систем штучного інтелекту в ігрових застосунках, обрати найбільш придатний для цього варіант, і зрозуміти раціональність їх використання.

У процесі дослідження були розглянуті види систем штучного інтелекту, варіанти їх інтеграції в ігровий застосунок та задачі які вирішуються за їх допомогою в застосунках такого типу. Даний додаток є ігровим застосунком, в якому за допомогою систем штучного інтелекту імітується інтелектуальна поведінки персонажів, яка виражається в обрані ефективного шляху розвитку та гнучкість цієї поведінки в умовах, які змінюються з часом.

Ключові слова: *ігровий штучний інтелект, нейрона мережа, навчання без вчителя, стратегічна гра, QLearning, Unity, C#.*

SUMMARY

Pages: 97

Figures: 46

Sources: 30

Ostrovetskyi S.V. Artificial intelligence in game programs using machine learning: qualification work of the master of specialty 121 "Software Engineering" / science. head A.I. Bezverkhy. Zaporizhzhia: ZNU, 2022. 97 p.

The purpose and objectives of the study are to develop a game application using methods of artificial intelligence and learning with reinforcement. Identify the advantages and disadvantages of using artificial intelligence systems in gaming applications, choose the most suitable option for this, and understand the rationality of their use.

In the course of the research the types of artificial intelligence systems, variants of their integration into a game application and the problems that are solved with their help in applications of this type were considered. This application is a game application in which artificial intelligence systems simulate the intellectual behavior of the characters, which is expressed in the chosen effective path of development and flexibility of this behavior in conditions that change over time.

Keywords: *game artificial intelligence, neural network, learning without a teacher, strategy game, QLearning, Unity, C#.*

ЗМІСТ

ВСТУП

Актуальність теми

У сучасному світі ігрова індустрія займає значний пласт індустрії розваг. Зараз кількість ігрових застосунків є майже незліченною, тому сучасні ігри, на відміну від їх перших представників, мають величезну конкуренцію, а значить вимоги є значно вищими ніж раніше. Також росту вимог сприяє технічний розвиток, сучасні комп'ютери, смартфони та ігрові приставки мають такі потужності, які раніше були лише мріями і завдяки цьому надають більше можливостей для реалізації нових механік в іграх, покращення якості зображень, моделей та аудіо.

Комп'ютерні ігри еволюціонували від простого переносу настільних ігор та таких, що мали якусь елементарну механіку до цілих світів, які мають складну структуру та надають величезну кількість можливостей гравцю.

Крім підвищених вимог до технічної складової ігрових застосунків також зросли вимоги до їх наповнення. Є велика кількість різноманітних жанрів ігор, які забезпечують задоволення тих бажань, які важливі для кожного окремого гравця. Кожна з них надає ту систему, яка необхідна для певних цілей:

- екшен-ігри для тих хто полюбляє багато дій на екрані та отримує більше задоволення від візуалу та самого процесу гри, ніж від результатів;
- більш сюжетні ігри, такі як, наприклад, візуальні новели та інші, для тих хто більше цінує історію, яку розповідає гра.

В незалежності від жанру основним завданням є «занурення» гравця в ігровий процес. Одним з факторів цього є поведінка неігрових персонажів. Зараз щоб не порушувати цього «занурення» гравцями очікується інтелектуальна поведінка персонажів, або, як це є насправді, гарна імітація такої поведінки.

Зазвичай, в сучасних іграх поведінка персонажів все ще описується за допомогою набору умовних операторів або дерев станів та правилам переходів

між ними. Даний підхід є досить ефективним з точки зору споживання ресурсів, а якщо таке дерево має велику кількість вузлів, то забезпечує достатню імітацію інтелектуальної поведінки. Але якщо якимсь чином персонаж потрапляє в ситуацію не передбачену ситуацію, то він або зовсім «зламається» та перестане якимсь чином реагувати на дії гравця та ситуацію навколо нього, або почне поводити себе алогічно. В такій ситуації про «занурення» гравця можна забути.

Варіантом вирішення даної проблеми є використання систем штучного інтелекту, нейронних мереж та різних підходів їх навчання.

Такий підхід не використовується повсюдно через деякі складності його реалізації. До таких відносяться:

- складність визначення моделі по якій буде навчатися система;
- «чорний ящик» прийняття рішень такою системою;
- довгий процес навчання та необхідність перенавчення при змінні вхідних даних.

Однак, в ньому є і переваги:

- прийняття рішень такою системою є швидким та ресурсно-економічним;
- система є гнучкою та не втрачає можливості приймати певні рішення навіть в ситуаціях в яких вона не навчалася;
- така система може надати новий ігровий досвід гравцю.

Одним з способів навчання такої системи є навчання без вчителя з підкріпленням.

В такому випадку будується система, яка складається з середовища, агентів та системи дій та нагородження за них.

Перевагою даного підходу до навчання в цілому є те, що вона навчається самостійно без необхідності збору датасетів, їх класифікації або розмітки. Важливо, щоб алгоритм вирішив задачу в цілому, а що конкретно він буде робити в процесі вирішення цього завдання і яким шляхом він піде, щоб її вирішити, не є принциповим — від нього очікується тільки кінцевий результат.

Перевагою даного підходу до навчання саме в рамках ігрових застосунків є те, що середовище в якому навчаються агенти, може використовуватися в самій грі, що дозволяє під час навчання наочно бачити, як система працює в таких умовах та зекономити ресурси необхідні на побудову застосунку в цілому.

Мета і завдання дослідження

Мета і завдання дослідження полягають у розробці ігрового застосунку з використанням методів штучного інтелекту та навчання з підкріпленням, визначити переваги та недоліки використання систем штучного інтелекту в ігрових застосунках, обрати найбільш придатний для цього варіант, і зрозуміти раціональність їх використання.

Об’єкт дослідження

Об’єктом дослідження є ігровий застосунок в якому використовуються системи штучного інтелекту навчені за допомогою навчання з підкріпленням.

Предмет дослідження

Предметом дослідження є визначення переваг та недоліків використання систем штучного навчання в ігрових застосунках та визначення оптимальних методів їх навчання в рамках розробки гри.

Методи дослідження

Для вирішення поставленої задачі використовуються наступні методи дослідження:

1. Аналіз особливостей та існуючих рішень, які використовують системи штучного інтелекту та без них.
2. Аналіз переваг та недоліків кожного з підходів.
3. Аналіз різноманітних бібліотек, призначених для реалізації.

4. Аналіз різних алгоритмів побудови та навчання систем штучного інтелекту.
5. Експериментування з використанням різних середовищ для навчання нейронної мережі.
6. Експериментування зі зміною параметрів роботи розробленої системи.

Наукова новизна одержаних результатів

Наукова новизна одержаних результатів дослідження полягає у тому, що під час розробки ігрового застосунку з використанням систем штучного інтелекту було розширено спектр рішень, які не були закладенні в програму, а були результатами навчання з підкріпленням, щоб в майбутньому стати підґрунтям для більшої інтеграції даних підходів в ігрову індустрію, як нова ефективна альтернатива підходам, які зазвичай використовуються.

Практичне значення одержаних результатів

Практичне значення одержаних результатів дослідження полягає у тому, що на їх основі був розроблений ігровий застосунок з використанням систем штучного інтелекту з використанням алгоритмів навчання з підкріпленням. Даний застосунок дозволяє дослідити всі переваги і недоліки вибору даного підходу до побудови ігрової програми в цілому та раціональність використання даного підходу в рамках питання витрат на нього та нових можливостей, які він надає. З цього можна зробити висновок, що представлений у роботі підхід є раціональним та ефективним для ігрових застосунків певних жанрів, а саме представленою в рамках роботи стратегічних ігор, але не обмежуючись ними.

Апробація одержаних результатів

Результати дослідження були представлені на І всеукраїнській науково-практичній конференції здобувачів вищої освіти, аспірантів та молодих вче-

них «Актуальні питання сталого науково-технічного та соціально-економічного розвитку регіонів України»[**Помилка! Джерело посилання не знайдено.**], на науково-технічній конференції студентів, аспірантів, магістрантів і викладачів Інженерного навчально-наукового інституту Запорізького національного університету «Молода наука-2022» [2], на ІІ всеукраїнській науково-практичній конференції «Актуальні питання сталого науково-технічного та соціально-економічного розвитку регіонів України» [3] та на міжнародній науково-практичній конференції Інженерного навчально-наукового інституту ім. Ю.М. Потебні Запорізького національного університету «Перспективи сталого розвитку в умовах глобалізації в економічному, управлінському та інженерному аспектах» [4].

Глосарій

Штучні нейронні мережі — це обчислювальні системи, що навчаються задач (поступально покращують свою продуктивність на них), розглядаючи приклади, загалом без спеціального програмування під задачу.

Ігровий штучний інтелект (ІШІ) (англ. Game artificial intelligence) — набір методів, які використовуються для побудови програмних систем так, щоб імітувати інтелектуальну поведінку персонажів, які керуються комп’ютером.

Машинне навчання (англ. machine learning) — це підгалузь штучного інтелекту в галузі інформатики, яка часто застосовує статистичні прийоми для надання комп'ютерам здатності «навчатися» (тобто, поступово покращувати продуктивність у певній задачі) з даних, без того, щоби бути програмованими явно.

Навчання без вчителя — це спосіб машинного навчання, при якому система, яка навчається, робить це без втручання експериментатора. Даний спосіб навчання відрізняється тим, що системі строго не задаються правильні відповіді і системі залишається лише знайти залежність між входами та результатами.

Стратегічна відеогра — жанр відеоігор, в якому запорукою досягнення перемоги є планування і стратегічне мислення. Можуть містити різну тематику: зокрема військових (Total War), економічних (Caesar), суспільствознавчих (Civilization) симуляторів тощо.

Тайли — невеликі зображення однакових розмірів, що є фрагментами великої картини. Кількість тайлів на один світ може досягати декількох сотень. Матриця клітин у своїй зберігає лише номери тайлів, за рахунок чого досягається економія пам'яті при побудові великих двомірних просторів.

Сабтайл — уявна проміжна частина між двома квадратними тайлами з’єднаними по діагоналі. Їх використання дозволяє отримати з мапи сформованої квадратними тайлами мапу з восьмикутників та проміжних сабтайлів ромбовидної форми.

Технологічне дерево — структура, що симулює процес технологічного розвитку в іграх детермінованим чином. Вона визначає переходи від одних технологій до інших, досконаліших, які розширюють можливості гравця — дозволяють формувати потужніші армії, будувати нові споруди тощо.

Ігровий рушій (англ. Game engine) — програмний рушій, центральна програмна частина будь-якої відеогри, яка відповідає за всю її технічну сторону, дозволяє полегшити розробку гри шляхом уніфікації та систематизації її внутрішньої структури. Важливим значенням рушія є можливість створення багатоплатформових ігор. Основну функціональність гри зазвичай забезпечує її рушій, до якого входить рушій рендерингу («візуалізатор»), фізичний рушій, звук, система скриптів, анімація, ігровий штучний інтелект, мережевий код, керування пам'яттю, багатонитевість і граф сцени. Часто на процесі розробки можна заощадити шляхом повторного використання одного рушія гри для створення декількох різних ігор.

Неігровий персонаж або негральний персонаж (англ. Non-Player Character, NPC) в комп’ютерних і настільних рольових іграх — персонаж, керований програмою або майстром, в останньому випадку іноді може називатися майстерним персонажем. У комп'ютерних та настільних рольових іграх

терміном NPC позначають персонажів, що спілкуються із гравцем, незалежно від їхнього ставлення до гравця. NPC можуть бути дружніми, нейтральними та ворожими. Неігрові персонажі слугують важливим засобом створення ігрової атмосфери, вони мотивують гравців робити ті чи інші дії, є основним джерелом інформації про ігровий світ та сюжет гри.

Система керування NPC — система, яка не має візуального відображення в грі, зазвичай має більше інформації про середовище в якому знаходиться та використовується для надання наказів на певну дію неігровим персонажам з пулу доступних їм дій.

Q-навчання (англ. Q-learning) — це алгоритм безмодельного навчання з підкріпленням. Метою Q-навчання є навчитися стратегії, яка каже агентові, до якої дії вдаватися за яких обставин. Воно не вимагає моделі середовища (звідси уточнення «безмодельного»), і може розв'язувати задачі зі стохастичними переходами та винагородами, не вимагаючи пристосувань.

РОЗДІЛ 1 ДОСЛІДЖЕННЯ ВИКОРИСТАННЯ СИСТЕМ ШТУЧНОГО ІНТЕЛЕКТУ В ІГРОВИХ ЗАСТОСУНКАХ. НАВЧАННЯ З ПІДКРІПЛЕННЯМ

1.1 Огляд проблеми

Завдання штучного інтелекту полягає в «сприйнятті» (наприклад, отримання зображення поверхні, що знаходиться перед об’єктом та намагання інтерпретувати побачене) і прийнятті, на основі сприйнятого, обміркованого рішення.

Для того, щоб штучний інтелект міг приймати зважені рішення, йому потрібно якимось чином сприймати середовище, в якому він знаходиться. У простих системах таке сприйняття може бути обмежене простою перевіркою положення об'єкта гравця. У більш складних системах необхідно визначити основні характеристики і властивості ігрового світу, наприклад, можливі маршрути для пересування, наявність природних укриттів на місцевості, зон конфліктів і т. д. При цьому розробники повинні придумати спосіб ідентифікації та визначення основних властивостей ігрового світу, які важливі для системи ІШІ. Наприклад, укриття місцевості можуть бути заздалегідь визначені дизайнерами рівнів або попередньо розраховані під час завантаження чи компіляції карти рівнів. Деякі елементи потрібно обчислювати в реальному часі, наприклад карти конфліктів і найближчі загрози.

Проблема традиційних ІШІ в тому, що гравець може помітити заскриптованість дій, що може зруйнувати його зануреність в ігровий процес та відповідно може негативно впливати на зацікавленість продовжувати гру, що безпосередньо впливає на її успіх.

Також визначення гравцем алгоритму дій NPC або систем керування може призвести до таких дій гравцем, що будуть направлені на отримання вигоди в грі по не передбачуваному та не обробленому розробниками сценарію.

Добре продуманий ІШІ дозволяє NPC поводитися більш реалістично, що робить ігровий процес більш реалістичним і занурює гравця в гру [5].

1.2 Сучасні підходи реалізації ІШІ

Існують різні методи реалізації ІШІ. Розглянемо найпопулярніші з них.

Система на основі правил. Найпростішою формою ІШІ є система, заснована на правилах. Таким чином, система найвіддаленіша від справжнього штучного інтелекту. Набір попередньо визначених алгоритмів, які визначають поведінку ігрових об’єктів. Враховуючи різноманітність дій, кінцевим результатом може бути неявна система поведінки, хоча така система зовсім не буде «розумною». Системи на основі правил є основою ІШІ [6].

Скінчені автомати. Скінченний автомат (машина зі скінченною кількістю станів) — це метод моделювання та реалізації об’єкта, який має різні стани протягом свого життя. Кожен «стан» може представляти фізичні умови, в яких перебуває об’єкт, або, наприклад, набір емоцій, які об’єкт виражає. Тут емоційні стани не мають нічого спільного з емоціями ІШІ, вони належать до наперед визначених поведінкових патернів, які вписуються в контекст гри [7].

Адаптивний ІШІ. Якщо грі потрібно більше різноманітності, якщо гравець повинен бути сильнішим і динамічнішим супротивником, AI повинен мати здатність розвиватися, адаптуватися і адаптуватися. Для адаптивної системи вкрай важлива здатність точно передбачити наступний хід противника. Для вибору наступного ходу можна використовувати різні техніки, наприклад розпізнавання шаблонів минулих ходів або випадкові припущення [8].

Один із найпростіших способів адаптації — відстежувати та оцінювати успішні рішення. Система ІШІ записує вибір, зроблений гравцем у минулому. Усі рішення, прийняті в минулому, потрібно певним чином оцінювати, наприклад, у бойових іграх як мірило успіху можна обрати шкоду яку наніс гравець. Ви можете використовувати отримання або втрату переваги, втрату здоров’я або часу. Наприклад, ви можете зібрати додаткову інформацію про ситуацію,

щоб забезпечити контекст для прийняття рішень, як-от відносний стан здоров’я, минулі дії та посади на рівні. Адаптивний ШІ сприймає зовнішній світ через «зір», «слух», «дотик», «нюх» тощо.

Віртуальні світи, в яких відбувається більшість ігор, мають величезну перевагу над реальним світом з точки зору ШІ та його сприйняття. На відміну від реального світу, ви заздалегідь знаєте кількість буквально всього, що існує у віртуальному світі: десь серед ресурсів гри є список, у якому перераховано все, що існує в грі. Ви можете шукати цей список за своїм запитом і миттєво отримувати інформацію, яку штучний інтелект може використовувати для прийняття більш розумних рішень. Ви можете або зупинитися на першому об’єкті, який цікавить ШІ, або отримати список усіх об’єктів у заданому діапазоні, щоб ШІ міг прийняти найкраще рішення щодо навколишнього світу. Так ІШІ «бачить» об’єкти навколо.

Проте кожна дія, яку може виконати об’єкт, пов’язана з певним рівнем звуку. Можна встановити рівні звуку заздалегідь (для оптимізації ігрового балансу) або розрахувати їх на основі фактичної енергії звукових ефектів, пов’язаних з певними діями (це забезпечує високий рівень реалістичності, але навряд чи є необхідним). Якщо звук голосніший за порогове значення, ІШІ «почує» об’єкт, який видає звук.

Можна використовувати основні функції, необхідні для надання ІШІ «зору» та «слуху», а також для імітації інших органів чуття (наприклад, нюху). Додавання чуття в гру не викликає особливих труднощів: достатньо присвоїти кожному ігровому об'єкту характерний номер запаху та його інтенсивність. Інтенсивність запаху визначається двома факторами: радіусом запаху та силою запахового сліду, який залишається за собою. Об’єкти активного гравця часто відстежують свої попередні позиції з кількох причин. Однією з таких причин може бути використання предметів із запахом. З часом сила запаху сліду зменшується, слід «остигає».

Дотик в іграх зберігається з самого початку, тому що в будь-якій грі вже є система автоматичної обробки зіткнень об'єктів. Досить переконатися, що ІШІ реагує на події зіткнення та пошкодження.

У багатьох шутерах адаптивний ШІ дозволяє ворогам ховатися за найближчим укриттям, а не просто стояти під вогнем на відкритому повітрі. Для реалізації необхідно:

- по-перше, правильно розпізнати укриття за геометрією навколишнього світу;
- по-друге, необхідно правильно визначити укриття на основі інформації про об'єкти по всьому світу.

Щоб визначити, чи здатне укриття захистити від атак, ви можете просто один раз порівняти розмір обмежувальної рамки персонажа з розміром можливого укриття. Потім ви повинні перевірити, чи може персонаж використовувати це укриття. Для цього проведіть промені від положення стрілка до укриття. Цей промінь можна використовувати, щоб визначити, чи місце за укриттям «захищене» (якщо дивитися збоку від стрілка), а потім позначити це місце як наступну ціль, до якої потрібно перемістити персонажа.

Навігація відіграє важливу роль в адаптивному ІШІ. Після того, як рішення прийнято, штучному інтелекту потрібно з’ясувати, як перемістити персонажа з точки А в точку Б. Ви можете використовувати різні підходи (наприклад, розгорнути просту подію зіткнення та дати персонажам «пам’ять»). Якщо персонажі можуть згадати, де вони були тоді, вони зможуть приймати більш обґрунтовані рішення про те, куди йти далі. Якщо всі можливі наслідки не привели до успіху, ШІ може повернути персонажа назад і вибрати інший маршрут. Таким чином, ІШІ проводитиме систематичний пошук шляху до мети.

Перевагою цього методу є низьке навантаження на обчислювальні ресурси. Це означає, що ви можете підтримувати велику кількість рухомих персо-

нажів, не сповільнюючи гру. Подібним чином можна реалізувати патрулювання в іграх. Потрібно лише вказати початкову, проміжну та кінцеву точки маршруту.

Тактичний ІШІ. Тактичний штучний інтелект не обмежується пересуванням, але також включає групову підтримку та бої в одній команді [9]. Відповідальність за планування та координацію роботи групи несе командир. Групи можна використовувати для реалізації тактики описаної раніше системи, наприклад системи на основі правил або кінцевих автоматів. Типовими прикладами групової поведінки в іграх є зцілення (цілителі тримаються поблизу персонажів, які швидше за все зазнають атаки), розвідка, вогневе прикриття, жертвоприношення (прикриття більш цінних персонажів менш цінними). Крім того, в групі може бути корисним інший рівень аналізу - аналіз можливостей кожного члена групи. Важливо знати, в яких ситуаціях група може бути ефективною, коли група отримує перевагу, а коли групі слід відступити.

Стратегічний ШІ — це штучний інтелект найвищого рівня, він керує цілою армією та виробляє оптимальні стратегії [10]. Стратегічний штучний інтелект зазвичай використовується в стратегіях реального часу. Однак зараз він все частіше реалізується в тактичних шутерах від першої особи та деяких інших жанрах ігор. Командир під керівництвом гравця може бути окремою системою, а може бути налаштований як порожній об'єкт, який не має місця і графічного зображення, але оновлюється і приймає рішення. Командири керуються ієрархічними системами правил і кінцевими автоматами, керуючи такими діями, як збір ресурсів, вивчення дерева технологій, створення армії тощо. Як правило, базова підтримка елементів гри не вимагає особливо складного інтелекту. Інтелект потрібен в основному під час взаємодії з іншими гравцями. Управління цією взаємодією (або боєм) — це те, де ШІ працює в першу чергу. Командир повинен вивчити карту гри, щоб ідентифікувати гравця, визначити ключові області інтересу, такі як вузькі проходи, побудувати захист і

проаналізувати захист інших гравців. Для цього використовуються карти рішень.

1.3 Технології, які були використані для реалізації застосунку

Для виконання цілей треба розробити: ігровий застосунок, систему керування юнітами, систему навчання. Для цього використовувалися наступні технології:

1. Unity — це кросплатформний ігровий рушій, розроблений Unity Technologies, вперше анонсований і випущений у червні 2005 року на Всесвітній конференції розробників Apple як ігровий рушій для Mac OS X. З тих пір рушій поступово розширювався для підтримки різноманітних настільних комп'ютерів, мобільних пристроїв, консолей і платформ віртуальної реальності [11].

Рушій можна використовувати для створення тривимірних (3D) і двовимірних (2D) ігор, а також для інтерактивного моделювання. Рушій був прийнятий галузями за межами відеоігор, такими як кіно, автомобілебудування, архітектура, інженерія, будівництво та Збройними силами США.

Для реалізації проекту був обраний саме цей ігровий рушій по декільком причинам:

- Безкоштовне використання для некомерційних проектів та для комерційних проектів, якщо дохід або фінансування компанії до 100 тисяч доларів за рік, що охоплює досить значний пласт розробників.
- Підтримка багатоплатформної розробки, що є важливим критерієм при розробці ігрових застосунків.
- Використання для написання скриптів популярної мови програмування C#.
- Відносна простота освоєння рушія.

2. TensorFlow — це безкоштовна бібліотека програмного забезпечення з відкритим вихідним кодом для машинного навчання та штучного інтелекту.

Його можна використовувати для виконання ряду завдань, але він зосереджений на навчанні глибоких нейронних мереж [12].

TensorFlow був розроблений командою Google Brain для внутрішнього використання Google у дослідженнях і виробництві. Початкова версія була випущена під ліцензією Apache License 2.0 у 2015 році. Google випустила оновлену версію TensorFlow під назвою TensorFlow 2.0 у вересні 2019 року.

TensorFlow можна використовувати в широкому спектрі мов програмування, особливо в Python, а також у Javascript, C++ і Java. Ця гнучкість підходить для низки додатків у багатьох різних секторах.

3. Keras — це бібліотека програмного забезпечення з відкритим вихідним кодом, яка надає інтерфейс Python для штучних нейронних мереж. Keras виступає в якості інтерфейсу для бібліотеки TensorFlow [13].

До версії 2.3 Keras підтримував декілька серверних програм, включаючи TensorFlow, Microsoft Cognitive Toolkit, Theano та PlaidML. Починаючи з версії 2.4, підтримується лише TensorFlow. Створений для швидкого експериментування з глибокими нейронними мережами, він зосереджений на тому, щоб бути зручним, модульним і розширюваним. Він був розроблений в рамках дослідницького проекту ONEIROS (Open-ended Neuro-Electronic Intelligent Robot Operating System), а його основним автором і супроводжувачем є Франсуа Шолле, інженер Google.

1.4 Огляд існуючих рішень реалізації ІШІ

Для розуміння сучасних тенденцій при розробці ІШІ, аналізу переваг та недоліків для аналізу було обрані ті існуючі рішення котрі демонструють використання стандартних підходів реалізації або ж розширюють їх оригінальними рішеннями.

1.4.1 Horizon Zero Dawn

Horizon Zero Dawn знаходиться в списку найкращих ексклюзивів для Playstation 4. У ролі мисливиці Елой гравці подорожують постапокаліптичними ландшафтами майбутнього, щоб розкрити таємниці її минулого і дізнатися про причини руйнування світу. Занепад людства призвів до розквіту «машин» — роботів різних форм і розмірів, що вільно живуть усюди. Ці тварини-роботи розумні, скоординовані та смертельно небезпечні: щоб вижити, потрібно швидко думати, щоб знищити їх — ретельно готуватися та планувати [14].

Система ШІ, що використовується в Horizon Zero Dawn називається «планувальник мережі ієрархічних завдань» (HTN). На відміну від систем планування, які використовуються в іграх на подібні F.E.A.R., які планують одну дію за іншою, планувальник HTN генерує плани, складені з макросових дій. Кожен макрос містить кілька дій у заданій послідовності. Це ідеально підходить для розробки ігор, адже дизайнери можуть створювати макроси хорошої поведінки, які очікують побачити в грі [15].

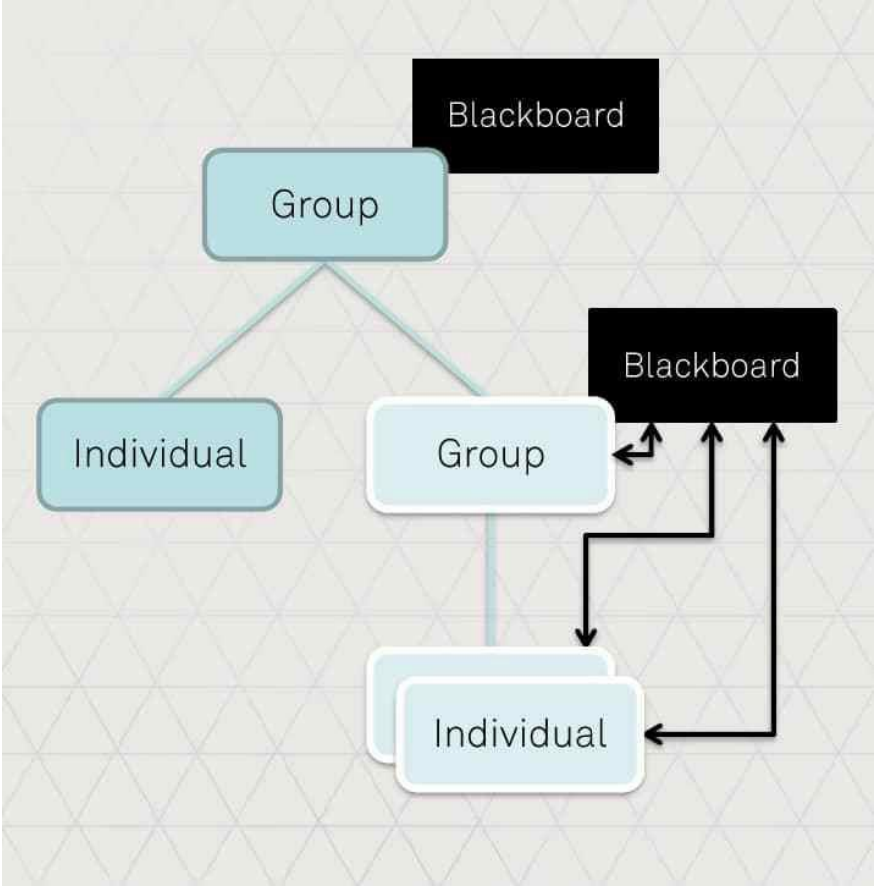

Рис. *1 Приклад групової ієрархії*

Планувальник HTN використовується машинами з двома цілями: окремі агенти можуть запитувати план дій для вирішення конкретних завдань, наприклад, пошуку нових областей, нападу на людину чи втечі.

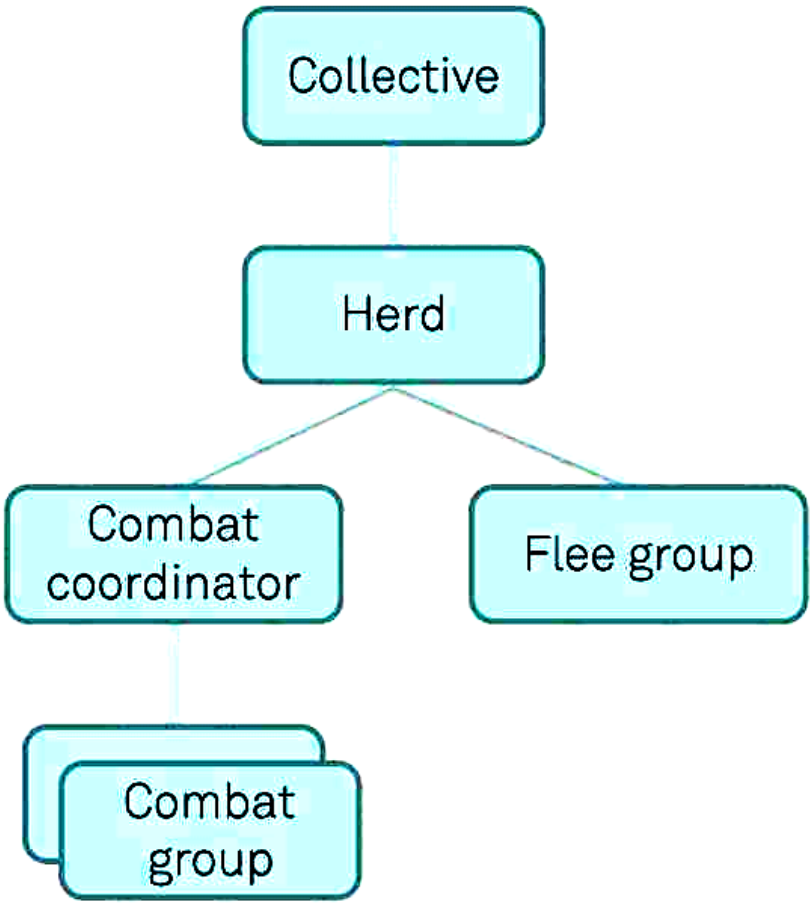

Рис. *2 Приклад групової поведінки для втечі*

Групові агенти можуть виконувати плани, надавати цілі, які повинні призначатися окремим агентам, що дозволяють групам активно ділитися інформацією з іншим усередині ієрархії, і потенційно формувати нові підгрупи чи

розбивати існуюче. Якщо коротко, то кожна окрема машина все одно несе велику відповідальність за власну поведінку та виконання призначених завдань, а також використовує планувальник для прийняття рішень. Групи зберігають координацію окремих машин, перегруповують їх на базі, що відбувається в кожному світі подій і, що саме важливе, назначають ролі машин, щоб вони знали свою задачу у функції підтримки стада.

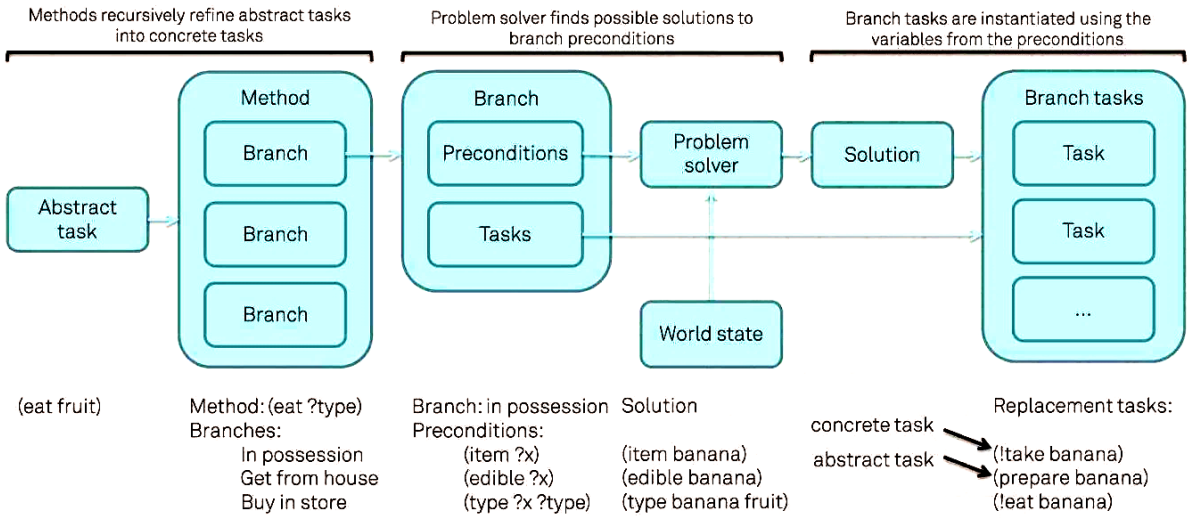

Рис. *3 Схема планувальника HTN*

Важливим в рамках даної роботи є планувальник HTN, який дозволяє більш глибоко планувати дії агентів цілими комплексними блоками. Також гарним рішенням при розробці ІШІ було додання групових агентів який дозволяє зробити поведінку групи більш реалістичною.

1.4.2 Alien: Isolation

Alien: Isolation — це гра хорор на виживання випущена 2014 року, розроблена Creative Assembly та видана Sega для Windows, PlayStation 3, PlayStation 4, Xbox 360 та Xbox One [16].

Alien: Isolation — це пригодницька гра для одного гравця з акцентом на стелс і жах на виживання. Гравець керує Амандою Ріплі від першої особи і

повинен досліджувати космічну станцію та виконувати цілі, уникаючи, перехитриючи та перемагаючи ворогів.

Щоб ксеноморф поводився як слід, Creative Assembly використовували перевірені техніки створення ШІ. Ксеноморф мав виконувати свої функції та діяти саме так, як було задумано розробниками.

Для цього грі були потрібні дві різні системи управління поведінкою: «макро» або «режисер» та «мікро», що відповідає безпосередньо за ксеноморфа. Перша спостерігає за діями гравця і знає, у якій точці рівня той перебуває. Друга, своєю чергою, орієнтується на власні «відчуття», щоб відстежувати користувача.

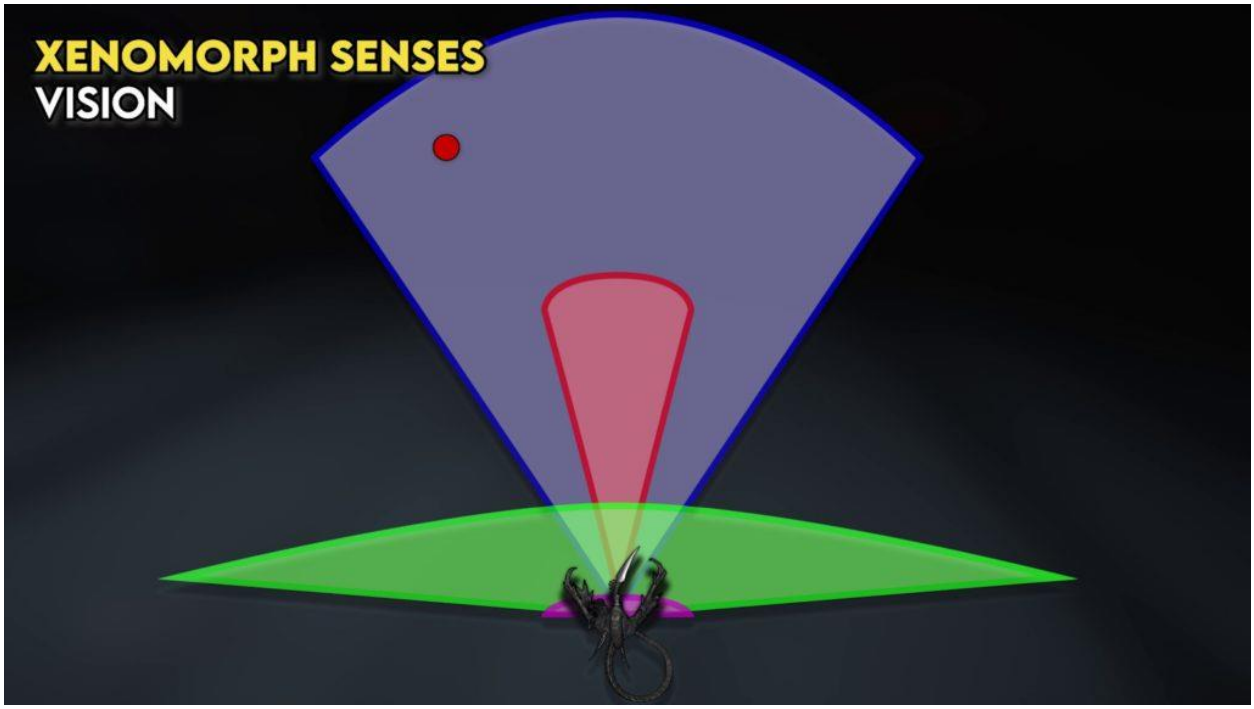

Рис. *4 Схема «зору» ксеноморфа*

"Макро"-система періодично вказує ксеноморфу місцезнаходження гравця, спрямовує його. Незважаючи на це, Чужий має сам знайти користувача, адже режисер не дає йому точних координат. Так гравець може обдурити монстра та втекти

Режисер відповідає за управління так званим «показником загрози», що дозволяє системі визначати, наскільки сильний тиск зазнає гравець. Щось

схоже було використано у Left 4 Dead, де кількість противників залежала від того, наскільки успішно гравці просуваються за рівнем. У випадку з Alien: Isolation йдеться не про атаку на користувача, а про збільшення її ймовірності.

Система періодично підвищує рівень загрози та повідомляє ксеноморфу, куди тому рухатися. Коли ксеноморф недалеко від гравця, на рівень загрози впливають такі фактори:

- відстань від користувача до монстра;
- чи користувач у полі зору ксеноморфа;
- чи знаходиться ксеноморф близько до гравця на екрані пристрою відстеження руху і чи може він швидко дістатися до користувача.

Останній пункт особливо важливий, адже навіть якщо пристрій показує, що ксеноморф поблизу, насправді він може бути в іншій кімнаті. У такому разі рівень загрози не піднімається.

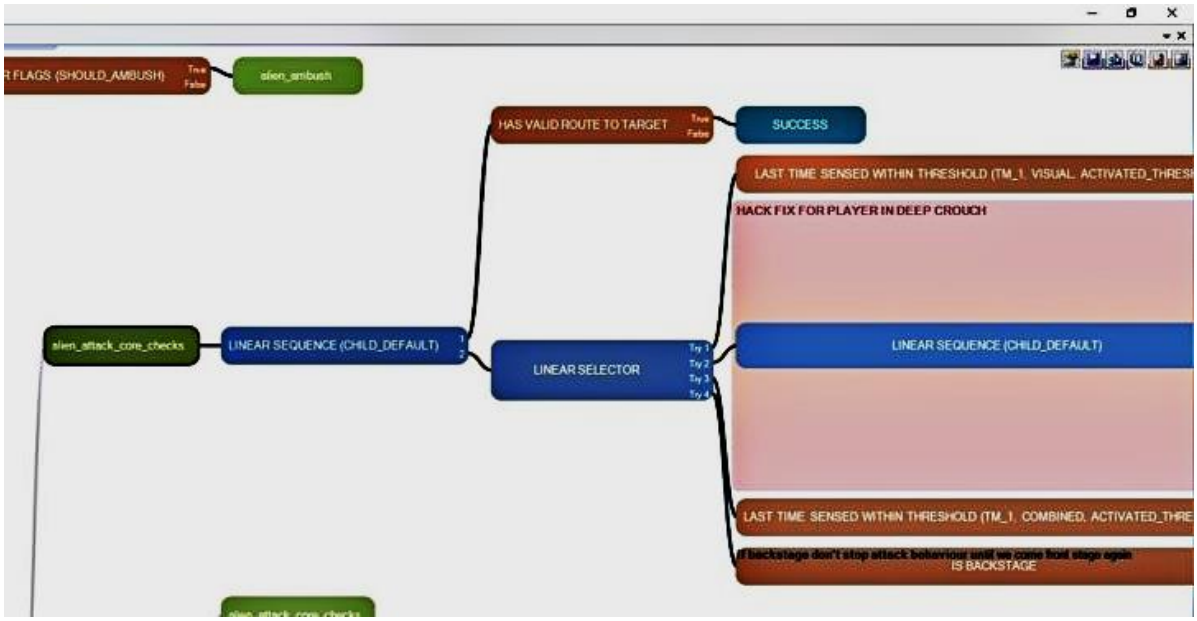

Рис. 5 *Частина дерева поведінки ксеноморфа*

Зазвичай, коли він досягає максимального значення, режисер «відкликає» ксеноморфа в інше місце. Ідея в тому, щоб час від часу нагадувати гравцеві про небезпеку, але потім давати час «віддихатись».

Усім цим процесом керує Utility AI. Раніше ця система використовувалася в BioShock: Infinite для Елізабет. В Alien: Isolation вона диктує, куди рухатися ксеноморфу і ставить пріоритети для його дій. Останні відповідають за те, чи повинен ксеноморф закінчити те, що робить зараз, або відразу кинутися в атаку.

Ксеноморф «працює» у двох станах: активному та пасивному. У першому він досліджує локації щодо джерела шуму чи інших подій, які привернули його увагу.

Ксеноморф переходить у пасивний стан, як тільки рівень загрози досягає піку. Тоді монстр залазить у вентиляцію.

Поведінка ксеноморфа диктується більш ніж сотнею вузлів «дерева поведінки». На його вершині знаходяться близько 30 вузлів, які відповідають за те, якого типу поведінки дотримуватиметься монстр. Кожен із головних вузлів має безліч підсекцій, які відповідають за конкретні дії.

На початку гри деякі частини дерева закриті. Система поступово відкриває їх - це дозволяє ксеноморфу "вчитися" новим трюкам і завжди тримати гравця у напрузі. Нові можливості відкриваються після певних дій гравця. При цьому жодна з них не веде до смерті останнього, щоб не давати перевагу ксеноморфу.

Проте, за словами Енді Брея, в окремих моментах кампанії система відкриває деякі моделі поведінки Чужого, якщо цього не сталося раніше. Таким чином, інтелект ксеноморфа як би «наздоганяє» прогрес гравця.

Отримавши завдання від режисера, Чужий кидається виконувати його, спираючись на систему пошуку шляху та власні «почуття». Монстр може чути кроки гравця, постріли і навіть звуки, що видаються пристроєм відстеження руху, якщо він знаходиться на відстані близько півтора метра від нього.

Крім того, ксеноморф має свого роду «очі на потилиці», які дозволяють йому бачити гравця, що знаходиться на невеликій відстані позаду.

Нарешті, Чужий має особливий патерн руху. Він не просто рухається до конкретної локації, але обшукує кімнати в пошуках здобичі. Його приваблюють певні об'єкти оточення, як заздалегідь задані дизайнерами, і обрані грою випадково.

Чужий гарантовано досліджує всі такі місця в кімнаті, але не обов'язково ходитиме від одного до іншого за оптимальним і найкоротшим маршрутом. Пріоритет надається тим об'єктам, що знаходяться в зоні видимості монстра

Важливим в рамках даної роботи є поєднання «макро» та «мікро» систем керування агентами. Це дозволяє розділити рівні відповідальності агентів, корегувати їх дії та створювати необхідний ігровий досвід в залежності від дій.

1.4.3 Seaman

Seaman це відеогра про віртуальних домашніх тварин для Sega Dreamcast. Це одна з небагатьох ігор Dreamcast, яка використовує переваги кріплення мікрофона. Гра стала культовою завдяки чорному гумору, химерній естетиці та інноваційному геймплею [17].

Seaman вважається унікальною відеогрою, оскільки містить дію, яка зустрічається в малій кількості ігор. Мета гравця — годувати та доглядати за seaman-ом, забезпечуючи йому необхідну компанію. Процес гри відбувається в реальному часі, тому гравець повинен перевіряти seaman-а щодня в режимі реального часу, інакше він може померти. Частина знань seaman-а — це випадкові дрібниці. Коли він запитує, який день народження гравця (а гравець відповідає через вхід мікрофона), seaman поділиться важливими подіями, які відбулися в цей день.

Протягом гри seaman проходить декілька форм життя, від грибниці до жабоподібної істоти.

Хоча seaman стає досить прирученим, він не перестає ображати гравця або робити не дуже дружні зауваження.

Seaman — це гра, для якої розробники поставили завдання щоб вона використовувала мовний фільтр, щоб подолати прірву між геймером і не-гравцем. Задача перетворити справжню мову на мовний елемент/фільтр для ігор неймовірна складна, але насправді це схоже на перетворення природних мовних виразів на гральні карти та гру в карти з іншою особою.

Це означає , що Seaman насправді як гра в покер. Ключові слова — це «карти», які відтворюються між кінцевим користувачем і ігровим персонажем (seaman-ом). Більшість геймерів цього не зрозуміють, але Seaman — це гра з великою кількістю ігрових елементів, які можна використовувати так само, як і в покері. Цей рух вперед-назад є основною механікою гри. Так само, як у містобудівних симуляторах , коли ви додаєте нову будівлю в місто, надання seaman-у іншого ключового слова веде до абсолютно нового шляху спілкування, і, можливо, це найпростіший спосіб передати концепцію гри [18].

Важливим в рамках даної роботи є те, що для свого часу(1999) концепція була новою та унікальною, що викликало зацікавленість гравців. Хоч розмова з seaman-ом і не може бути безкінечною це було проривним використанням ігрового штучного інтелекту, що на той час давало унікальний геймплей: через перетворення голосу в набір слів та виділення окремих слів, що були тригерами, вести розмову яка здавалася осмисленою.

1.4.4 The Elder Scrolls IV: Oblivion (Radiant A.I.)

The Elder Scrolls IV: Oblivion — це рольова відеогра у відкритому світі, розроблена Bethesda Game Studios і видана Bethesda Softworks і 2K Games. Це четверта частина серії Elder Scrolls, після The Elder Scrolls III: Morrowind 2002 року, і була випущена для Microsoft Windows і Xbox 360 у 2006 році, а потім для PlayStation 3 у 2007 році. Дія гри відбувається у вигаданій провінції Сироділ. Основна історія зосереджена на зусиллях персонажа гравця перешкодити фанатичному культу, відомому як Міфічний Світанок, який планує відкрити ворота порталу в демонічний світ, відомий як Забуття [19].

Гра продовжує традицію відкритого світу своїх попередників, дозволяючи гравцеві подорожувати будь-де в ігровому світі в будь-який час і ігнорувати або відкладати основну сюжетну лінію на невизначений термін. Постійною метою гравців є вдосконалення навичок свого персонажа, які є числовими представленнями певних здібностей. На початку гри гравець вибирає сім умінь як основні для свого персонажа, а ті, що залишаються, називаються другорядними.

Radiant A.I. — це система динамічних реакцій персонажів і ігрового світу на дії гравця. Використання Radiant A.I. створює більш особистий і непередбачуваний досвід для гравців, адаптуючи світ відповідно до минулих дій і досвіду гравця. Наприклад, Radiant A.I. може використовуватися для встановлення мети квесту в місці, яке гравець раніше не відкрив, і заповнення цього місця ворогами, які відповідають здібностям гравця, або деталізують реакцію та поведінку персонажів на основі минулих дій.

Radiant A. I. також використовувався в інших рольових іграх від Bethesda, таких як Fallout 3 і Fallout: New Vegas, в яких вони розширили його, і дали розробникам ідею повністю розширити його в Skyrim, але до певної міри; ніколи не буває випадків, коли гравець не може щось зробити через расу, клас (або сильні навички, у випадку з Elder Scrolls: Skyrim) або знак народження свого персонажа.[20]

Хоч така система давала унікальний досвід для гравця, бо NPC реагували досить реалістично, розробникам довелось спеціально обмежувати дії персонажів.

Через повну свободу персонажів досить часто виникали ситуації повного хаосу в ігровому світі: від одної невеликої дії, як сніжний ком наростали дії NPC, що були поруч і часто такі реакції були неконтрольованими.

Рис. *6 Взаємодія NPC між собою в* The Elder Scrolls IV: Oblivion

В рольових іграх часто однією, з головних частин гри є сюжет. Коли ж персонажі діють вільно то певна низка подій може призвести до того, що персонаж, який необхідний для сюжету в певний момент може до цього моменту не дістатися.

Тому розробники Oblivion обмежили дії які можуть виконувати певні персонажі в певній ситуації та додали певні хитрощі для правильного функціонування гри такі, як безсмертя сюжетно важливих персонажів.

Першопочатковий же варіант Radiant A.I. зіштовхнувся же з основною проблемою використання традиційного AI в іграх, а саме певна некотрольованість та непередбачуваність дій та результатів.

Важливим в рамках даної роботи є те, що підхід до розробки ІШІ в цьому рішенні показує переваги та недоліки надання свободи дій неігровим персонажа. Також процес розробки даного рішення продемонстрував як в деяких аспектах відрізняється розробка традиційного штучного інтелекту від розробки ІШІ.

1.4.5 Facade

Facade — це інтерактивна драма в реальному часі, де гравці безпосередньо спілкуються з неігровими персонажами в короткому драматичному сценарії. Незважаючи на те, що грі понад 15 років, вона є одним із рідкісних випадків наукового дослідницького проекту, який досяг глобального статусу та популярності [21].

У Facade гравців запрошують до квартири Грейс і Тріпа: подружньої пари, яка познайомилася завдяки персонажу гравця ще в коледжі. Однак у їхніх стосунках справи йдуть не дуже добре, і ваш прихід того вечора в гості є вирішальним моментом для їхнього шлюбу. Вас мимоволі втягують у пасивно-агресивні психологічні пари коли вони один з одним починають сперечатися по різним темам.

Гравці взаємодіють з парою, блукаючи по їхній квартирі та взаємодіючи з предметами та меблями в будинку, або спілкуючись з ними безпосередньо. Facade вирізняється з-поміж багатьох інших ігор тим що ви розмовляєте з Грейс і Тріп, набираючи текст на клавіатурі, а персонажі інтерпретують ваші введення та відповідають відповідним чином. Вони реагують захопленням, здивуванням, сміхом, відразою чи навіть гнівом, коли ви оскаржуєте їхню точку зору або рішення, запитуєте про проблеми, які вони не хочуть обговорювати, або просто відверто ображаєте їх [22].

Рис. *7 Взаємодія за допомогою введення тексту*

Зрештою, залежно від ваших дій, гра призведе до однієї з кількох попередньо визначених кінцівок: ви можете або допомогти вирішити їх проблеми, або зайняти сторону одного з них і тоді інший покине будівлю, або якщо ви розгнівяєте обох персонажів вас можуть вигнати.

Розробники розуміли, що поки немає змоги зробити штучний інтелект який міг би обдумано реагувати на будь які запити, тому вони поставили перед собою ціль зробити гру в якій штучний інтелект буде грати переконливий спектакль, однією з дійових осіб якого буде сам гравець.

Щоб контролювати темп і керувати кожною соціальною грою в межах запланованої драми історії, Facade координує це за допомогою так званого менеджера драми. Він з'ясовує, куди розвивається сюжет на основі того, що трапилося та які емоції «відчуває» (точніше, записав) кожен персонаж у той час. Серед них мова Action Behavior Language ABL [23], що вимовляється як «Able(здатний)», яка кодує не лише реакції та діалоги, вибрані кожним персонажем у конкретних ситуаціях, а й те, як вони розташовуються в кімнаті, взаємодіють з об'єктами світу та поводяться в спосіб, який відповідає тому, що Менеджер драми хоче відтворити. І нарешті, ваш внесок як гравця. Facade запускає інтерпретатор у режимі реального часу [24], який намагається визначити основне почуття того, що вводить гравець, а потім використовує це, щоб впливати на те, як менеджер драми координує ритми історії — потенційно додаючи нові сюжетні моменти, переписуючи сюжет, коли він йде — що в кінцевому підсумку впливає на те, як актори реагують на вас.

```
sequential behavior AnswerTheDoor() {
  WME w;
  with success_test { w = (KnockWME) } wait;
  act sigh();
  subgoal OpenDoor();
  subgoal GreetGuest();
  mental_act { deleteWME(w); }
}
```

Рис. *8 Скрипт дій написаний за допомогою* ABL

Менеджер драми — також відомий як такт-секвенсер — вирішує, які сюжетні дії відтворюються в реальному часі під час виконання. Він розглядає поточний стан історії та персонажів і вирішує, які невикористані дії допоможуть перенести історію через заплановану дугу напруги. Лише один ритм є активним у будь-який момент часу, і на основі того, що відбувається в грі — або, радше, що ви робите під час гри — він може прийняти рішення змінити поточний ритм. У грі є 27 основних сюжетних дій. Деякі використовуються в першій частині гри для створення напруги, кілька для однієї із доступних кінцівок у фінальній частині, а решта вибирається для основних соціальних ігор у середині історії [25].

Representation of Discourse Acts	***Pragmatic Meaning of Discourse Acts***
`(DAAgree ?char)`	Agree with a character. (e.g. "certainly", no doubt", "I would love to")
`(DADisagree ?char)`	Disagree with a character. (e.g. "No way", "Fat chance", "Get real", "Not by a long shot")
`(DAPositiveExcl ?char)`	A positive exclamation, potentially directed at a character. ("Yeah", "Wow", "Breath of fresh air")
`(DANegExcl ?char)`	A negative exclamation, potentially directed at a character. (e.g. "Damn", "That really sucks", "How awful", "I can't stomach that", "D'oh!")
`(DAExpress ?char ?type)`	Express an emotion, potentially directed at a character. The emotion types are *happy* ("I'm thrilled", ":-)"), *sad* ("That bums me out", ":-("), *laughter* ("ha ha"), and angry ("It really pisses me off", "grrrr").
`(DAMaybeUnsure ?char)`	Unsure or indecisive, potentially directed at a character. This discourse act is usually a response to a question. (e.g. "I don't know", "maybe", "I guess so", "You've lost me")
`(DAThank ?char)`	Thank a character (e.g. "Thanks a lot")
`(DAGreet ?char)`	Greet a character. (e.g. "Hello", "What's up")
`(DAAlly ?char)`	Ally with a character. (e.g. "I like you", "You are my friend", "I'm here for you")
`(DAOppose ?char)`	Oppose a character. (e.g. "Kiss off", "You're the worst, "I hate you", "Get out of my life")
`(DADontUnderstand ?char)`	Don't understand a character utterance, or the current situation (e.g. "I'm confused", "I don't get it", "What are you talking about")
`(DAApologize ?char)`	Apologize to a character. (e.g. "I'm sorry", "My bad", "How can I make this up to you")
`(DAPraise ?char)`	Praise a character. (e.g. "You're a genius", "What a sweetheart you are", "You've got good ideas")
`(DACriticize ?char ?level)`	Criticize a character. There are two levels of criticism, *light* ("You're weird", "Don't be so up tight"), and *harsh* ("Idiot", "What a dipshit")
`(DAFlirt ?char)`	Flirt with a character. (e.g. "You look gorgeous", "Kiss me", "Let's get together alone sometime")
`(DAPacify ?char)`	Pacify a character. (e.g. "Calm down", "Relax, guys" "Keep your shirt on", "Take it easy")

Рис. *9 Список доступних дій*

У грі ніколи не використовуються всі сюжетні дії протягом одного проходження. Отже, хоча є 27 основних дій, кожен запуск, подібно до того як відбувається в теорії драматургії, створений як найменша оповідна послідовність, яку може виконувати гра.

Однак в даному підході була одна фундаментальна проблема: його можна було дуже легко зламати.

Якщо гравець вирішував не слідувати спектаклю то персонажі починали безкінечно перепитувати гравця, що він сказав, давати незрозумілі або не зв’язні відповіді або виконувати нелогічні дії. Саме це і є однією з основних проблем використання підходів традиційного штучного інтелекту в ігрових застосунках: невизначеність та непередбачуваність. При створені гри розробники хочуть передати гравцю певний досвід, а якщо в грі буде якийсь елемент який поводить себе повністю непередбачувано, то не факт, що цього вдасться досягти.

Рис. *10 Реакція персонажа на непередбачувану дію*

Важливим в рамках даної роботи є те, що підхід до реалізації ІШІ в цьому рішенні показує проблеми які можуть виникнути в разі надання свободи

дій агентам, а саме неоднозначність результату та неможливість передбачення чи буде цей результат взагалі досягнутий.

1.5 Висновки з розділу 1

В даному розділі було проведено огляд проблеми використання штучного інтелекту в ігрових застосунках в цілому та його розробка з використанням машинного навчання; аналіз сучасних підходів до реалізації ІШІ; аналіз існуючих рішень. На основі цього, можна зробити наступні висновки:

1. Основна причина відмінностей ігрового штучного від традиційного полягає в цілі, яку переслідує кожен з них: імітація розумної поведінки в прийняті рішень в певній задачі для традиційного та створення певного антуражу ігрового світу та допомога в передачі певного ігрового досвіду в випадку ІШІ.
2. Вибір підходу до розробки ІШІ накладає певні обмеження щодо того, які дані потрібно бути отримувати про ігрове середовище, NPC та т. п. тому потрібно розробляти систему гнучкою, щоб потім при імплементації системи керування, яка буде навчатися було менше проблем та така імплементація взагалі була можлива
3. Розглянуті основні сучасні підходи до розробки ІШІ та їх переваги та недоліки.
4. Розглянуті існуючі рішення реалізація ІШІ та проаналізовані основні труднощі реалізації та обмеження які в них присутні.
5. Зроблений огляд існуючих рішень показує що проблема вирішена не повністю та комбінація різних підходів дозволить підвищити ефективність їх використання при розробці ІШІ.

РОЗДІЛ 2 ДОСЛІДЖЕННЯ ЗАСОБІВ РЕАЛІЗАЦІЇ ІГРОВОГО ШТУЧНОГО ІНТЕЛЕКТУ

2.1 Засоби реалізації

Дана робота була виконана за допомогою ігрового рушія Unity, скрипти були написані у середовищі розробки Microsoft Visual Studio 2019 високо функціональною мовою C#. C# - є об'єктно-орієнтованою мовою програмування і аналогічно іншим сучасним мовам групує пов'язані поля, методи, властивості і події в структури даних, які називаються класами.

2.2 Модулі і алгоритми

Під час розробки ігрового застосунку використовувалися два основні типи алгоритмів: алгоритми для керування ігровим середовищем та алгоритми для навчання системи керування NPC.

Перший тип алгоритмів використовується в традиційних реалізаціях ІШІ та направлений на ефективне отримання інформації та управління NPC.

2.2.1 Алгоритми для керування ігровим середовищем

В класі «**AStar**» реалізовано метод находження найкоротшого шляху алгоритмом A*. Відмінність від стандартного A* полягає в методі находження сусідніх прохідних клітинок.

Лістинг *1 Метод знаходження прохідних сусідів для клітинки*

```
private static List<Tile>
GetWalkableTiles(List<List<TileData>> map, Tile
currentTile, Tile targetTile)
{
```

```
    List<Tile> possibleTiles = AllPosibleTiles(map,
currentTile);

    possibleTiles.ForEach(tile =>
tile.SetDistance(targetTile.X, targetTile.Y));

    var maxX = map.First().Count - 1;
    var maxY = map.Count - 1;

    possibleTiles.ForEach(tile =>
    {
        if (IsSub(currentTile, tile))
        {
            if (tile.X >= 0 && tile.X <= maxX && tile.Y >=
0 && tile.Y <= maxY)
            {
                var subTile = new SubTile(new
Vector2Int((currentTile.X - 1 + tile.X) / 2, (currentTile.Y
- 1 + tile.Y) / 2));
                tile.isWalkable =
subTile.walkableData.IsWalkable;
            }
            else
            {
                tile.isWalkable = false;
            }
        }
        else
        {
            if (tile.X >= 0 && tile.X <= maxX && tile.Y >=
0 && tile.Y <= maxY)
            {
```

```
                tile.isWalkable =
map[tile.Y][tile.X].WalkableData.IsWalkable;
            }
            else
            {
                tile.isWalkable = false;
            }
        }
    });
    return possibleTiles
            .Where(tile => tile.X >= 0 && tile.X <= maxX)
            .Where(tile => tile.Y >= 0 && tile.Y <= maxY)
            .Where(tile => tile.isWalkable)
            .ToList();
}
```

Особливістю даної реалізації є те, що для кожної клітинки доступними сусідами є не тільки розташовані по горизонталі та вертикалі сусіди, але й по діагоналі. Але у випадку переміщень по діагоналі використовуються сабтайли.

Кожна клітинка може бути прохідною або не прохідною та має свою вартість проходження, яка визначається типом та особливостями місцевості та будівлями, які там знаходяться. Сабтайли вираховують прохідність та вартість проходу базуючись на правилі з’єднання, праве або ліве. Від цього правила залежить які з тайлів використовуються для розрахунків.

Лістинг *2 Метод знаходження даних про прохідність для сабтайлу*

```
public TileWalkableData
TileWalkableDataForSubtile(Vector2Int Position)
    {
        TileWalkableData walk1, walk2;
```

```
        if(RightDirectionsSubTiles.Contains(Position))
        {
            walk1 =
WalkableDataForTile(TileForPosition(Position).FormingData);
            walk2 =
WalkableDataForTile(TileForPosition(Position.Down().Right()
).FormingData);
        } else
        {
            walk1 =
WalkableDataForTile(TileForPosition(Position).FormingData);
            walk2 =
WalkableDataForTile(TileForPosition(Position.Down().Right()
).FormingData);
        }

        TileWalkableData result = new TileWalkableData();
        result.IsWalkable = walk1.IsWalkable &&
walk2.IsWalkable;
        result.cost = (walk1.cost + walk2.cost) * 0.2;
        return result;
    }
```

Схематичні приклади розрахунку прохідності для лівого та правого правил з’єднання зображенні на рисунках 11 та 12.

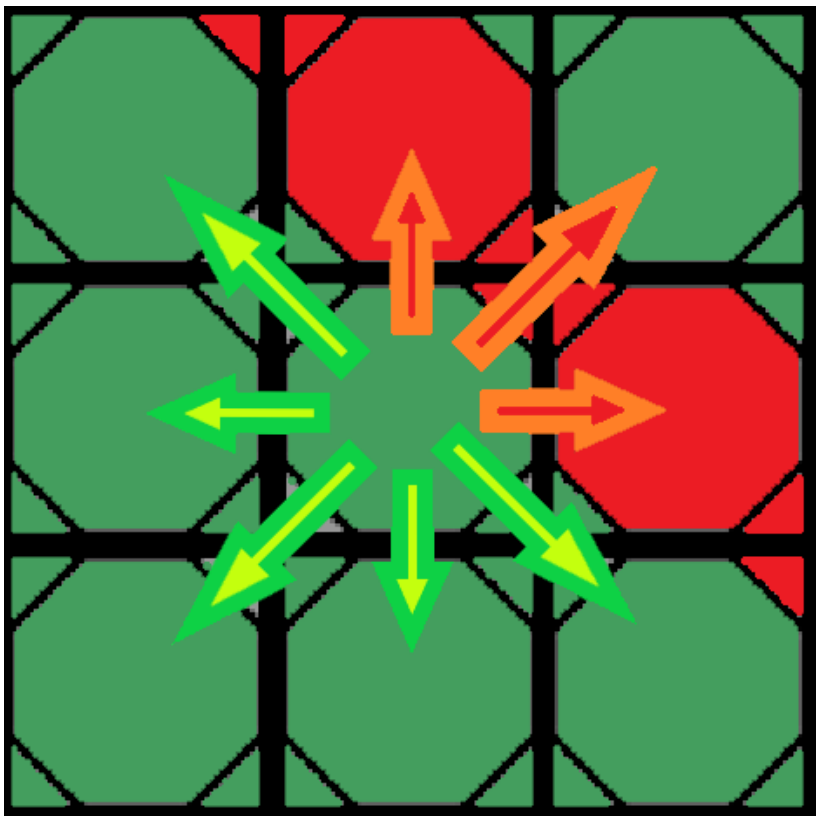

Рис. *11 Прохідність шляху при лівому правилу з'єднання сабтайлів*

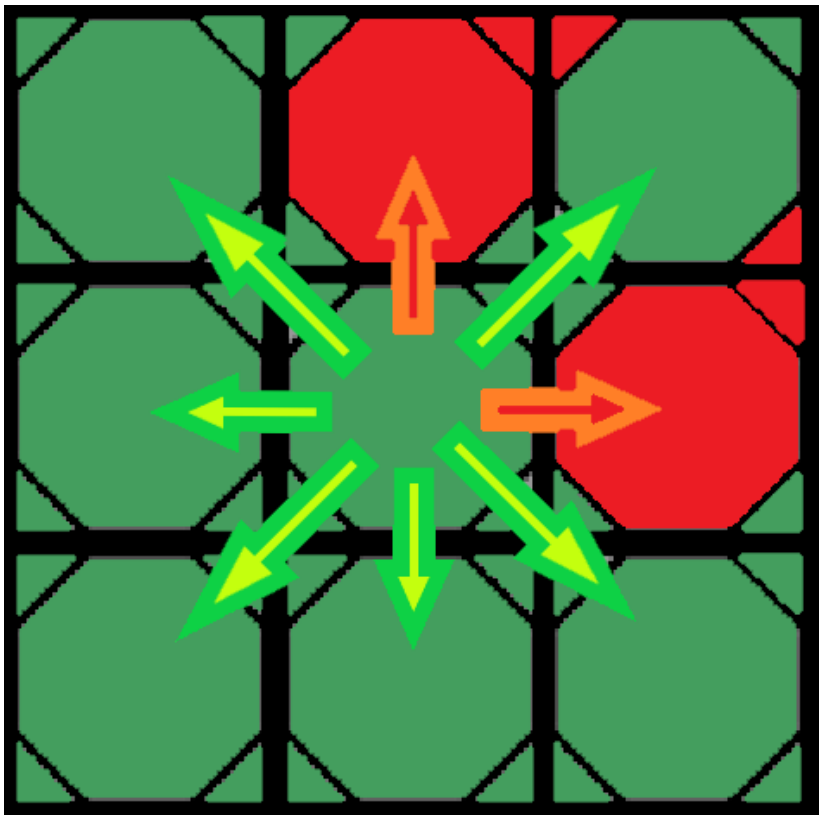

Рис. *12 Прохідність шляху при правому правилу з'єднання сабтайлів*

Також було додано реалізацію пошуку можливо шляху з урахуванням доступних технологій. Тобто шлях прокладався не тільки через вже прохідні клітинки, а й через такі, що можуть стати прохідними використовуючи поточні технології гравця.

Даний метод використовувався для визначення найкращого місця для зведення будівлі в певному радіусі від доступних клітинок для будівництва, що в свою чергу підвищувало ефективність даної системи.

Лістинг *3 Метод знаходження даних про прохідність враховуючи можливу прохідність*

```
return possibleTiles
                .Where(tile => tile.X >= 0 && tile.X <= maxX)
                .Where(tile => tile.Y >= 0 && tile.Y <= maxY)
                .Where(tile => tile.isWalkable || planing ?
tile.isPotentialWalkable : false)
                .ToList();
…
```

Лістинг *4 Метод визначення можливої прохідності для тайлу*

```
static public bool IsPotentialWalkable(TileFormationData
tile)
    {
        if(!tile.IsLand)
        {
            return
CurrentPlayerData.CurrentPlayer.ResearchManager.ExploredRes
earches.Contains(Research.WoodCollecting) &&

CurrentPlayerData.CurrentPlayer.ResearchManager.ExploredRes
earches.Contains(Research.SimpleTool);
```

```
        }
        return true;
}
```

Також був розроблений метод отримання найближчого тайлу з певним кодом, який визначає його структуру.

Цей метод використовується досить часто тому було проведена його оптимізація.

Від тайлу для якого проводиться пошук найближчого специфічного тайлу будується 9 секторів.

Лістинг *5 Перерахування Sector*

```
public enum Sector
{
  UpLeft,
  Up,
  UpRight,
  Right,
  DownRight,
  Down,
  DownLeft,
  Left,
  Center
}
```

З кожного сектору визначається найближчий тайл, але використовуючи більш дешеву в плані затрат ресурсів та часу функцію, а саме квадрат довжини вектора між двома координатами тайлів.

Після визначення найближчого тайлу в секторі по евристичній функції використовувався AStar для визначення можливості дістатися до цього тайлу та отримання фінальної вартості шляху.

Лістинг 6 *Метод знаходження найближчого специфічного тайлу*

```
public Vector2Int? nearestTile(string code)
    {
        var tiles = MapTiles.FindAll(x => {
            return x.FormationData.TileCode() == code &&
DataManager.sharedInstance.TakedTile.FindAll(y => { return
y.position == x.tilePosition; }).Count <= 0;
        });

        var updatedList = new List<TileModel>();

        var grouped = tiles.GroupBy(x =>
TownhallTile.tilePosition.GetSector(x.tilePosition)).ToList
();
        grouped.ForEach( x => {
            var list = x.ToList();
            list.Sort((q1, q2)=>{ return
TownhallTile.tilePosition.Euhristic(q1.tilePosition).Compar
eTo(TownhallTile.tilePosition.Euhristic(q2.tilePosition));
});

            if(list.Count > 0) updatedList.Add(list[0]);
        });

        Debug.Log(code);
        return
DataManager.sharedInstance.GetNearestTile(TownhallTile.tile
Position, updatedList)?.tilePosition ?? null;
         }
```

2.2.2 Алгоритми навчання

Серед алгоритмів машинного навчання особливе місце займають алгоритми машинного навчання де алгоритм вчиться вирішувати поставлене завдання самостійно без участі людини, безпосередньо взаємодіючи з середовищем в якій він навчається [26].

Такі алгоритми отримали загальну назву — алгоритми навчання без вчителя, для таких алгоритмів не потрібно збирати бази даних, не потрібно проводити їх класифікацію або розмітку.

Алгоритму навчається без вчителя досить тільки давати зворотний відгук на його дії або рішення — хороші вони були чи ні.

Важливо, щоб алгоритм вирішив задачу в цілому, а як конкретно він буде робити в процесі вирішення цього завдання і яким шляхом він піде, щоб її вирішити не так важливо тому можна це відати це вирішувати самому алгоритму очікуючи тільки кінцевий результат.

Тому кінцевому результату дамо алгоритму зрозуміти, добре він вирішив завдання чи ні.

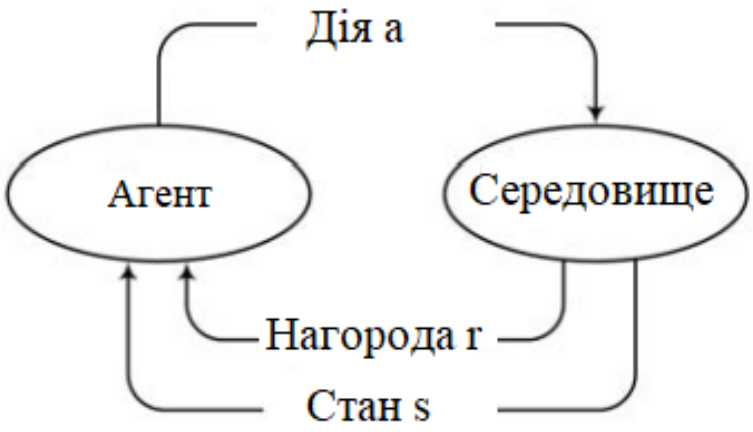

Рис. *13 Схема взаємодія агенту та середовища*

Агент — алгоритм який вміє аналізувати стан середовища і здійснювати в ній якісь дії.

Середовище — віртуальний світ в якому існує наш Агент і своїми діями може змінювати його стан.

Нагорода — зворотній зв'язок від середовища до агенту як відповідь на його дії.

При цьому:

- st — стан середовища (state) на кроці t;
- at — дія агенту (action) на кроці t;
- rt — нагорода (reward) на кроці t.

В кожен момент часу t наш агент спостерігає стан середовища — st, виконує дію — at, за що отримує від середовища нагороду — rt, після чого середовище переходить в стан st+1, яке спостерігає наш агент, виконуючи дію — at+1, за що отримує від середовища нагороду — rt+1 і таких станів t у нас може бути нескінченна множина — n.

Генетичний алгоритм. Це еволюційний алгоритм, тобто основна ідея алгоритму - схрещування (комбінування) [27]. Шляхом перебору та найголовніше відбору виходить правильна «комбінація».

Алгоритм ділиться на три етапи: Схрещування, селекція (відбір), формування нового покоління

Якщо результат нас не влаштовує, ці кроки повторюються до тих пір, поки результат нас не почне задовольняти або станеться одна з нижченаведених умов:

- Кількість поколінь (циклів) досягне заздалегідь вибраного максимуму
- Вичерпано час на мутацію

Створення нової популяції. На цьому етапі створюється початкова популяція, яка, цілком можливо, виявиться не підходящою, проте велика ймовірність, що алгоритм цю проблему виправить. Головне, щоб вони відповідали «формату» та були «пристосовані до розмноження».

Розмноження. Для отримання нащадку потрібні два батьки. Головне, щоб нащадок (дитина) міг успадкувати у батьків їхні риси. При цьому розмножуються всі, а не тільки ті, що вижили, інакше виділиться домінуючий представник, гени якого перекриють всіх інших, а це не прийнятно.

Мутації. Мутації схожі з розмноженням, з мутантів вибирають кілька особин і змінюють їх відповідно до заздалегідь визначених операцій.

Відбір. На цьому починаємо вибирати з популяції частку тих, хто піде далі. При цьому частку тих, хто «вижив» після нашого відбору, ми визначаємо заздалегідь руками, вказуючи у вигляді параметра. Інші представники видаляються.

Q-навчання. Це алгоритм навчання з підкріпленням для вивчення значення дії в певному стані.

Для будь-якого кінцевого марковського процесу прийняття рішень (FMDP) Q-навчання знаходить оптимальну політику в сенсі максимізації очікуваної вартості загальної винагороди за будь-які та всі послідовні кроки, починаючи з поточного стану [28]. Q-навчання може визначити оптимальну політику вибору дій для будь-якого заданого FMDP, враховуючи нескінченний час дослідження та частково випадкову політику. «Q» означає функцію, яку обчислює алгоритм — очікувану винагороду за дію, виконану в даному стані.

Q-learning у найпростішому вигляді зберігає дані в таблицях [29]. Цей підхід дає збої зі збільшенням кількості станів/дій, оскільки ймовірність того, що агент відвідає певний стан і виконає певну дію, стає дедалі меншою.

Для зменшення простору станів було використане квантування станів.

Тобто стан середовища в певний момент визначається не абсолютною кількістю того чи іншого ресурсу, юнітів або будівель певного типу, а степенем задоволення певних потреб від критично низького до чудового, що дозволило з безперервної кількості можливих значень перейти до дискретного значення можливих станів.

DQN. Це є розвитком ідей Q-навчання, але замість таблиць використовує нейронні мережі: на вхід до нейронної мережі входять параметри, які відповідають за визначення певного стану, узагальнює ці стани та повертає значення нагороди для кожної з можливих дій на виході [30].

Завдяки цьому вирішується головна проблема базової реалізації Q-навчання, а саме зменшення ефективності алгоритму під час збільшення кількості можливих станів та дій.

2.2.3 Оптимальний підхід для навчання в ігровому застосунку

Основні підходи для навчання з підкріпленням, які були визначенні мною були базовий QLearning та DQN.

Перевагами DQN є те що вона генералізує всі стани системи до визначеної кількості та не буде ситуації коли на певний стан не буде відповіді.

Однак є недоліки:

- складність імплементації;
- збільшена складність визначення моделі, яка буде підходити і для навчання і для використання;
- DQN є black box-ом і на етапі навчання і також результатів.

Останній недолік є найбільш критичним в розрізі його використання в ігрових програмах, адже для більшості потребується детермінована або, як мінімум прогнозована поведінка від неігрових персонажів, або систем.

Реалізація через базовий Q-Learning має недолік в тому, що поведінка для тих станів, які не були задіяні під час навчання, поведінка буде або невизначеною, або випадковою.

Перевагами є:

- Відносно нескладна імплементація в порівнянні з DQN.
- Навіть при процесі навчання, який є black box-ом, результати є такими, що можуть бути зрозумілими для гейм дизайнерів.
- Результати роботи алгоритму можуть бути використанні, як основа для реалізація ІШІ традиційними методами.

В задачах, для яких застосовується традиційний штучний інтелект, DQN є більш ефективним. Але в ігрових застосунках DQN має серйозний недолік, а саме, те що з даних отриманих в результаті навчання не можна зробити якийсь висновок щодо конкретних дій системи при певному стані. У зв'язку з цим, для реалізації навчання в ігровому застосунку була обрана базова реалізація Q-навчання.

2.3 Структури даних

У даному розділі представлений до уваги опис важливих структур даних, що були створені для побудови моделі даних ігрового застосунку, які будуть використовуватися для навчання системи керування:

- **TileData**— клас, який зберігає в собі дані про тайл, одиницю побудови мапи.
- **TileFormingData** — клас, який зберігає дані про тип та особливості місцевості певного тайлу та про побудовані на цьому тайлі будівлі.
- **TileWalkableData** — клас, який зберігає дані про прохідність тайлу, а саме чи можна пройти по цьому тайлу та яка вартість цього проходу.
- **TileProductionData** — клас, який зберігає інформацію про можливі дії на цьому тайлі. Обчислення цієї інформації базується на інформації про тип та особливості місцевості та про будівлі на цьому тайлі.
- **ProductionData** — клас, який зберігає інформацію про певну дію, що може бути виконана на певному тайлі. До такої інформації належать необхідні ресурси, необхідна кількість роботи, ресурси які будуть отримані в результаті даної дії та необхідні технології, які повинні бути дослідження для виконання цієї дії.

- **BuildingInfo** — клас, який зберігає інформацію про певну будівлю, що може бути побудована на тайлі. До такої інформації належать необхідні ресурси, необхідна кількість роботи, ідентифікатор будівлі, що буде отримана в результаті, та необхідні технології, які повинні бути дослідження для побудови даної будівлі.
- **ResearchManager** — клас, що зберігає інформацію про дослідження. До такої інформації належать всі можливі технології та соціальні інститути, вже вивченні технології та інститути, поточне дослідження.
- **ResearchPoint** — клас, що зберігає інформацію про конкретне дослідження. До такої інформації належать необхідна кількість поінтів науки та культури для цього дослідження, список технологій, що повинні бути вже дослідженні, щоб почати це дослідження, статус дослідження та список подій, які можуть пришвидшити це дослідження.
- **DataManager** — клас, що оперує даними системи. Даний клас повертає дані для класів, такі як дані про будівлю на основі її ідентифікатора, вираховує можливі побудови для тайлу на основі даних про його тип та особливості місцевості, повертає дані про можливі дії на тайлі.
- **UnitData** — клас, що зберігає дані про юніта, а саме його позицію, чи зайнятий він на поточному ході, список команд, що він має виконати, швидкість руху, продуктивність та інше.
- **QAction** — інтерфейс, який визначає основні функції необхідні для дій, котрі виконуються в рамках Q-навчання. Надає базову реалізацію визначення конкретної оптимальної в відповідь на переданий в нього стан.
- **QState** — клас, що визначає основні параметри, які впливають на визначення стану середовища для алгоритму Q навчання.

- **RewardManager** — клас, який повертає величину умовної нагороди за кожну дію відповідно до поточного стану середовища.

2.4 Проєкт інтерфейсу

Основним критерієм для інтерфейсу в ігрових програмах є зручність його використання та логічність позначень, які використовуються для показу можливих дій. Для стратегічних ігор важливим є також те, щоб одночасно на екрані була розміщена тільки та інформація, яка потрібна в поточний момент часу. В іншому випадку є ризик, що гравець не зможе зосередитися на важливій інформації та процесі гри, а саме, пошук необхідної інформації, буде викликати дискомфорт у гравця.

На рисунках нижче (Рис.14-17) представлені схеми інтерфейсу програмної системи.

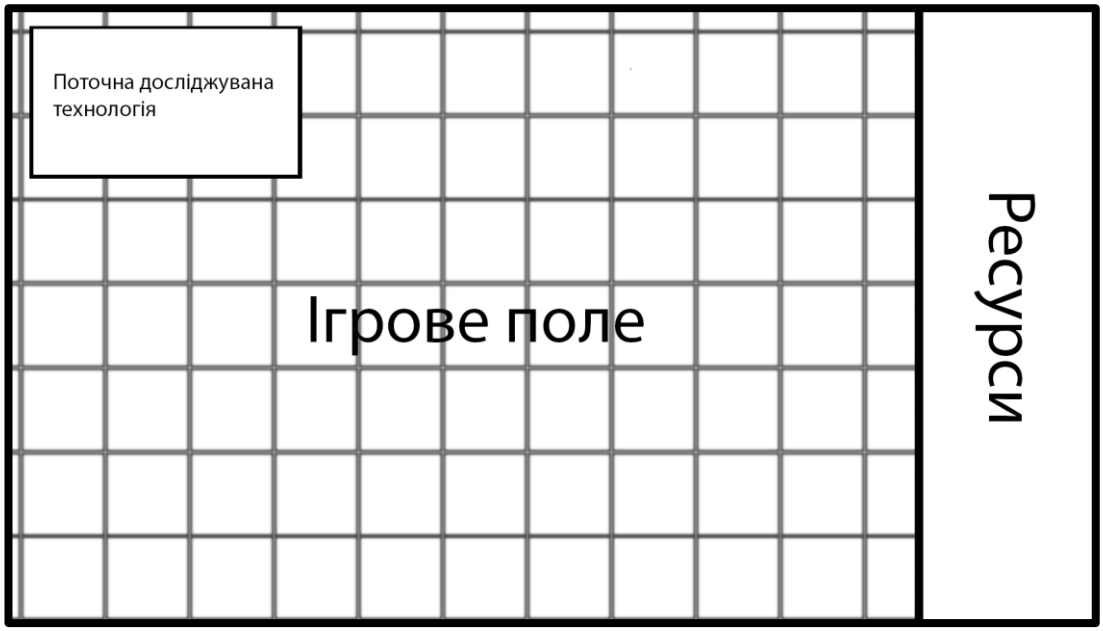

Рис. *14 Схема інтерфейсу головного екрану*

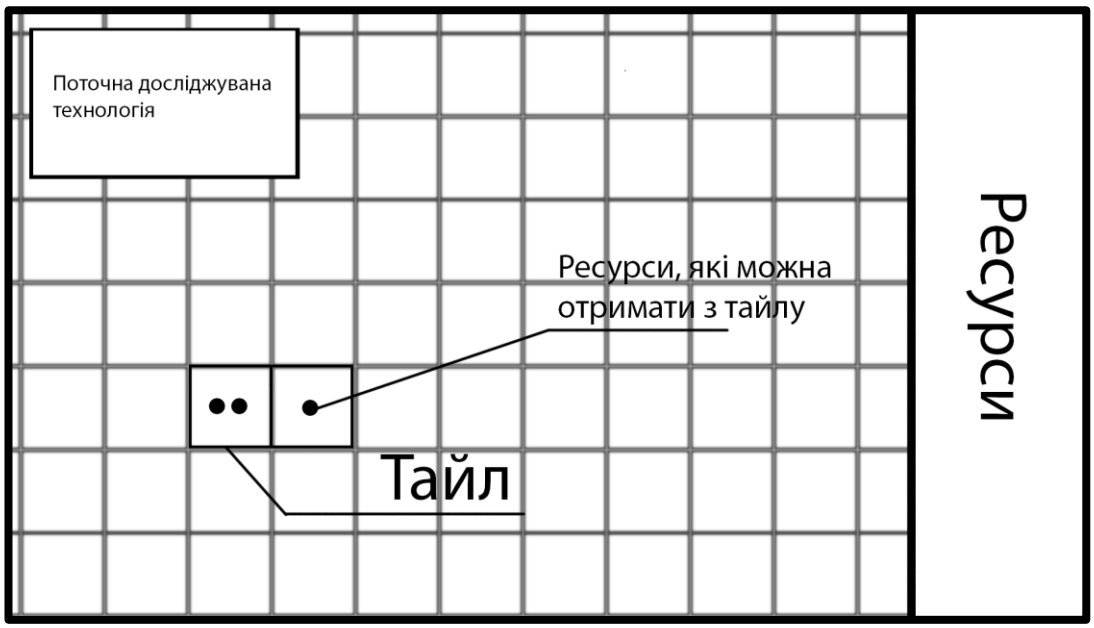

Рис. *15 Схема тайлу мапи*

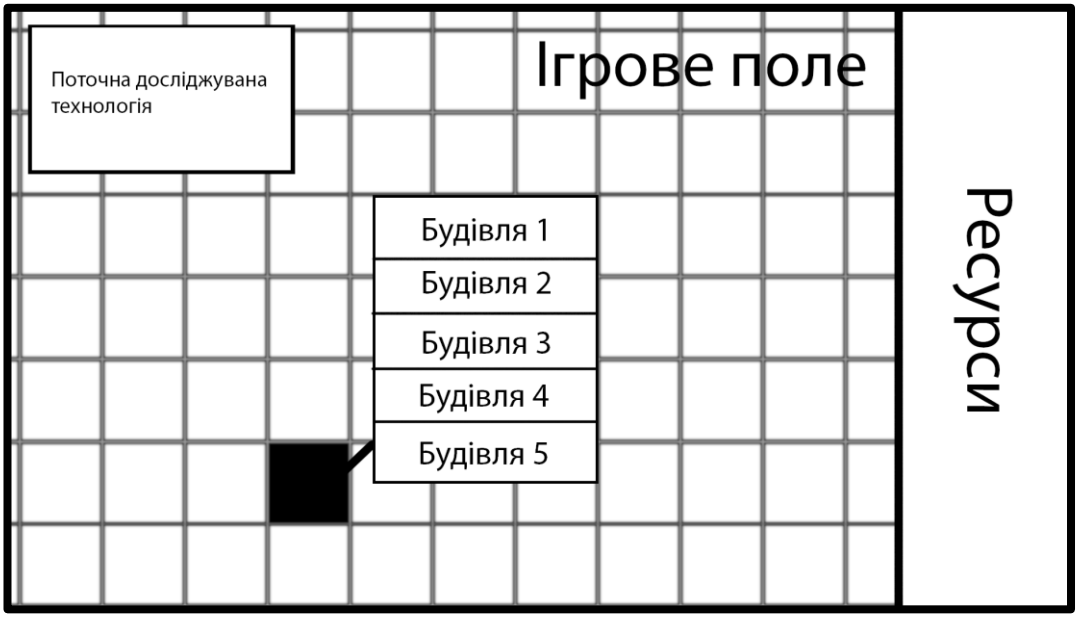

Рис. *16 Схема елементу інтерфейсу для будівництва споруд*

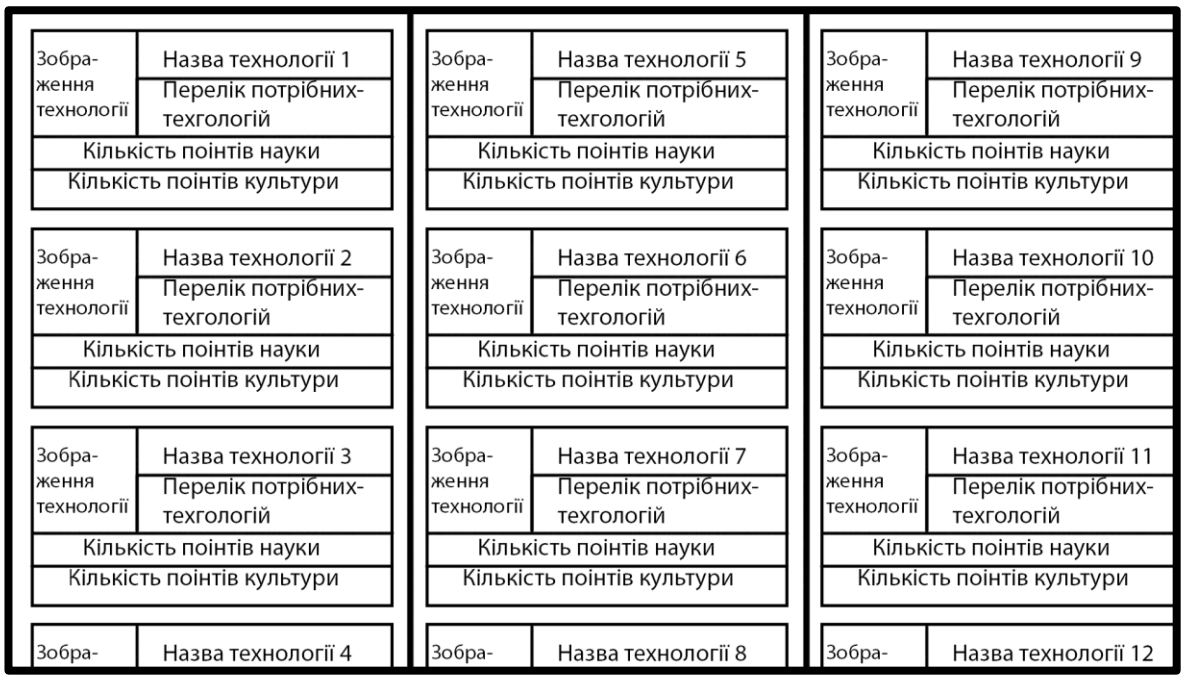

Рис. *17 Схема дерева технологій*

2.5 Висновки з розділу 2

В даному розділі було проведено аналіз інструментів, технологій, бібліотек, які використовуються для побудови ігрового застосунку та ІШІ для нього; аналіз алгоритмів навчання. На основі цього можна сформулювати такі висновки:

1. Визначенні та реалізовані основні алгоритми необхідні для ігрового застосунку з елементами машинного навчання.
2. Навчання традиційного штучного інтелекту та ІШІ має певні відмінності через те що їх результати будуть використані для різних цілей. Тому ефективніший для традиційно ШІ DQN не є найкращим вибором при розробці ігрового застосунку.
3. Визначені цілі ІШІ та обрано оптимальний підхід його реалізації та навчання.

РОЗДІЛ 3 РЕАЛІЗАЦІЯ І ТЕСТУВАННЯ ІГРОВОГО ШТУЧНОГО ІНТЕЛЕКТУ

3.1 Реалізація основного застосунку

Програмна система представляє собою стратегічну гру. Метою гри є розвиток своєї общини, за допомогою обробки тайлів, побудови нових споруд, дослідження нових технологій, щоб виконати одну з умов перемоги:

- відкриття та реалізація глобального наукового проекту;
- зростання кількості населення до певної кількості.

На рисунку 18 зображено основну архітектуру застосунку: головні модулі та основні зв’язки між ними.

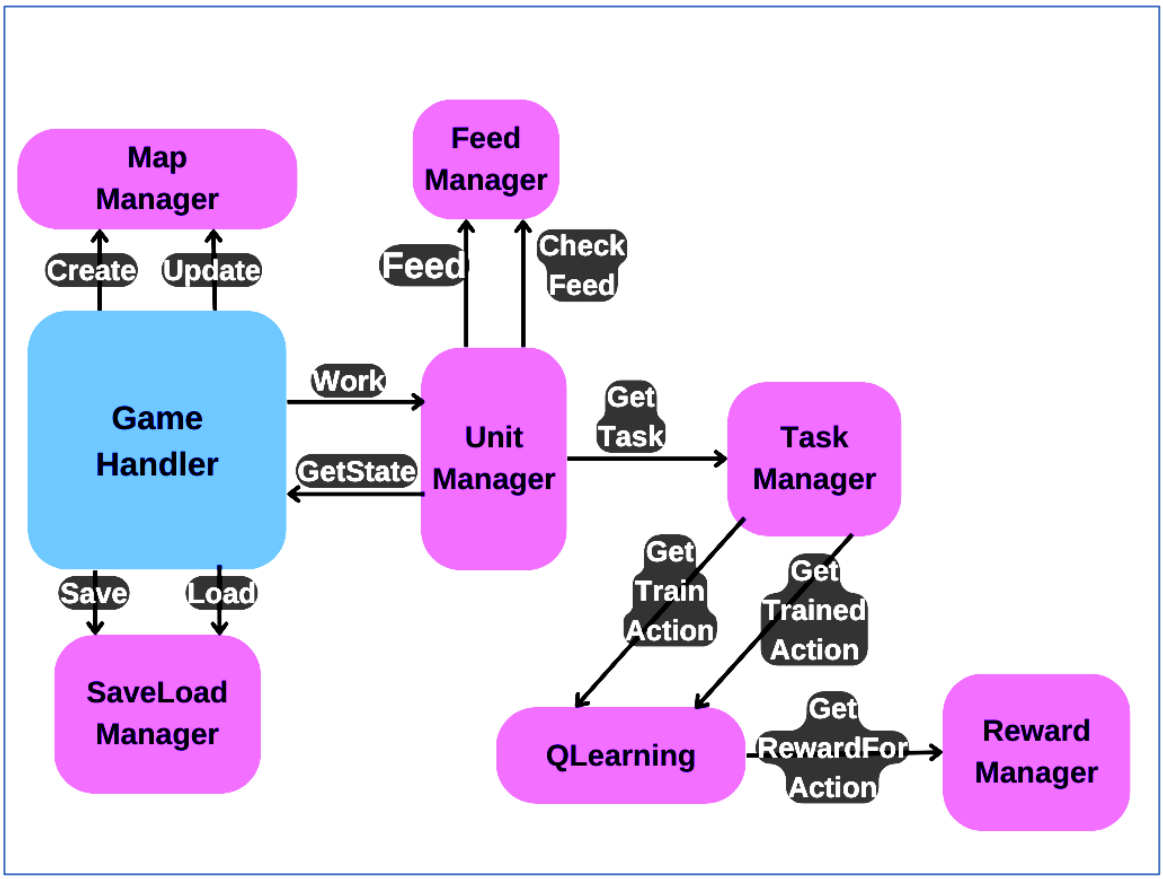

Рис. *18 Архітектура застосунку*

3.1.1 Ігрове поле

Ігрове поле є мапою створеною з тайлів. Кожен тайл є різним типом місцевості.

Поверх карти місцевості накладається мапа додаткових особливостей місцевості: пагорби, ліс, болота тощо.

Після того, як буде згенерована мапа місцевості, перевіряється файл із записаною інформацією про побудовані будівлі, чий рівень накладається поверх згенерованих до цього.

Існує кілька видів особливостей місцевості: пагорби, ліс, болота тощо. Кожна особливість накладає певні модифікатори на клітину, де знаходиться такі як зниження прохідності, надання бонусів або штрафів при обробці клітини.

Певні типи або особливості місцевості можуть надавати доступ до ресурсів. Деякі ресурси дають бонус до їжі або виробництва, деякі є тими, що добуваються та необхідні для певного виробництва, спорудження будівель, тощо.

Споруди зводяться за допомогою юнітів, які були відправлені на їхнє зведення. Споруди можуть давати можливість видобувати певні ресурси, збільшувати ефективність обробки клітини, підвищувати прохідність клітини, надавати додаткові модифікатори для клітини тощо.

Для будівництва споруд необхідна певна кількість виробництва, також можуть знадобитися додаткові ресурси.

Юніти використовуються для обробки клітин, збору ресурсів з клітин, спорудження будівель, роботи в спорудах. Юніти можуть виконувати певну кількість роботи в хід, яка потрібна на тайлах.

На рисунку 19 зображено вигляд мапа на першому етапі розробки.

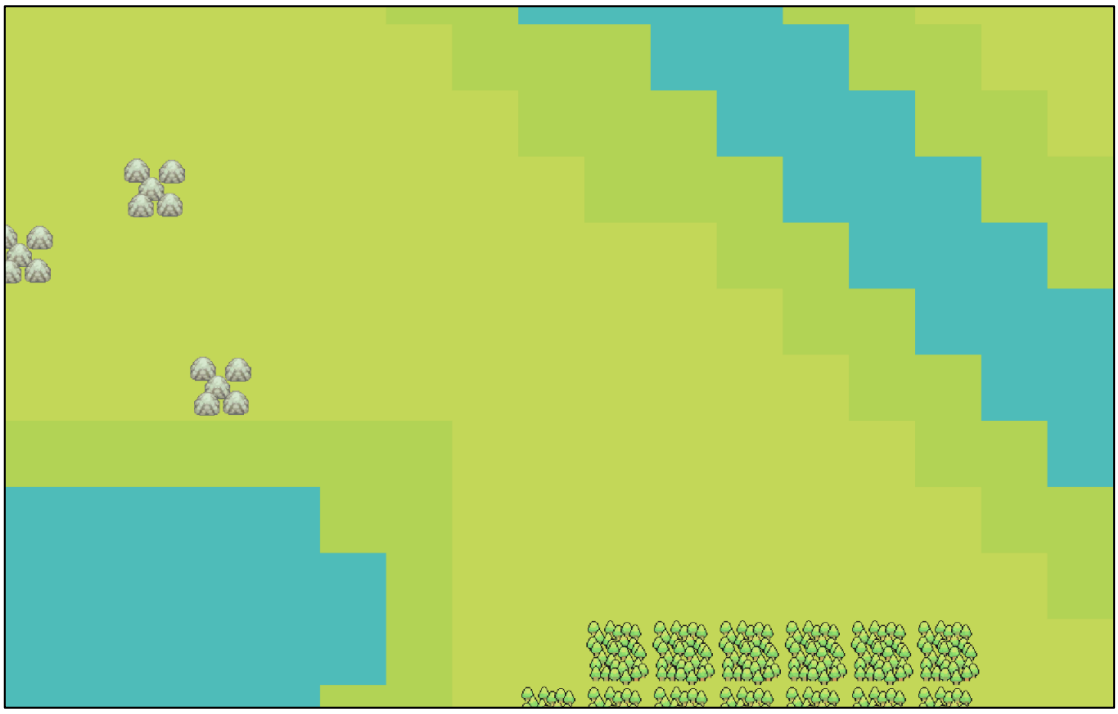

Рис. *19 Початковий варіант побудованої мапи*

Для даного варіанту мапи був реалізований стандартний алгоритм A* з чотирма сусідами для кожного тайлу.

Після аналізу було прийнято рішення додати юнітам можливість руху не тільки по горизонталі або вертикалі, але й по діагоналі. З даним рішенням були зв’язані наступні проблеми: визначити можливість переходу по діагоналі, обчислити вартість переходу по діагоналі.

Для вирішення даних проблем була додана сутність сабтайлу, методи якого дозволити проводити потрібні розрахунки. Після додання сабтайлу, на логічному рівні, мапа складалася з восьмикутних основних тайлів та ромбовидних сабтайлів.

Після реалізації даної функціональності на логічному рівні, був модифікований алгоритм A* з використанням сабтайлів.

Проте після реалізації сабтайлів на логічному рівні, з’явилися проблеми коли юніти візуально проходили по непрохідним тайлам. Тому було вирішено додати композитні тайли.

Кожен тайл розбитий на зони, які будуть зафарбовуватися в залежності від сусідніх тайлів, та значень які знаходяться в їх зонах.

На рисунку 20 зображена розбивка тайлу на зони.

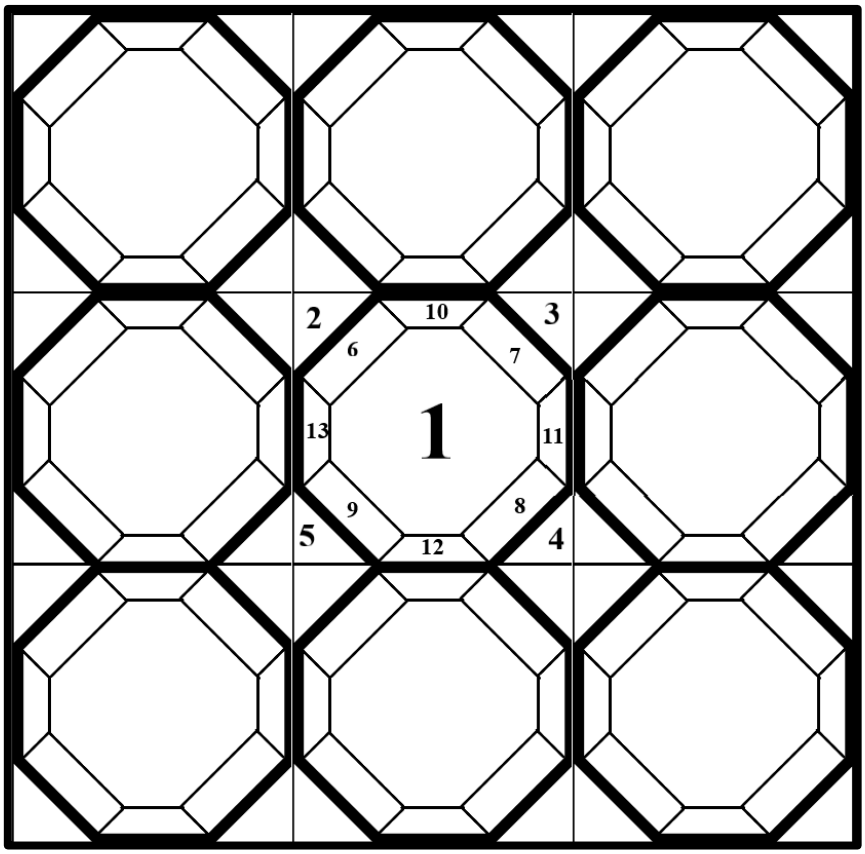

Рис.*20 Схема поділу тайлу на зони*

Зона 1 зафарбовується в залежності від типу місцевості тайлу. Зони 2-5 заповнюються, як перехідні стани між відповідними тайлами з’єднаними по діагоналі, які визначаються лівим або правим правилом з’єднання тайлів. Зони 6-9 зафарбовуються, як перехідний стан між тайлом та перехідними зонами 2-5 відповідно сусідству. Зони 10-13 зафарбовуються, як перехідний стан між тайлом та відповідним сусідом.

На рисунку 21 зображена частина мапи після реалізації зонування тайлу.

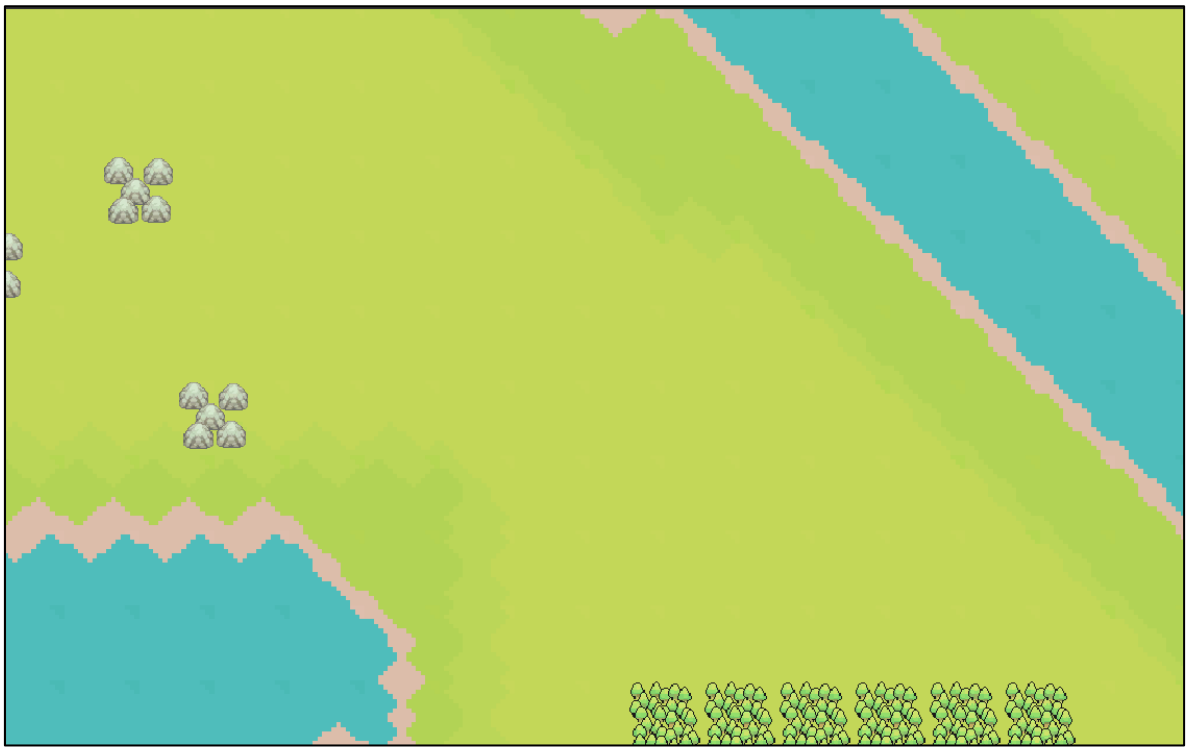

Рис. *21 Мапа з зонованими тайлами*

3.1.2 Система харчування

Була реалізована система харчування в поселенні, яка спрямована на контролювання розмноження юнітів .

Кожен цикл харчування триває певну визначену кількість ходів, наразі це 50. Після того, як пройде визначена кількість ходів відбувається зняття необхідної кількості їжі. Необхідна кількість їжі розраховується по кількості юнітів.

Є три стани індикатору ситості поселення(рис. 22): їжі не вистачає, їжі вистачає, є надлишок їжі.

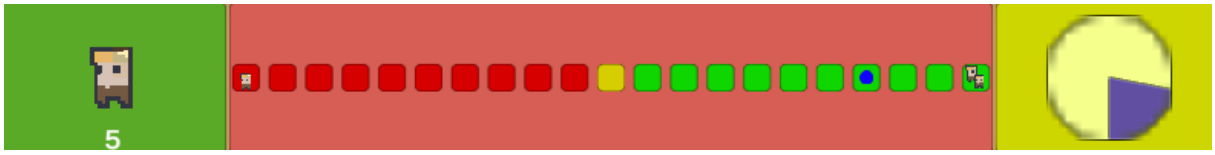

Рис.*22 Індикатор системи харчування*

В залежності від кількості запасів їжі поселення індикатор ситості змінює свою позицію. Якщо індикатор знаходиться протягом десяти циклів харчування в положенні дефіциту їжі то, поселення втрачає одного юніта та індикатор збільшується на 5 пойнтів. Якщо індикатор знаходиться протягом десяти циклів харчування в положенні надлишку їжі то, поселення отримує одного юніта та індикатор зменшується на 5 пойнтів.

Якщо два цикли харчування підряд мають однаковий статус, то індикатор зміщується не на одну позицію, а на дві, що прискорює або втрату, або отримання нового юніта.

3.1.3 Система досліджень

Система досліджень представляє собою дерево технологій вузли якого відкриваються один за одним.

Кожен вузол може досліджуватися, якщо досліджені всі вузли які йдуть перед ним та з якими він зв’язаний.

Кожне дослідження для завершення потребує певну кількість поінтів науки та культури. Також дослідження можуть бути прискоренні за рахунок певних визначених дій, які визначенні для кожного дослідження.

Після завершення кожного дослідження відкриваються або нові ресурси які можна добувати або оброблювати, або нові споруди, які можна побудувати.

3.1.4 Система наказів

Наказ, в контексті даної ігрової системи, є дією, яку ми делегуємо юніту виконати.

Є два основних типи наказів: обробка/добування ресурсу та будівництво будівель.

Якщо наразі всі юніти є зайнятими то накази додаються в чергу виконання та будуть взяті в роботу після того, як з’явиться вільний юніт.

На рисунку 23 зображена черга наказів.

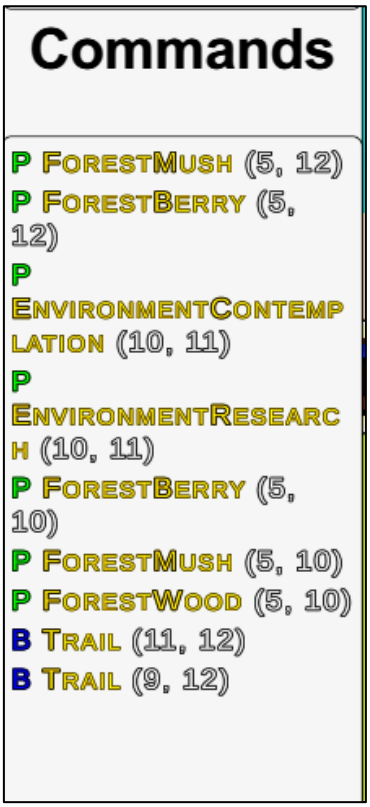

Рис. *23 Черга наказів*

Виконання конкретних наказів забезпечуватиметься штучним інтелектом, екземпляр якого буде в кожному з юнітів.

На рис. 24 зображено розподіл наказів між юнітами.

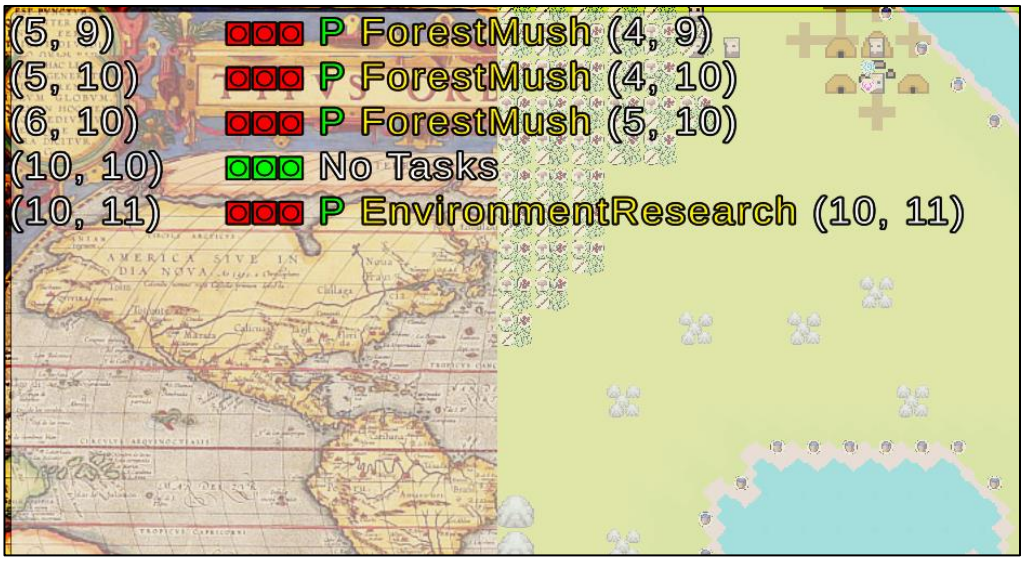

Рис. *24 Розподіл наказів між юнітами*

3.1.5 Система збереження / завантаження

Для збереження проміжних досягнень гравця була розроблена система збереження та завантаження.

Крім збереження інформації про ігрове поле та юнітів була додана функція збереження та завантаження інформації необхідної для навчання ІШІ. Це дозволить в майбутньому переносити навчену систему керування без необхідності нового навчання системи.

На рис. 25 зображено елемент через який відбувається збереження та завантаження необхідної інформації.

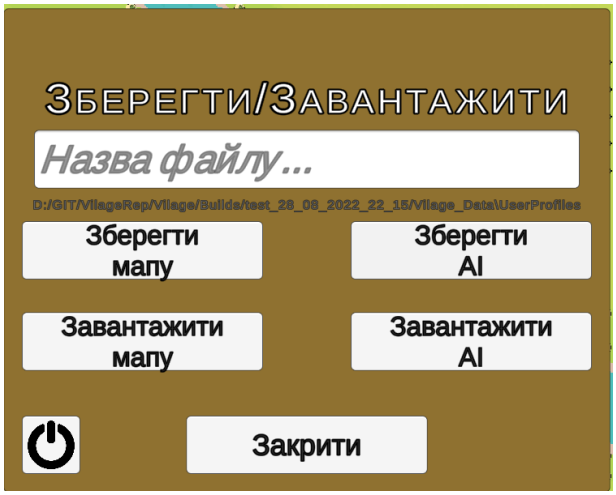

Рис. *25 Елемент для збереження та завантаження ігрових даних та інформації ІШІ*

3.1.6 Тестування основного функціоналу ігрового додатку

Тестування проводилося мануально. На фінальному етапі роботи помилок роботи програми не було виявлено.

Ілюстрації представленні нижче (Рис.26-33), демонструють результати які були отримані.

Рис. *26 Побудована мапа з доступними діями на початку*

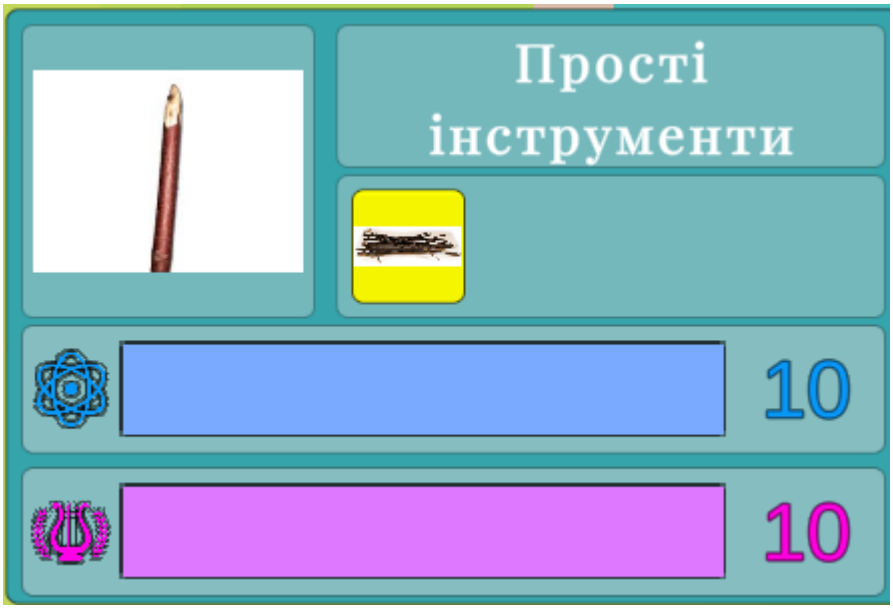

Рис. *27 Елемент інтерфейсу з інформацією про дослідження*

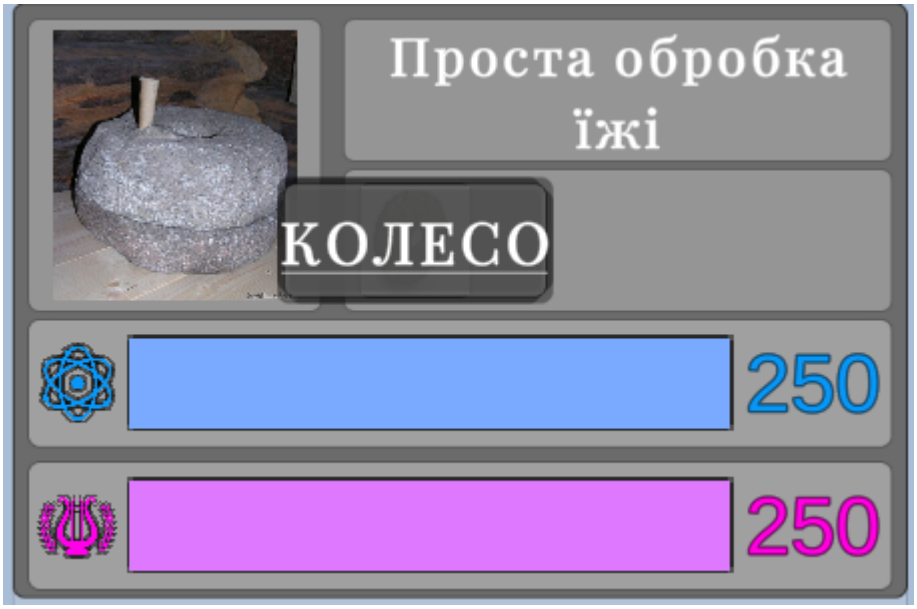

Рис. *28 Елемент інтерфейсу з інформацією про дослідження з показаними підказками*

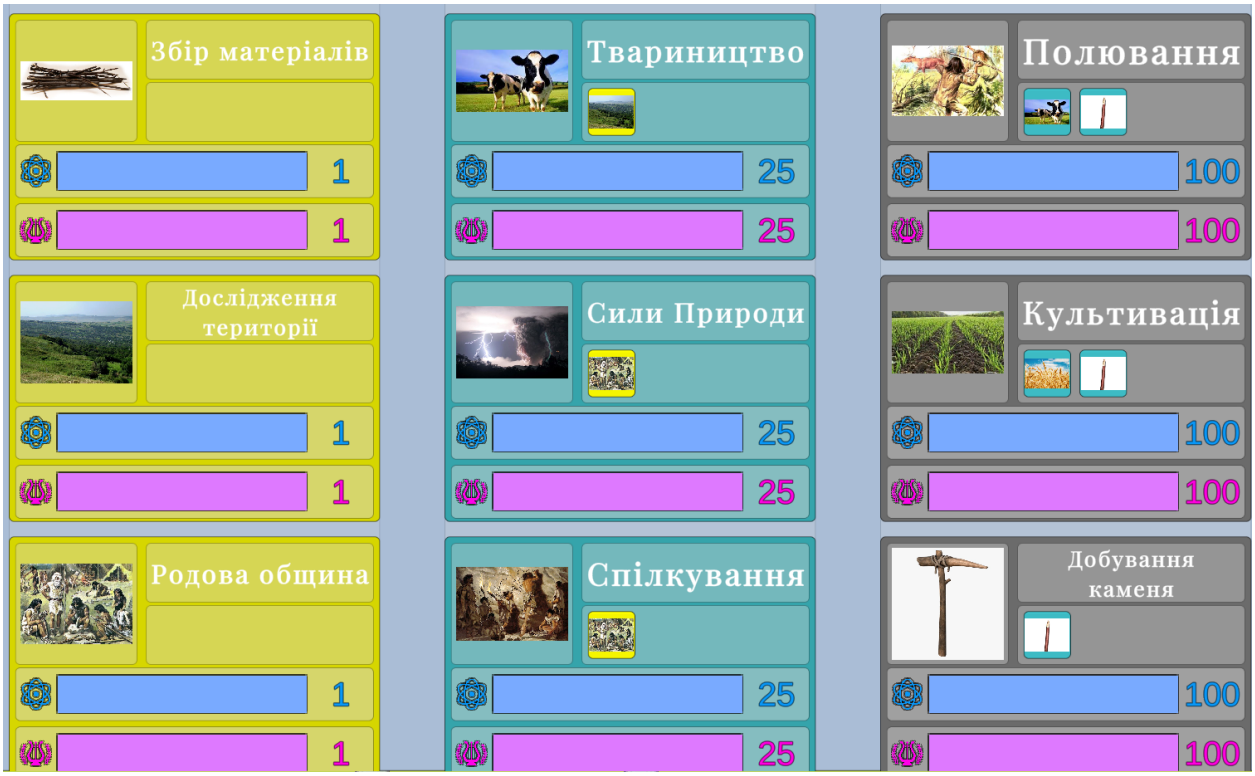

Рис. *29 Вікно досліджень*

Рис. *30 Елемент інтерфейсу, що показує інформацію про можливі побудови*

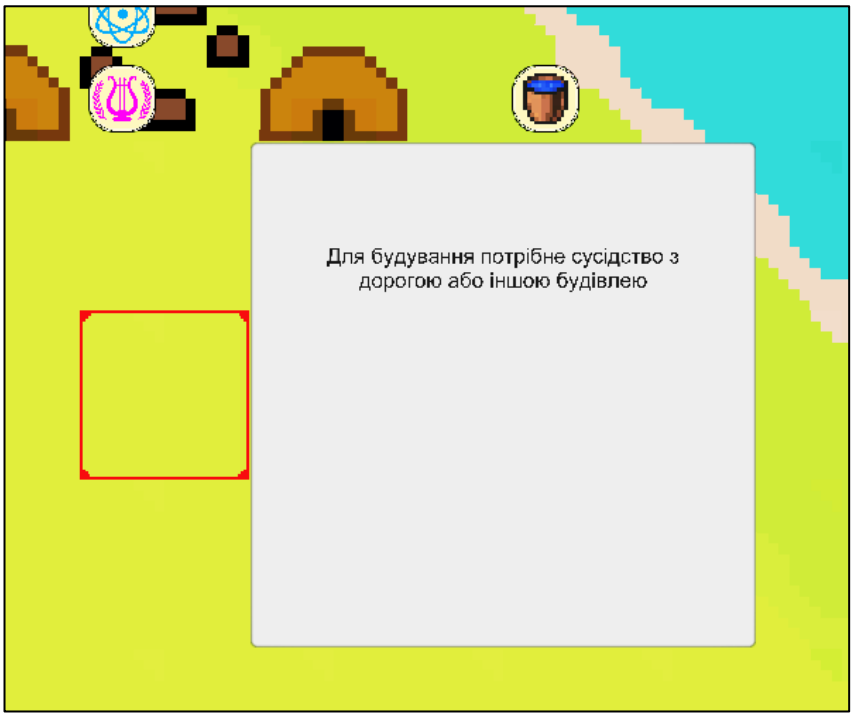

Рис. *31 Елемент інтерфейсу, що показує інформацію неможливість дії та з поясненням причини*

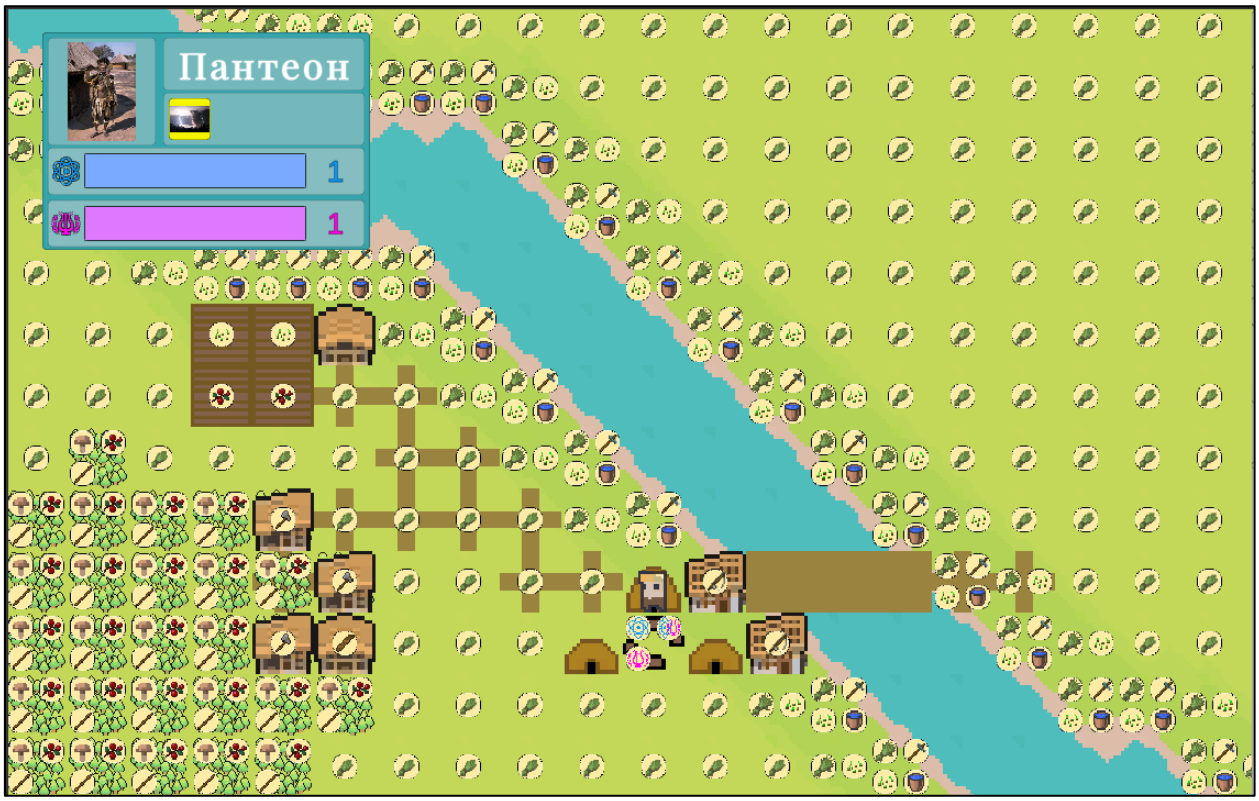

Рис. *32 Мапа після відкриття технологій та побудови певних споруд*

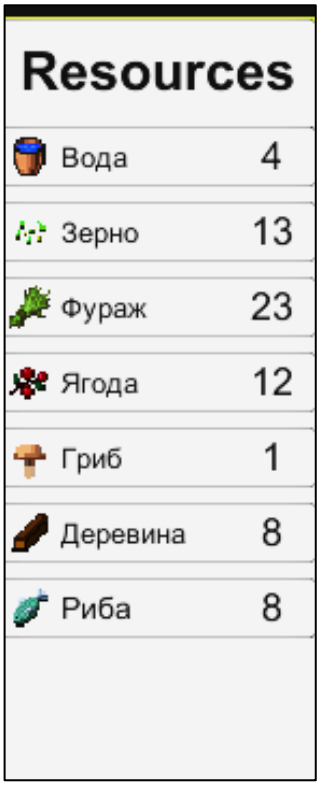

Рис. *33 Список здобутих гравцем ресурсів*

3.2 Реалізація ІШІ в застосунку

Після дослідження можливих варіантів реалізації системи навчання в ігровому застосунку вибір зупинився на Q-навчанні. Він поєднує необхідні якості як для процесу навчання системи так і для його використання в ігровому застосунку.

3.2.1 Реалізація алгоритму Q-навчання

В ігровому застосунку було реалізовано ІШІ з використанням алгоритмів Q-навчання.

Заохоченням для неї буде:

- надлишок їжі (більше, ніж необхідно для виживання населення);
- будівництво споруд;
- наукове відкриття;
- вивчення культурних інституцій;
- розмір населення.

Штрафами будуть:

- дефіцит ресурсів;
- втрата юнітів;
- довга відсутність наукового та культурного розвитку.

На кожному кроці, посилаючись на вхідні дані, система керування розставляє пріоритети на напрямки розвитку:

- харчове забезпечення;
- промислове виробництво;
- наукові відкриття;
- культурний розвиток.

Після визначення пріоритетів юніти розподіляються серед відомств у кількості пропорційному пріоритету.

Після отримання у своє розпорядження юнітів відомство з допомогою евристик визначає найефективніший наказ і посилає робітника його виконувати.

У міру відкриття технологій, інститутів, спорудження нових будівель, до кожного відомства додаватимуться додаткові можливі накази.

Ефективність та корисність кожного з наказів визначається відповідно до поточного стану поселення.

Була розроблена система нагород, яка повертає певну кількість пойнтів для кожного з можливих наказів виходячи з поточного стану поселення. В лістингу 7 відображений клас нагороди «Reward», який відповідає за представлення нагороди по чотирьом основним курсам розвитку поселення.

Лістинг *7 Клас нагороди «Reward»*

```
public class Reward
{
    public double FeedReward = 0;
    public double ProductionReward = 0;
    public double ScienceReward = 0;
    public double CultureReward = 0;
}
```

В лістингах 8-9 зображенні основні методи класу «RewardManager».

Лістинг *8 Методи для оцінки нагороди за ресурси*

```
public Reward RewardsForProduct(Production product)
    {
        var reward = new Reward();

        product.data.RequiredResources.ForEach(x => {
reward.Minus(RewardsForProduct(x)); });
```

```
        product.data.ResultResources.ForEach(x => {
reward.Add(RewardsForProduct(x)); });

        return reward;
    }
    Reward RewardsForProduct(Product product)
    {
        var reward = RewardsForResource(product.Resource);
        reward.Multiple(product.Count);
        return reward;
    }

    Reward RewardsForProduct(List<Product> products)
    {
        var reward = new Reward();

        products.ForEach(x => {
            var temp = RewardsForResource(x.Resource);
            reward.Multiple(x.Count);
            reward.Add(temp);

        });

        return reward;
    }

    Reward RewardsForResource(Resource resource)
    {
        return
RewardsForResources((DataManager.Resources)System.Enum.Pars
e(typeof(DataManager.Resources), resource.Id));

         }
```

Лістинг 9 *Метод для оцінки нагороди за обробку тайлу*

```
public List<(Reward, string)>
RewardsForTileProductionData(TileProductionData product,
bool gipoteticAvailableCheck = true)
{
    if(count !=
CurrentPlayerData.CurrentPlayer.ResearchManager.ExploredRes
earches.Count)
    {
        count =
CurrentPlayerData.CurrentPlayer.ResearchManager.ExploredRes
earches.Count;
```

```
            productionsidRewards.Clear();
        }
        var ids = "";

        product.AllExploredProductions.ForEach(x => { ids +=
x.id; });

        if(!productionsidRewards.ContainsKey(ids))
        {
            var rewards =
product.AllExploredProductions.Select(x => {
                return (RewardsForProduct(x), x.id);
            }).ToList();
            productionsidRewards.Add(ids,
Helper.DeepClone(rewards));

        }

        if(gipoteticAvailableCheck)
        {
            return Helper.DeepClone(productionsidRewards[ids]);
        } else
        {
            return
Helper.DeepClone(productionsidRewards[ids]).FindAll(x =>
DataManager.sharedInstance.gipoteticProducionAvailable(x.It
em2));
        }
}
```

Враховуючи, що дані методи будуть викликатися масово під час навчання системи, для підвищення ефективності роботи були використані методи кешування інформації, яка буде незмінна в рамках одного запуску.

Розроблений інтерфейс «IQLearningProblem»(лістинг 10), який визначає необхідні методи для класу, задачу якого має вирішувати штучний інтелект, який буде навчатися за допомогою QLearning алгоритму.

Лістинг *10 Інтерфейс «IQLearningProblem»*

```
public interface IQLearningProblem
{
    int NumberOfStates { get; }
```

```
    int NumberOfActions { get; }
    int[] GetValidActions(int currentState);
    double GetReward(int currentState, int action);
    bool GoalStateIsReached(int currentState);

    void performAction(int action);
}
```

Був розроблений клас «QLearning». Він відповідає за навчання штучного інтелекту, за допомогою q-таблиць, в яких зберігається інформація про нагороду, для певних дій, в певному стані.

На етапі навчання дії для стану обираються випадково, але після завершення процесу навчання, дії обираються опираючись вже на отримані дані визначаючи найефективнішу з можливих дій для певного стану.

В лістингу 11 зображено основні поля необхідні для класу «QLearning».

Лістинг *11 Клас «QLearning»*

```
public class QLearning
{
    private System.Random _random = new System.Random();
    public double _gamma;
    public double Gamma { get => _gamma; }

    public List<DoubleDataRow> _qTable;
    public List<DoubleDataRow> QTable { get => _qTable; }

    public IQLearningProblem _qLearningProblem;

    public double LF = 0.5;
    public double DF = 0.5;

    public QLearning(double gamma, IQLearningProblem
qLearningProblem)
{
    _qLearningProblem = qLearningProblem;
    _gamma = gamma;

    _qTable = new List<DoubleDataRow>();
```

```
    for (int i = 0; i < qLearningProblem.NumberOfStates;
i++)
    {
        _qTable.Add(new DoubleDataRow());
        for (int j = 0; j <
qLearningProblem.NumberOfActions; j++)
        {
            _qTable[i].data.Add(0);
        }
    }
}
```

…

Завдяки параметрам gamma, LF та DF можна налаштовувати процес навчання системи.

В залежності від їх значень система буде віддавати більшу перевагу або старим даним, які вже вивченні або ж новим на кожному кроці навчання.

В лістингу 12-13 зображені методи класу «QLearning», які відповідають за обрання необхідної дії під час навчання та після завершення процесу навчання системи.

Лістинг *12 Метод «TakeTrainAction*

```
public int TakeTrainAction(int currentStateCode)
{
    var state = new QState();
    state.setupFromCode(currentStateCode);

    var currentState = state.getIndex();

    var validActions =
_qLearningProblem.GetValidActions(currentState);
    int randomIndexAction = _random.Next(0,
validActions.Length);
    int action = validActions[randomIndexAction];

    double saReward =
_qLearningProblem.GetReward(currentStateCode, action);
    double nsReward = _qTable[action].data.Max();
    double currentReward =
_qTable[currentState].data[action];
```

```
    double qCurrentState = (1.0 - LF) * currentReward + LF
* (saReward + DF * nsReward);
    _qTable[currentState].data[action] = qCurrentState;
    _qLearningProblem.performAction(action);
    return UnitService.Handler.getCurrentState().getCode();
}
```

Лістинг *13 Метод «TakeAction»*

```
public void TakeAction(int currentStateCode)
{
    var state = new QState();
    state.setupFromCode(currentStateCode);

    var currentState = state.getIndex();

    int action =
_qTable[currentState].data.ToList().IndexOf(_qTable[current
State].data.Max());
    _qLearningProblem.performAction(action);
}
```

Крім безпосередньо повернення поточної дії для виконання в методі «TakeTrainAction» відбувається процес навчання системи.

В залежності від обраної дії, нагороди для неї та визначеним параметрам навчання коригуються значення в q-таблиці, які потім будуть використовуватися для обрання пріорітетної дії після закінчення процесу навчання.

В лістингу 14 зображений клас «VilageProblem», який реалізує інтерфейс «IQLearningProblem» та визначає необхідні методи для Q навчання, для штучного інтелекту управління поселенням.

Лістинг *14 Клас «VilageProblem»*

```
class VilageProblem : IQLearningProblem
{
    public QAction[] actions = { new ProduceFeedAction(),
new ProduceResourceAction() };
```

```
    public int[] actionsId = { 0, 1};
    public int NumberOfStates => 7 * 7 * 7 * 7 * 7 * 2 * 2;
    public int NumberOfActions => 2;
    public double GetReward(int currentStateCode, int
action){
    var state = new QState();
    state.setupFromCode(currentStateCode);

    var currentAction = actions[action];

    var specAction =
currentAction.GetPrefferedActionReward(state);

    return specAction;
}

public int[] GetValidActions(int currentStateCode)
{
    return actionsId;
}

public bool GoalStateIsReached(int currentStateCode)
{
    bool population = currentStateCode % 10 != 0;

    bool allResearch = (currentStateCode / 10) % 10 != 0;

    return population || allResearch;
}

public void performAction(int action)
{
```

```
    UnitService.Handler.performAction(actions[action]);
}
```

3.2.2 Навчання системи керування

Для визначення оптимальних параметрів навчання було виконано багатократний запуск системи навчання та порівняння отриманих під час запуску нагород.

На рис. 34 зображення елемент для управління процесом навчання.

Рис. *34 Елемент управління процесом навчання*

Після визначення оптимальних параметрів було багатократне повторення навчання.

Через присутність впливу рандомних значень під час процесу навчання та обмеженості ресурсів на яких проводиться навчання(неможливість навчати систему безкінечну кількість кроків) були використанні різні підходи до навчання.

Фінальний підхід включає в себе почергову зміну періодів навчання та періодів роботи системи на вже отриманих даних.

Ефективність системи була протестована за рахунок запуску навченої системи в умовах різної конфігурації мап.

З пулу найефективніших конфігурацій навченої системи була обрана та, яка показала себе з достатньою ефективністю (кількість ходів до досягнення

мети, кількість ходів підтримання стану системи в умовах дефіциту ресурсів) в різних умовах.

3.3Висновки з розділу 3

В даному розділі було описано реалізацію ігрового застосунку, ІШІ та процес навчання. На основі цього, можна зробити такі висновки:

1. Реалізована основа ігрового застосунку з генерацією мапи;
2. Додані усі необхідні системи для керування ігровим застосунком та підтримання ігрового процесу;
3. Протестований основний функціонал ігрового застосунку;
4. Реалізована система навчання за допомогою алгоритму Q-навчання;
5. Реалізована система управління процесом навчання та отримання даних про процес навчання;
6. Визначенні оптимальні значення для параметрів навчання;
7. Навчена система керування юнітами;
8. Протестована працездатність навченої системи в різних умовах.

РОЗДІЛ 4 ДОСЛІДЖЕННЯ РЕЗУЛЬТАТІВ ВИКОРИСТАННЯ СИСТЕМ ШТУЧНОГО ІНТЕЛЕКТУ В ІГРОВИХ ЗАСТОСУНКАХ

4.1 Відмінності традиційних систем штучного інтелекту від ігрового штучного інтелекту

Основні відмінності традиційних систем штучного інтелекту від ігрового штучного інтелекту випливають з цілей використання цих систем.

Якщо для традиційного ШІ зазвичай це «інтелектуальне» вирішення певної задачі то ІШІ використовується для створення ілюзії інтелекту в поведінці NPC.

Тобто, якщо для традиційного ШІ не менш важливою є «інтелектуальність» вирішення задачі то для ІШІ головним є щоб вирішення задачі виглядало інтелектуальним та, досить часто, таке вирішення давало певний результат.

4.1.1 Проблеми при використанні машинного навчання в ігрових застосунках

Використання різних підходів розробки ігор великими компаніями накладають додаткові обмеження на використання, важливим з яких є час імплементації.

Як показала реалізація системи керування, яка підтримує навчання та саму систему, її навчання навіть для невеликого проекту тягне за собою великі витратити часу та включає певні етапи, які не можуть бути розділені між кількома розробниками.

Додатковими факторами не зі сторони бізнесу, а саме зі сторони розробників є те, що різними компаніями використовуються не лише загальновідомі та загальнодоступні ігрові рушії, а й рушії власної розробки.

Це означає, що навіть при можливості реалізації системи, яку можна було б використовувати в багатьох проектах, кожній компанії довелося б це

робити власноруч, або як мінімум власноруч реалізовувати адаптер між такою системою та власним рушієм, що може виявитися навіть складнішою задачею в залежності від обставин.

Також важливим факторам є те, що зазвичай при машинному навчанні дана система залишається для розробників black box-ом, тобто неможливо, або надто складно, заздалегідь передбачити всі реакції даної системи, що для певних ігор є критичним. Як наслідок з цього, можлива ситуація, коли при необхідності змінити поведінку при певному стані системи, це буде неможливо або потягне за собою значні витрати в залежності від розміру проекту.

Також однією з причин, чому складне використання машинного навчання в ігрових програмах, є те, що ігри часто піддаються змінам, і при кожній з них треба перенавчати систему та перетестовувати її взаємодію з іншими системами. Якщо ж навчати на останньому етапі, то це призведе до збільшення вартості розробки, яка може багатократно збільшиться, якщо будуть знайдені проблеми в цій системі. Якщо вводити в майже готовий продукт нову підсистему, — це тягне за собою великі витратити на імплементацію, переконфігуровування та тестування.

4.1.2 Оцінка доцільності використання машинного навчання в ігрових застосунках

Доцільність використання машинного навчання в грі визначається метою, для якої вводять неігрового персонажа (NPC) або систему в цю гру.

Для визначення критеріїв, за яким можна оцінити доцільність використання машинного навчання в ігрових застосунках, були проаналізовані існуючі рішення, а саме ігри різних жанрів. Під час аналізу був визначений спектр задач, які виконують неігрові персонажі та системи керування ними, було визначено рівень свободи вибору дій, які надаються неігровим персонажам та можливі варіанти використання машинного навчання.

Так для сюжетної гри не можна дати повну свободу дій для неігрових персонажів тому що , якщо її надати, може не відбутися той сюжет, який

запланований і тоді мета гри може бути не виконана.

Так для NPC, який виконує роль перешкоди на шляху гравця, можна дати певну свободу, в певному діапазоні дій, які він може виконувати, проте не можна дати свободу на всі дії. Тобто можна дозволити йому ті дії, які обмежені діапазоном тих, що підходять для даної ситуації і для даного стану, тобто такі дії, які призведуть до необхідного результату.

Якщо ж основна ідея гри це якраз взаємодія з механікою гри безпосередньо, наприклад, для реалізації настільних ігор таких як шахи, шашки або го, то тут машинне навчання дуже доцільне.

По-перше – це звільняє розробника від необхідності визначення всіх можливих станів системи.

По-друге – кількість таких можливих станів може бути надто великою, щоб їх усі передбачити та закодувати відповідь на них.

По-третє – завдяки машинному навчанню обирається найкраща стратегія, або така, що найбільше відповідає допустимому рівню виконання поставленої задачі.

Так для вищезгаданих шахів, шашок та го було б недоцільно прописувати відповіді на всі можливі стани тому, що їх надто багато.

Також пластом ігор, для яких доцільно використовувати машинне навчання, є ігри жанру стратегія.

В такій грі є певна кількість станів та набір можливих дій. Ціль такої гри, зазвичай, полягає в тому, щоб перемогти умовного ворога використовуючи доступні дії.

Тобто можна сказати що типова стратегія є розширеною моделлю шахів, з більшою кількістю можливих дій та станів, але суть якої є те, що безпосередньо зіштовхується уміння використовувати ці механіки гравцем та його опонентом чи середовищем гри.

Тому для стратегій є доцільним використання машинного навчання для NPC або систем, якщо кількість станів системи така, що створити систему на-

вчання буде легше ніж передбачити всі стани, або якщо ці стани можуть розширюватися. У цих випадках не треба буде дописувати нові правила, а лише перезапустити процес навчання.

Також машинне навчання доцільно використовувати в іграх, для яких не треба обов'язкове виконання певних визначених дій на відповідь певним станам системи. Тобто, якщо можна дати свободу дій нашій системі або NPC, то машинне навчання підходить, бо воно генералізує всі можливі стани, визначає відповіді на них та дозволяє знайти вигідну стратегію дій.

На доцільність використання машинного навчання в комп'ютерній грі також впливають дії, яким потрібно навчити NPC або систему. Так для стратегії немає сенсу використовувати машинне навчання для переходу юнітів з точки А в точку В оскільки, основний сенс гри полягає в ефективному використанні ресурсів та юнітів, а не процес їх переміщення. Якщо ж це гоночна гра або симулятор їзди, то є сенс давати можливість системі навчатися керувати безпосередньо процесом переміщення, оскільки це є основною механікою та складовою мети такої гри.

4.2 Підходи до навчання системи з використанням QLearning

Кількість кроків необхідних для навчання напряму залежить від кількості можливих станів системи та можливих дій, які може виконувати гравець.

Так, якщо у грі велика кількість станів та можливих дій, то за малу кількість кроків система пройде лише малу частину з них і не будуть відомі ті стани, в які система не перейшла та дії, які не виконала, то відповідно ефективність системи не буде максимальною.

Якщо ж використовувати надто багато кроків, то може бути кілька проблем:

- Збільшення вартості навчання, адже в залежності від системи, витрати часу та ресурсів на один крок можуть відрізнятися, та в певних ситуаціях бути високими і вони будуть багатократно зростати від на одну ітерацію навчання.
- Зменшення ефективності на пізніших кроках у випадках, коли їх кількість надлишкова.
- Якщо можливі дії гравця змінюються з плином часу та розвитку гри, надлишкова кількість кроків може зорієнтувати систему на умови пізньої роботи, що буде абсолютно неефективним на ранніх етапах розвитку гри.

Для навчання даної системи керування були використані кілька підходів:

1. Статична кількість кроків безперервного навчання

Перший та найпростіший підхід — це запуск навчання від початку та на певну кількість кроків.

Перевагами цього підходу є:

- простота реалізації керування;
- не потрібні додаткові системи слідкування за нею;
- мінімально можлива кількість необхідних додаткових даних;
- мінімальна необхідна кількість часу на одну ітерацію навчання.

Недоліками цього підходу є:

- важко пройти таким чином всі стани;
- якщо проходження всіх станів є обов’язковою умовою кількість кроків стає великою та нівелює переваги по оптимізації часу та ресурсів.

Для даної системи керування оптимальною кількістю кроків при такому підході виявилося 2000-2500. При меншій кількості частина можливостей залишалась незадіяними, при більшій не було приросту ефективності роботи системи.

2. Чергування кроків навчання та кроків використання системи в одній ітерації навчання

Відмінність від першого підходу в тому, що після певної кількості кроків навчання йдуть кроки, коли система використовує проміжні результати так, як використовувала б на фінальний результат. Це дозволяє системі зробити певну кількість кроків, які б не дали особливих змін в даному контексті, та дозволяє устаткувати зміни та підготувати систему до нового етапу навчання без скидання стану системи та перезапуску ітерації.

Перевагами цього підходу є:

- За результатами запуску покривається більша кількість станів ніж при безперервному навчанні з однаковою кількістю кроків.
- Етапи без навчання наближують процес до реального, коли гравцеві для переходу на наступний етап треба певний час, використати вже отримані знання та навички.
- Після встановлення ефективної конфігурації змін кроків навчання та кроків базової роботи система показує в середньому кращі результати ніж при безперервному навчанні при однакових витратах часу.

Недоліками цього підходу є:

- В порівняні з безперервним навчанням потребує розробки додаткової системи керування, яка дозволяє перемикати систему між навчанням та звичайним використанням.
- Потребує більше часу на встановлення ефективної конфігурації виконання.
- Трохи збільшується час виконання одного кроку через необхідність виконувати системою керування необхідні перевірки та перемикання. При малих кількостях кроків вплив малопомітний, але в системах, які потребують великої кількості кроків навчання, це може призвести до значного збільшення однієї ітерації.

Для даної системи оптимальною кількість кроків для такого підходу виявилося 1500н-500б-500н-500б-200н-200б-100н-500б-100н, де н — кроки навчання, б — кроки базової роботи.

Дана конфігурація чергування кроків показала найкращі результати з використання можливих дій гравця та найбільш ефективних відповідей на конкретні стани системи.

3. Навчання певною кількістю кроків лише в нових станах

Цей підхід будувався на припущені, що процес навчання необхідний лише при нових станах системи, а для відомих станів використовувати раніше отримані дані.

Перевага цього підходу передбачалася в тому, що витрати на кроки навчання більші ніж на звичайні робочі кроки, тому якщо використовувати їх, лише коли це необхідно, можна значно знизити витрати часу на одну ітерацію навчання.

Даний підхід виявився неефективним та в результаті ряду тестів від нього довелось відмовитись на користь попередніх підходів, які показали значно вищу ефективність при співставних витратах.

Недоліками цього підходу є:

- За результатами тестування виявилося, що при такому підході важче покрити стани навіть ніж при безперервному навчанні.
- В порівняні з безперервним навчанням цей підхід потребує додаткову систему керування, яка дозволяє перемикати систему між навчанням та звичайним використанням.
- Цей підхід потребує додаткових даних для кожного стану, що при великій кількості станів значно збільшує витрати на навчання.
- Цей підхід потребує більше часу на встановлення ефективної конфігурації роботи системи.
- Значно збільшується час виконання одного кроку через необхідність робити необхідні перевірки та перемикання. Збільшення

часу достатньо відчутне навіть при відносно невеликій кількості кроків навчання в ітерації.

4.3 Аналіз ефективності використання машинного навчання для ІШІ

Для більш детального аналізу доцільності використання машинного навчання було обрано гру жанру стратегія тому, що, як показав аналіз, це один з тих жанрів, де використання машинного навчання є найбільш доцільним. Після реалізації типової стратегічної гри була додана система навчання для системи керування юнітами з використанням методів Q-навчання.

Для оцінки ефективності роботи навченої системи вирішено порівнювати результати запусків в різних умовах. Показником ефективності виступає те, що навчена система може перемагати (або не програвати) в грі, в якій кількість кроків для досягнення перемоги менше ніж при використання випадкових дій.

В якості контрольної вибірки, були обрані підходи «Feed Only», при якому всі накази, що віддаються юнітам направлені на збір ресурсів харчування, та підхід «Random» — на кожному кроці при цьому підході дія обирається випадковим чином.

Крім цього для оцінки ефективності в незалежності від прив’язки до конкретних умов були підготовлені три типи мап:

- «Rich map» — мапа з великою кількістю різних ресурсів та близьким їх розташуванням до початкової точки.
- «Average map» — карта з достатньою кількістю ресурсів, але різні їх типи знаходяться на віддаленні одне від одного.
- «Poor map» — мапа з малою кількістю ресурсів.

На рисунках 35-43 зображенні діаграми результатів від запусків з різними підходами та в різних умовах.

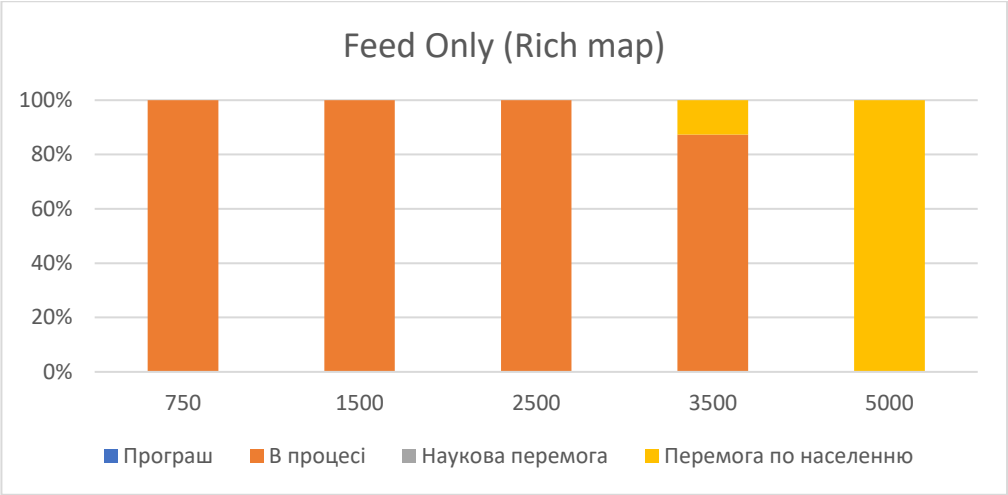

Рис. *35 Діаграма результатів підходу «Feed Only» на карті з генерацією типу «Rich map»*

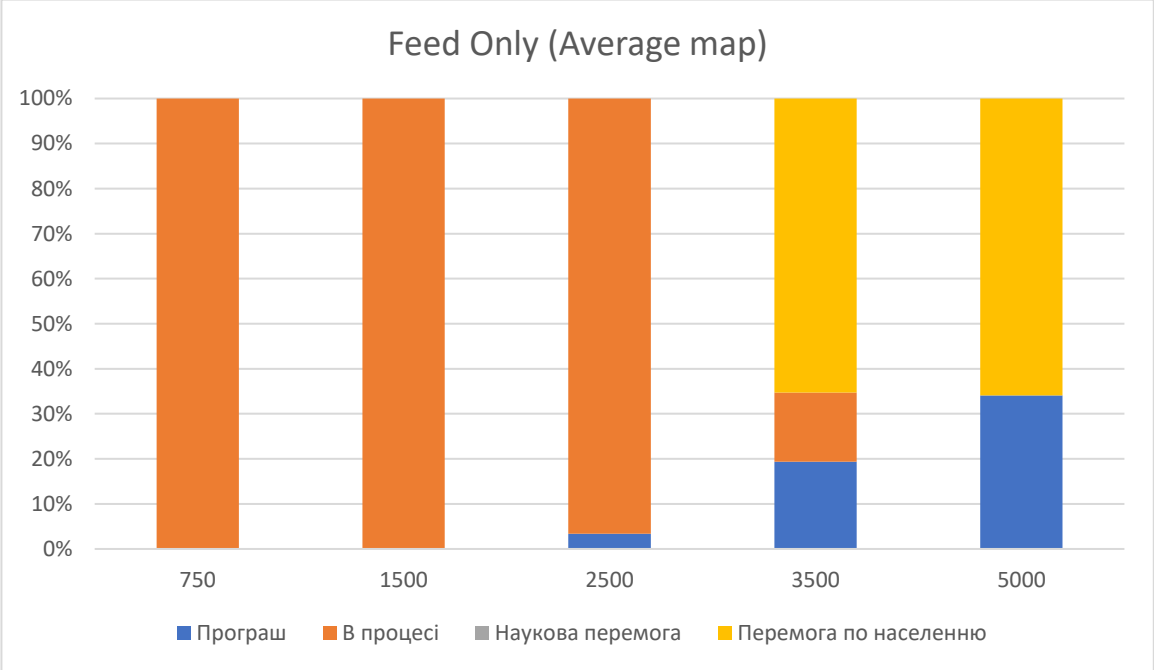

Рис. *36 Діаграма результатів підходу «Feed Only» на карті з генерацією типу «Average map»*

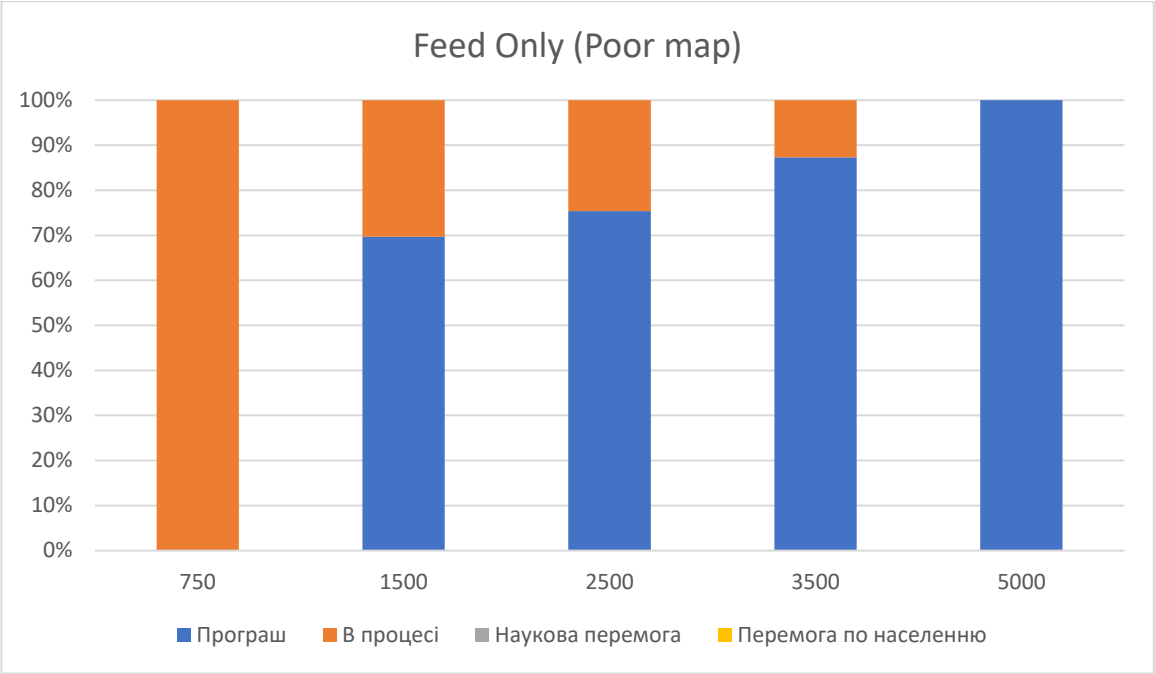

Рис. *37. Діаграма результатів підходу «Feed Only» на карті з генерацією типу «Poor map»*

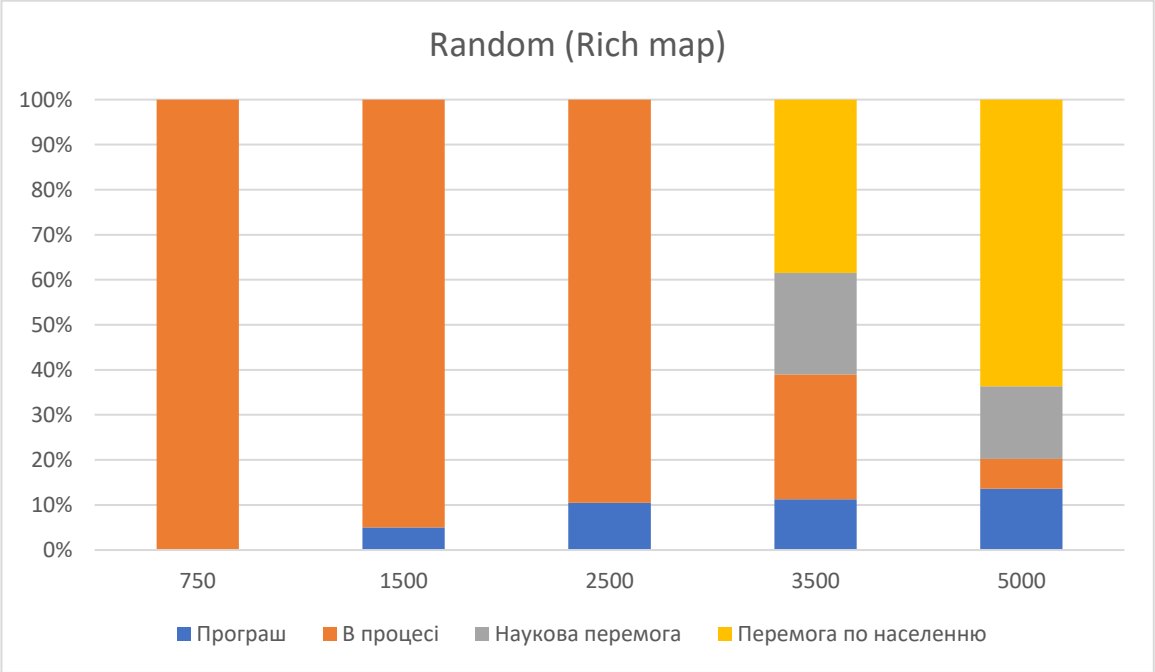

Рис. *38. Діаграма результатів підходу «Random» на карті з генерацією типу «Rich map»*

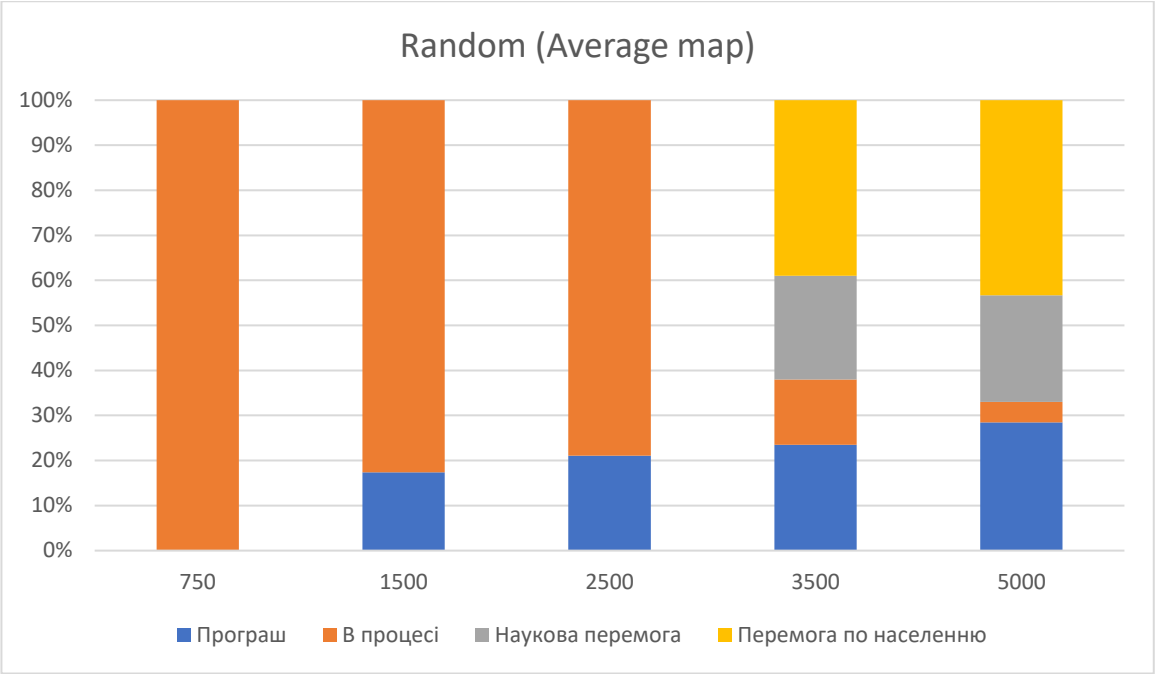

Рис. *39 Діаграма результатів підходу «Random» на карті з генерацією типу «Average map»*

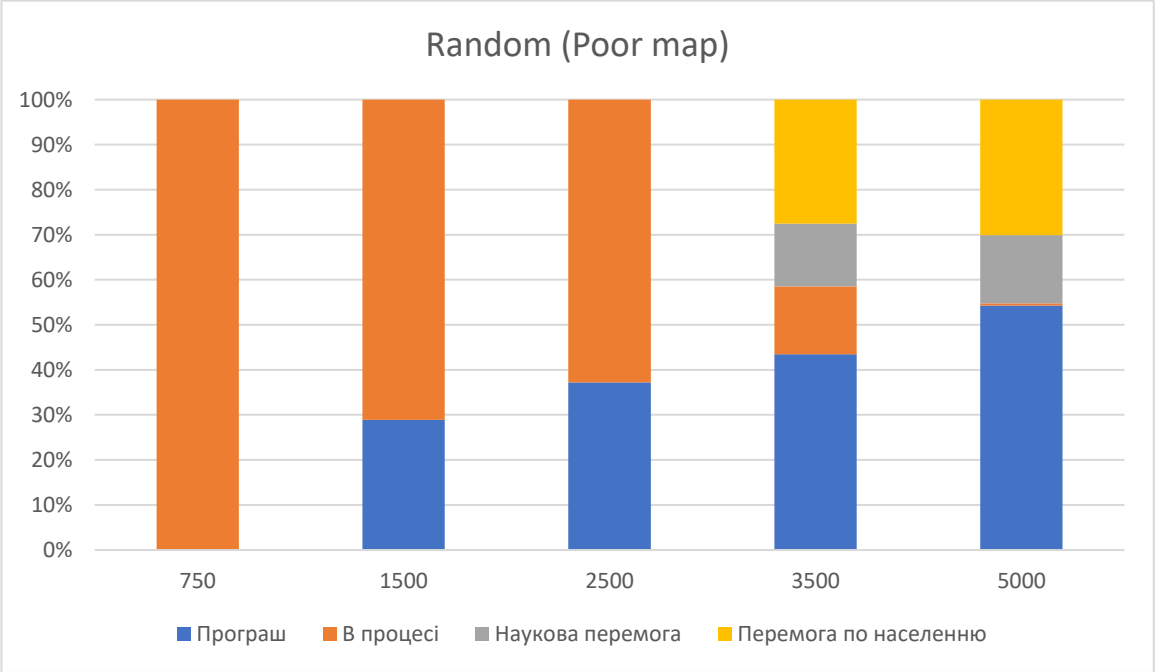

Рис. *40 Діаграма результатів підходу «Random» на карті з генерацією типу «Poor map»*

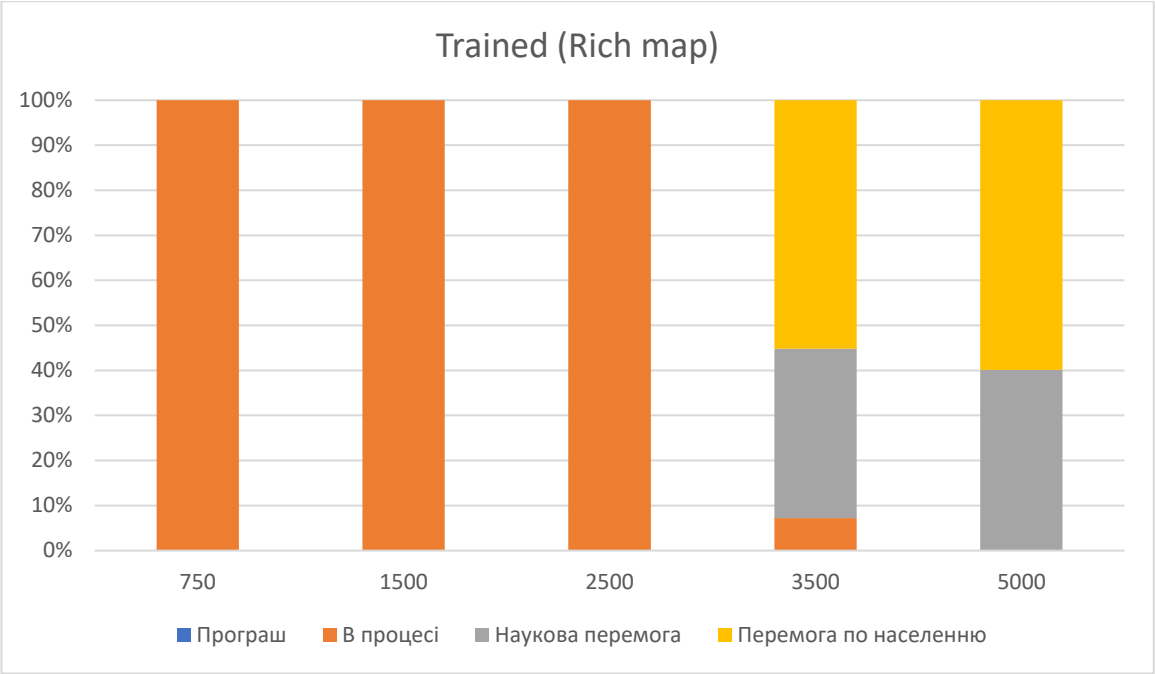

Рис. *41 Діаграма результатів підходу «Trained» на карті з генерацією типу «Rich map»*

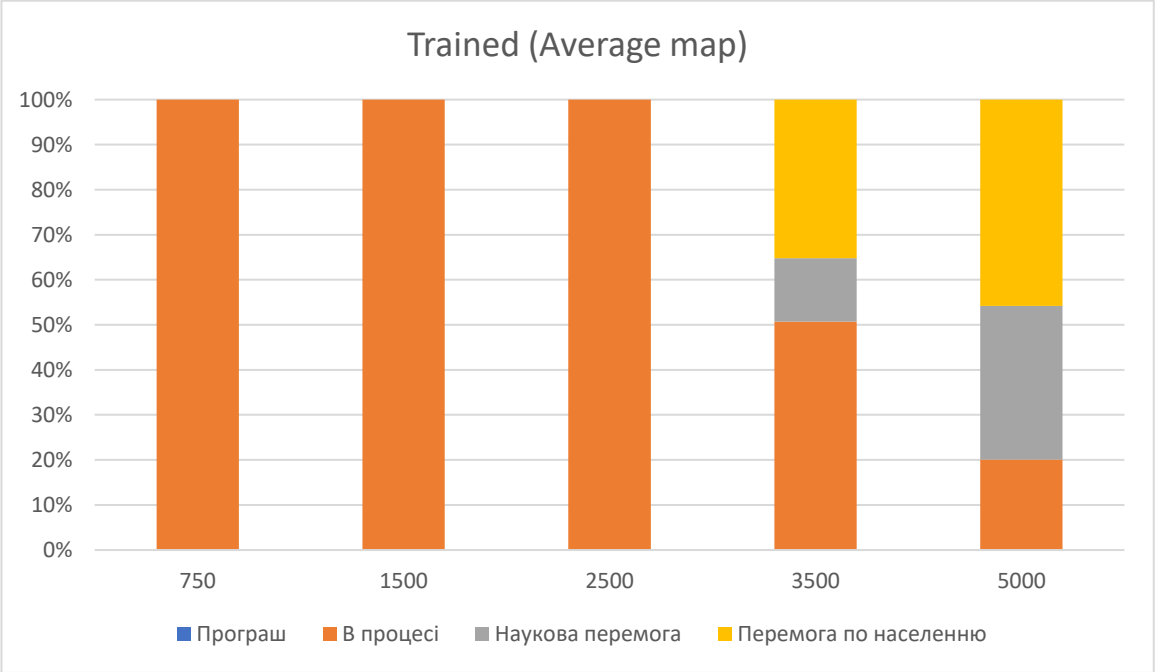

Рис. *42 Діаграма результатів підходу «Trained» на карті з генерацією типу «Average map»*

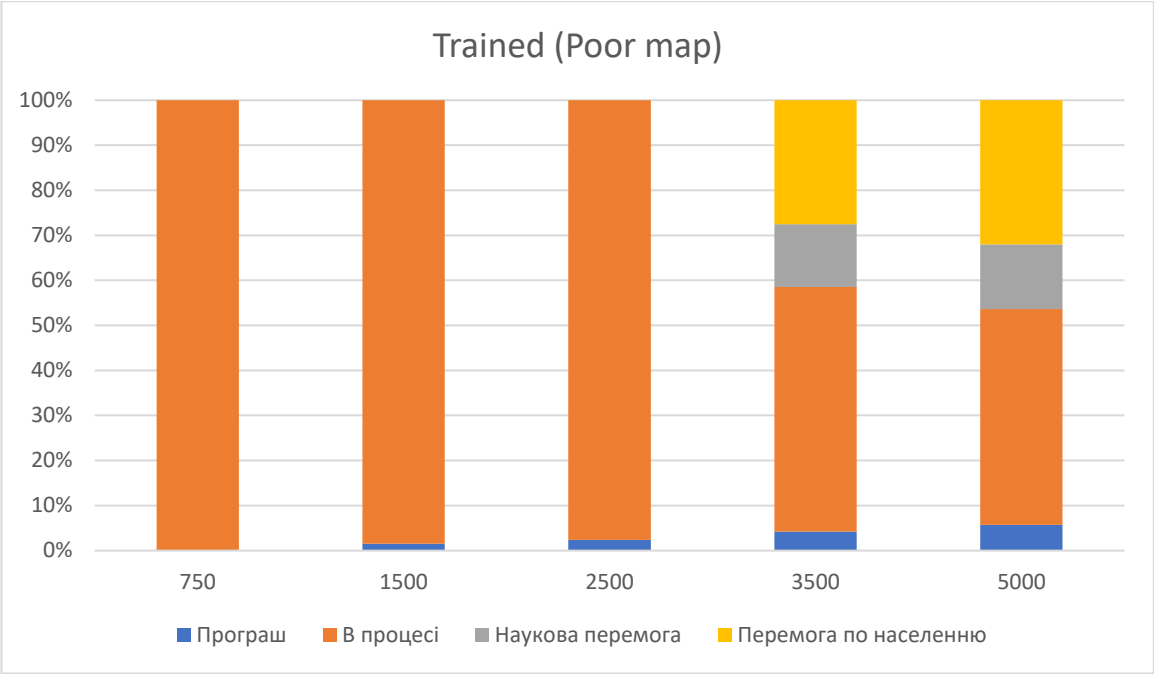

Рис. *43 Діаграма результатів підходу «Trained» на карті з генерацією типу «Poor map»*

Для аналізу ефективності кожним з методів було проведено 1000 запусків на кожному типі мапи. Після 750, 1500, 2500, 3500 та 5000 кроків роботи отримувався стан системи, що визначався одним з можливих станів: програш (випадок коли населення скоротилося до 1 або менше), в процесі (ще не виконався ні одна з умов перемоги, але система ще не програла), наукова перемога або перемога по населенню.

По кількості певних результатів можна оцінити ефективність роботи системи.

На рисунках 44-46 зображені зведені діаграми з кількістю програшів під час використання різних підходів в різних умовах.

Рис. *44 Введена діаграма програшів по підходам на карті з генерацією типу «Rich map»*

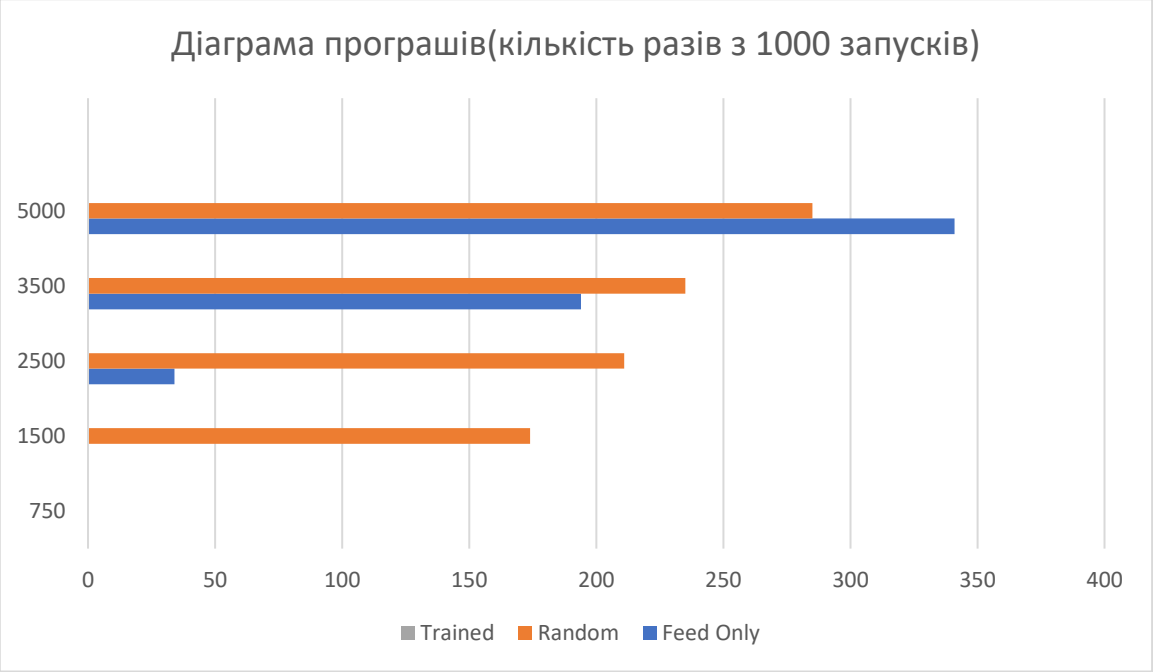

Рис. *45 Введена діаграма програшів по підходам на карті з генерацією типу «Average map»*

Рис. *46 Зведена діаграма програшів по підходам на карті з генерацією типу «Poor map»*

Аналіз діаграм показує, що при конфігураціях «Rich map» та «Average map», навчена система не мала жодного програшного результату (з 1000 запусків), на відміну від інших вибірок, а саме 124 для «Random - Rich Map», 285 «Random - Average Map» та 341 «FeedOnly - Average Map».

При конфігурації «Poor map», кількість програшних результатів навченої системи в десятки разів менше ніж в інших вибірках: 57 — навчена система, 541 — «Random» та 1000 — «FeedOnly».

Опираючись на результати, які можна побачити на діаграмах можна однозначно сказати, що навчена система керування показала значно більшу ефективність ніж контрольні вибірки. При цьому, перевага така, що без заперечень перевищує статистичну похибку та випадкові збурення.

Отже підтвердився висновок що для ігрових застосунків жанру стратегія доцільне використання машинного навчання.

4.4 Висновки з розділу 4

В цьому розділі було проведене дослідження результатів використання систем штучного інтелекту в ігрових застосунках; проведена оцінка доцільності використання машинного навчання в ігрових застосунках. На основі цього можна зробити такі висновки:

1. Система керування, яка була навчена за допомогою підходу чергування кроків навчання та кроків використання системи в одній ітерації навчання, показала більшу ефективність у порівнянні з іншими підходами та у порівнянні з контрольними вибірками.

2. Для покращення ефективності роботи навченої системи існують альтернативні підходи до навчання з підкріпленням, однак вони значно ускладнюють процес їх імплементації в ігровий застосунок та ускладнюють процес взаємодії та аналізу необхідний в рамках їх використання в ігрових застосунках.

3. Розроблений ігровий застосунок з імплементованою в нього системою навчання для системи керування неігровими персонажами придатний для виконання дослідження ефективності ІШІ.

4. Визначені основні проблеми при використанні машинного навчання в ігрових застосунках, основною з яких є часта непрозорість результатів для їх редагування та/або використання.

5. Визначені характеристики, за якими можна визначати доцільність використання машинного навчання в ігрових застосунках, однією з яких є орієнтація в більшій мірі на використання механік ніж на реалізацію певної заздалегідь визначеної історії.

ВИСНОВКИ

1. Досліджена проблема використання машинного навчання в ігрових застосунках та особливості реалізації ІШІ в цілому: розглянуті переваги та недоліки існуючих технологій та систем для її вирішення і сучасні підходи реалізації ІШІ та підходи навчання з підкріпленням.

2. Досліджені засоби для вирішення поставленої проблеми: як ігровий рушій для реалізації ігрового застосунку був обраний рушій Unity; як підхід до навчання системи був обраний метод Q-Learning,

3. Проведено навчання системи керування неігровими персонажами: налаштоване середовище для навчання, налаштовані необхідні параметри для навчання, проведено налаштування конфігурації для навчання та успішно виконане навчання системи керування.

4. Розроблений ігровий застосунок з імплементованою в нього навченою системою керування неігровими персонажами навченою за допомогою навчання з підкріпленням.

5. Досліджені результати навчання та ефективність навченої системи керування. Система. яка навчалася за допомогою підходу чергування кроків навчання та кроків використання системи в одній ітерації навчання, показала вищу ефективність.

6. Дослідження результатів роботи створеного ігрового застосунок доводять, що навчена система показала достатній рівень ефективності, що виражається в пристосовуваності системи керування до різних умов та ефективного виконання поставлених цілей.

7. Визначені основні характеристики, за якими можна оцінити доцільність використання машинного навчання в ігровому застосунку перед його імплементацією: орієнтованість на взаємодію з механіками гри, можливість нелінійної структури гри, необов’язковість виконання певних дій.

8. Поставлена проблема імплементації ІШІ, навченого за допомогою навчання з підкріплення, вирішена та досліджена доцільність використання машинного навчання в ігрових застосунках.

9. Результати навчання можуть використовуватися в майбутньому для аналізу ігрових застосунків для налаштування їх на інтелектуальні дії.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Островецький С. В., магістрант 1 курсу, Безверхий А. І., доцент — науковий керівник. Штучний інтелект в ігрових програмах. І всеукраїнська науково-практична конференція здобувачів вищої освіти, аспірантів та молодих вчених «Актуальні питання сталого науково-технічного та соціально-економічного розвитку регіонів України» : матеріали І всеукраїнської науково-практичної конференції здобувачів вищої освіти, аспірантів та молодих вчених «Актуальні питання сталого науково-технічного та соціально-економічного розвитку регіонів України» 19-21 жовтня 2021 року. Запоріжжя : ЗНУ, 2021. С. 336–337.
2. Островецький С. В., магістрант 1 курсу, Безверхий А. І., доцент — науковий керівник. Штучний інтелект в ігрових програмах з використанням Q-learning та DQN. Молода наука-2022 : зб. наук. праць студентів, аспірантів і молодих вчених. Запоріжжя : ЗНУ, 2022. Т. 5. С. 184–185.
3. Островецький С. В., магістрант 2 курсу, Безверхий А. І., доцент — науковий керівник. Доцільність використання машинного навчання в ігрових застосунках . 18-20 жовтня 2022 року ІІ Всеукраїнська науково-практична конференція за участю молодих науковців «АКТУАЛЬНІ ПИТАННЯ СТАЛОГО НАУКОВО-ТЕХНІЧНОГО ТА СОЦІАЛЬНО-ЕКОНОМІЧНОГО РОЗВИТКУ РЕГІОНІВ УКРАЇНИ» С. 311–312.
4. Островецький С. В., магістрант, Безверхий А. І., доцент — науковий керівник. Штучний інтелект в ігрових застосунках з використанням машинного навчання. Міжнародна науково-практична конференція Інженерного навчально-наукового інституту ім. Ю.М. Потебні Запорізького національного університету «Перспективи сталого розвитку в умовах глобалізації в економічному, управлінському та інженерному аспектах» 3 – 4 листопада 2022 р. С. 253–255.
5. Ігровий Штучний Інтелект - Штучний Інтелект. URL: https://sites.google.com/site/stuchintel/igrovij-stucnij-intelekt. (Дата звернення 07.10.2022).

6. Rabin, S. Game AI Pro 360: Guide to Architecture (1st ed.). CRC Press, 2019. URL: https://doi.org/10.1201/9780429055058 (Дата звернення 22. 11.2022).
7. Розробка штучного інтелекта для ігор – Habr URL: https://habr.com/ru/company/intel/blog/265679/ (дата звернення 21.02.2022).
8. Millington, I., Millington, I., & Funge, J. Artificial Intelligence for Games (2nd ed.). CRC Press, 2009. URL: https://doi.org/10.1201/9781315375229(Дата звернення 22. 11.2022).
9. G. Synnaeve and P. Bessière, Special tactics: A Bayesian approach to tactical decision-making, 2012 IEEE Conference on Computational Intelligence and Games (CIG), 2012, pp. 409-416.
10. Ponsen, Marc, and Pieter Spronck. Improving adaptive game AI with evolutionary learning. Diss. Masters Thesis, Delft University of Technology, 2004.
11. Unity. URL: https://unity.com (Дата звернення 12. 10.2022).
12. TensorFlow. URL: https://www.tensorflow.org (Дата звернення 12. 10.2022).
13. Keras. URL: https://keras.io (Дата звернення 12.10.2022).
14. Horizon Zero Dawn Review. URL: https://www.ign.com/articles/2017/02/20/horizon-zero-dawn-review (Дата звернення 22. 11.2022).
15. The AI of Horizon Zero Dawn. URL: https://www.guerrilla-games.com/read/the-ai-of-horizon-zero-dawn (Дата звернення 12. 10.2022).
16. The Perfect Organism: The AI of Alien: Isolation. URL: https://www.gamedeveloper.com/design/the-perfect-organism-the-ai-of-alien-isolation (Дата звернення 12. 10.2022).
17. Seamen. URL: https://tvtropes.org/pmwiki/pmwiki.php/VideoGame/Seaman (Дата звернення 22. 11.2022).
18. Seaman creator Yoot Saito on the fishy Dreamcast AI that was way ahead of its time. URL: https://www.theverge.com/2019/9/6/20850674/yoot-saito-interview-seaman-sega-dreamcast-ai-20th-anniversary (Дата звернення 12. 10.2022).
19. The Elder Scrolls IV: Oblivion Review. URL: https://www.gamespot.com/reviews/the-elder-scrolls-iv-oblivion-review/1900-6146657/ (Дата звернення 22. 11.2022).

20. The Technology Behind The Elder Scrolls V: Skyrim. URL: https://www.gameinformer.com/games/the_elder_scrolls_v_skyrim/b/xbox360/archive/2011/01/17/the-technology-behind-elder-scrolls-v-skyrim.aspx (Дата звернення 12. 10.2022).

21. Facade review. URL: http://www.spagmag.org/archives/f.html#facade (Дата звернення 22. 11.2022).

22. Mateas, M., Stern, A. (2004). Natural Language Understanding in Façade: Surface-Text Processing. In: , et al. Technologies for Interactive Digital Storytelling and Entertainment. TIDSE 2004. Lecture Notes in Computer Science, vol 3105. Springer, Berlin, Heidelberg. URL: https://doi.org/10.1007/978-3-540-27797-2_2 (дата звернення 16.09.2022).

23. M. Mateas and A. Stern, A behavior language for story-based believable agents, IEEE Intelligent Systems, vol. 17, no. 4, pp. 39-47, July-Aug. 2002, URL: https://doi.org/10.1109/MIS.2002.1024751 (дата звернення 19.09.2022).

24. Michael Mateas & Andrew Stern, An Experiment in Building a Fully-Realized Interactive Drama, Game Developers Conference, San Jose. 2003, URL: https://faculty.cc.gatech.edu/~isbell/reading/papers/MateasSternGDC03.pdf (дата звернення 25.09.2022).

25. Mateas, Michael & Stern, Andrew, Structuring Content in the Façade Interactive Drama Architecture, Proceedings of the First Artificial Intelligence and Interactive Digital Entertainment Conference pp. 93-98. 2005, URL: https://www.aaai.org/Papers/AIIDE/2005/AIIDE05-016.pdf (дата звернення 21.09.2022).

26. LAMPLE, G.; CHAPLOT, D. Playing FPS Games with Deep Reinforcement Learning. AAAI Conference on Artificial Intelligence, North America, feb. 2017, URL:https://www.aaai.org/ocs/index.php/AAAI/AAAI17/paper/view/14456/14385 (дата звернення 12.05.2022).

27. Генетичні алгоритми. Ключові поняття і методи реалізації. URL: http://www.znannya.org/?view=ga_general (Дата звернення 12.09.2022).

28. Watkins, C.J.C.H., Dayan, P. Q-learning. Mach Learn 8, P. 279–292 (1992).

29. Zell Andreas. Simulation Neuronaler Netze [Simulation of Neural Networks] German, Addison-Wesley, Chapter 5, 1994.

30. Playing Atari with Deep Reinforcement Learning URL: https://www.cs.toronto.edu/~vmnih/docs/dqn.pdf (дата звернення 10.04.2022).

Декларація
академічної доброчесності
здобувача ступеня вищої освіти ЗНУ

Я, Островецький Сергій Володимирович , студент 2 курсу магістратури, форми навчання денної, Інженерного навчально-наукового інституту ім. Ю.М. Потебні, спеціальність 121 Інженерія програмного забезпечення, адреса електронної пошти ipz17bd-24@stu.zsea.edu.ua, — підтверджую, що написана мною кваліфікаційна робота на тему «**Штучний інтелект в ігрових програмах з використанням машинного навчання**» відповідає вимогам академічної доброчесності та не містить порушень, що визначені у ст.42 Закону України «Про освіту», зі змістом яких ознайомлений.

- заявляю, що надана мною для перевірки електронна версія роботи є ідентичною її друкованій версії;

- згоден на перевірку моєї роботи на відповідність критеріям академічної доброчесності у будь-який спосіб, у тому числі за допомогою інтернет-системи, а також на архівування моєї роботи в базі даних цієї системи.

Дата 30.11.2022 _____ Островецький Сергій Володимирович

(студент)

Дата 30.11.2022 _____ Безверхий Анатолій Ігорович

(науковий керівник)