Міністерство освіти і науки України
Харківський національний університет радіоелектроніки
Факультет _____Комп'ютерних наук_____
Кафедра_____Програмної інженерії_____

АТЕСТАЦІЙНА РОБОТА
Пояснювальна записка
рівень вищої освіти – другий (магістерський)

Дослідження методів реалізації штучного інтелекту в іграх

Виконав: студент 2 курсу, групи ІПЗм-18-2
_____Алєксєєв Я.В._____

спеціальності 121 – Інженерія програмного забезпечення

Освітньо-наукової програми
Інженерія програмного забезпечення

Керівник _____доц. Вечур О.В._____

Допускається до захисту
Зав. кафедри, проф. _____ З.В.Дудар
(підпис)

2020 р.

Харківський національний університет радіоелектроніки

Факультет Комп‘ютерних наук

Кафедра Програмної інженерії

Рівень вищої освіти - другий (магістерський)

Спеціальність 121 – Інженерія програмного забезпечення

Тип програми освітньо-наукова програма

Освітня програма Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ:

Зав. кафедри _____

(підпис)

«___»_____ 2020 р.

ЗАВДАННЯ

НА АТЕСТАЦІЙНУ РОБОТУ

Студентові Алєксєєву Ярославу Владиславовичу

1. Тема роботи (проекту) «Дослідження методів реалізації штучного інтелекту в іграх»

затверджена наказом по університету від «____» _____ 2020р. № _____

2. Термін подання студентом роботи (проекту): _____ 2020р.

3. Вихідні дані до роботи (проекту): пояснювальна записка.

4. Перелік питань, що потрібно опрацювати в роботі: мета роботи, аналіз проблемної галузі і постановка задачі, аналітичний огляд методів реалізації штучного інтелекту, аналіз нейронних мереж штучного інтелекту, експериментальне дослідження ефективності роботи нейронних мереж штучного інтелекту.

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Терміни виконання етапів роботи	Примітка*
1.	Аналіз предметної галузі	03 квітня 2020 р.	
2.	Огляд існуючих методів	10 квітня 2020 р.	
3.	Дослідження методів класифікації томографічних зображень	17 квітня 2020 р.	
4.	Підготовка пояснювальної записки	20 листопада 2020 р.	
5.	Спецчастина	23 листопада 2020 р.	
6.	Підготовка презентації та доповіді	25 листопада 2020 р.	
7.	Попередній захист	15 грудня 2020 р.	
8.	Нормоконтроль, рецензування	15 грудня 2020 р.	
9.	Занесення диплома в електронний архів	15 грудня 2020 р.	
10.	Допуск до захисту у зав. кафедри	15 грудня 2020 р.	

Дата видачі завдання: «____»_____ 2020 р.

Студент _____ Алєксєєв Я.В.

(підпис)

Керівник роботи _____ доц. Вечур О.В.

РЕФЕРАТ / ABSTRACT

Пояснювальна записка до атестаційної магістерської роботи містить 91 сторінок, 27 рисунків, 7 таблиць,18 джерел.

Об’єкт дослідження – методи реалізації штучного інтелекту в іграх.

Метою роботи є дослідження штучного інтелекту у різних іграх.

У результаті роботи здійснено аналіз поведінки штучного інтелекту у різних іграх реалізованих на різних ігрових двигунах.

ШТУЧНИЙ ІНТЕЛЕКТ, КОМП’ЮТЕРНІ ІГРИ, РЕАЛІЗАЦІЯ ШТУЧНОГО ІНТЕЛЕКТУ, ШТУЧНИЙ ІНТЕЛЕКТ В ІГРОВИХ ДВИГУНАХ, ГЕЙМДЕВ, АЛГОРИТМИ ШТУЧНОГО ІНТЕЛЕКТУ

The object of research - methods of implementing artificial intelligence in games. The aim of the work is to study the study of artificial intelligence in various games.

As a result, the analysis of the behavior of artificial intelligence in different games implemented on different game engines.

ARTIFICIAL INTELLIGENCE, COMPUTER GAMES, IMPLEMENTATION OF ARTIFICIAL INTELLIGENCE, ARTIFICIAL INTELLIGENCE IN GAME ENGINES, GAMEDEV, SHT ALGORITHM

ЗМІСТ

ВСТУП

Найперші комп'ютерні ігри, створені у 1960-1970 роках, ігри: Спейсвар, Понг і Гоча, їх побудвали на логіці дискретної структури які орієнтувалися на змаганні 2-х противників не використовуючи штучний інтелект.

Такі ігри як Медден Футбол, Ерл Вівер Бейсбол та Тоні Ла Раша Бейсбол були на своєму штучному інтелекті який копіює комп'ютерний розвиток чи управління обрану тобою популярну особу. Розробники ігр Медден, Вівер та Ла Раша пройшли багато етапів розробки, для того щоб домогтися як можливо більшої коректності. Ігри які були пізніше у спортивній тематиці давали гравцям можливість «модернізувати» штучний інтелект для того щоб визначати організаторську або тренувальну стратегію.

Наближений алгоритм у грі штучний інтелект використовується у широкому асортименті у різних етапах гри. Найочевидніше використання штучного інтелекту у іграх може бути виражено в управлінні неігровими персонажами, але і використання кодування персонажу теж дуже поширено. Метод пошуку шляху теж дуже широко поширений в іграх і штучний інтелект, - найвидніше це показано у РТС(стратегії реального часу. Метод пошуку шляху це метод у визначенні, як неігровий персонаж переходить із з одного місця на карті у інше: треба врахувати ту чи іншу перешкоду, побудову ландшафту, та туман війни чи інше. Штучний інтелект у іграх пов'язаний з динамікою ігрового балансування.

1 АНАЛІТИЧНИЙ ОГЛЯД ШТУЧНОГО ІНТЕЛЕКТУ І ЙОГО ЗАРОДЖЕННЯ

1.1 Аналіз предметної області

Концепцію непередбачуваного штучного інтелекту дослідили. У комп'ютерних іграх: Клечурс, Чорне та біле і Нінтендогс, та у відомій всім грі тамагочі. Індивідуальність цих ігор у тому, що їх поведінка залежить від того як поведе себе гравець. Але великий мінус токої взаємодії у тому що поведінка штучного інтелекту залежить від тих чи інших прийнятих рішень гравця, вибірково з усіх рандомних змінних.

Розробники ігор інколи створюють такий досвід як не справжній штучний інтелект, це є ілюзією того як веде себе комп'ютер і гравець думає, що це розумний штучний інтелект. Але це не завжди є не дуже рентабельним рішенням для любого штучного інтелекту, тому що він включає методики, які широко використовуються поза движка ігрового штучного інтелекту. Наприклад, знання про потенційне майбутнє зіткнення це дуже важлива інформація, яка добавляється в логічну частину, це допомагає створенню штучного інтелекту, який буде таким розумним що зможе обходити всі об'єкти і не давати йому врізатися в об'єкт . Насамперед для любого ігрового двигуна дуже важлива ця деталь, з якою і передбачена функція обходу об'єктів. Такі ж результати випробувального напрямку погляду бота є дуже важливою стадією де штучний інтелект визначає свій шлях; зі данні є дуже важливими у рендерензі штучного інтелекту в ігровому двигуні. Самим важливим етапом вважається кодування, яке є найзручнішим інструментарієм для усіх етапів геймдеву, але це часто асоціювати з поведінкою штучного інтелекту.

Ідеалісти кажуть, те що ігровий комп'ютерний інтелект в виразі штучний інтелект є, неправдоподібним так як штучний інтелект описує не інтелект і використовує напрямки академічну науку штучний інтелект. Насамперед тому, що «справжній» штучний інтелект відноситься до того що потрібен самонавчатися і самому приймати рішення , він базується на введені даних довільно, і насамперед,

що штучний інтелект сам повинен думати що йому робити, ігровий штучний інтелект в першу чергу повинен створювати зручність для часоутворення будь-якого гравця, та щоби гравець розумів, що він грає не сам із собою, а насамперед утворювався геймплей і чудові відчуття від всієї гри.

Перебільшення того що академічний штучний інтелект ігровими розробниками і багатий інтерес наукової спілки до комп'ютерних ігор викликає багато питань, на скільки штучний інтелект має багато нестиковок з реальним штучним інтелектом. Однак є велика різниця між штучним інтелектом який застосовується у різних напрямках и може бути як підвид штучного інтелекту, що ігровий штучний інтелект все ще може бути розглянутий як окрема під-галузь штучного інтелекту. Зокрема, здатність «законним» чином вирішити деякі проблеми штучний інтелект є ілюзією. На прикладі того як штучний інтелект який не бачить гравця і з тим застосовує пошук його на карті це є великою проблемою, цим штучний інтелект повинен шукати гравця по заданим шляхам. Така ілюзія ще вважається обманом, і буває і таке що є такі ситуації коли штучний інтелект викликає багато помилок.

1.2 Опис проведених досліджень

Після 1990-го року з’явилось дуже багато інших жанрів ігор і завдяки цьому розробники ігор почали використовувати кінцеві автомати. Стратегії реального часу принесли багато інновацій де штучний інтелект отримував нові рішення він не мав повну інформацію де знаходиться гравець, шукав найближчий і зручніший шлях, розвивався у економічному сенсі. В перших стратегіях реального часу було багато проблем з такими рішеннями. Приклади таких проблем і ігор: Херзог Звеі де штучний інтелект не завжди міг знайти шлях до гравця, в Дюні комп’ютерний штучний інтелект, не правильно працював, бо кінцевий автомат не функціонував

так як і потрібен був з 3 різними станами для управління персонажами, після цього вже стратегії реального часу еволюціонували, як наведено на рисунку 1.1

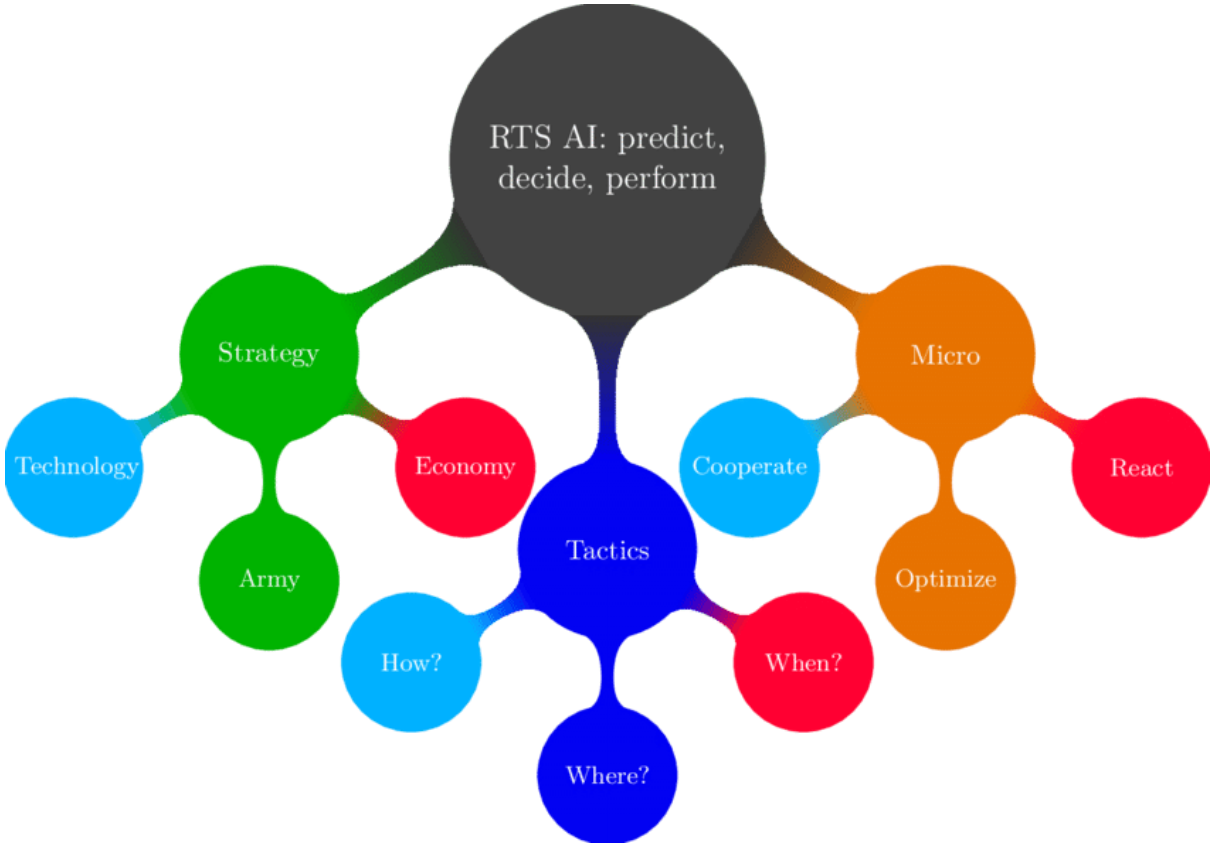

Рисунок 1.1 – Штучний інтелект у стратегіях

Після цього ігри використовували не стандартні методи штучного інтелекту, використовуючи нейронні мережі 1996 рік, БатлКрісер 3000АД до поведінки яку гравець не може передбачити і відштовхуючись від дій супротивника, як Кречурс І Чорне і Біле.

ГолденЕй 007 1997-го року це самий перший шутер від першого обличчя, в якому штучний інтелект реагував на те як поводиться гравець, а ще вин використовував об’єкти для укриття і перекати щоби гравець не влучив у противника. Штучний інтелект ще був здатний кидати гранату у потрібну хвилину. Після цього розробники цієї компанії зробили штучний інтелект ще краще і застосували його у грі Перфект Дарк. Але була дуже велика проблема цієї гри, це

те що штучний інтелект завжди знав де точно знаходиться гравець і від цього відштовхувалися.

Halo combat evolved 2001-го року мав інноваційний штучний інтелект, супротивники могли обходити гравця з усіх боків, і використовувати гранати, користуватися транспортними засобами і мав дуже багато різних станів. Ще цей штучний інтелект міг скинути, та забрати транспорт гравця. Ще цей штучний інтелект міг визначати де небезпека і покидати зону загрози. Були ще багато різновидів супротивників і це передбачувало, якщо вбити ворожого капітана, то ворожа піхота, якщо іх було мало на полі, то відступали, бігли у різні боки і перегрупувалися

Фар край 2004-го року шутер від першого був дуже розвинутим для свого часу, бо він так же міг обійти гравця з різних боків, але ще був запрограмованим так, що використовував реальні тактики військовослужбовців. Він постійно змінювався, але насамперед не знав де знаходиться гравець, просто, він діяв від того що якщо бачить гравця від того і відштовхувався.

Багатий вклад у розвиток штучного інтелекту приніс шутер від першого обличчя ФІР, його розробила компанія Моноліт Продакшн в 2005 році. На свого часу він містив розвинений штучний інтелект, який був зустрінутий дуже позитивно всіма ігровими рецензентами і пресою. Насамперед, бо бої відбуваються в невеличких приміщеннях и штучний інтелект не «стоїть на місці» він використовує всі ігрові об’єкти і кооперується і правильно діє в тій чи іншій ситуації де є взаємодія с противником гравцем. Правильно користуються гранатами для того щоб випроводити гравця з укриття.

The Elder Scrolls IV: Oblivion Ролева гра користувалася дуже хорошим інтелектом для ігор цього жанру. Заявлено, що штучний інтелект мав своє життя, проте іноді можливо спостерігати, як неігровий персонаж стоїть на одному місці кілька годин поспіль (бармен за стійкою вночі без відвідувачів). Так, вони їдять, сплять і виконують свої щоденні обов'язки. Основна перевага у тому що, штучний інтелект підлаштовувався під все що загально відбувалося у грі.

Комп’ютерна гра від українських розробників СТАЛКЕР.: Тінь Чорнобиля, 2007-го року, мала унікальний штучний інтелект з назвою «A-Life». Розроблялася с 2002-го року, але було в кінцевій версії багато вирізано «A-Life». «A-Life» допрацювали у Сталкер.: Чисте небо 2008-го року.

Шутер від першого обличчя для декількох гравців Лефт Фо Дед 2008-го року використовував штучний інтелект який називався «Режисер». «Режисер» . Велика перевага цього способу у тому що, штучний інтелект процедурно генерується виходячи з усіх ігр різних гравців, аналізуючи це і при тому дає можливість пройти гру, але кожний раз як новий ігровий досвід.

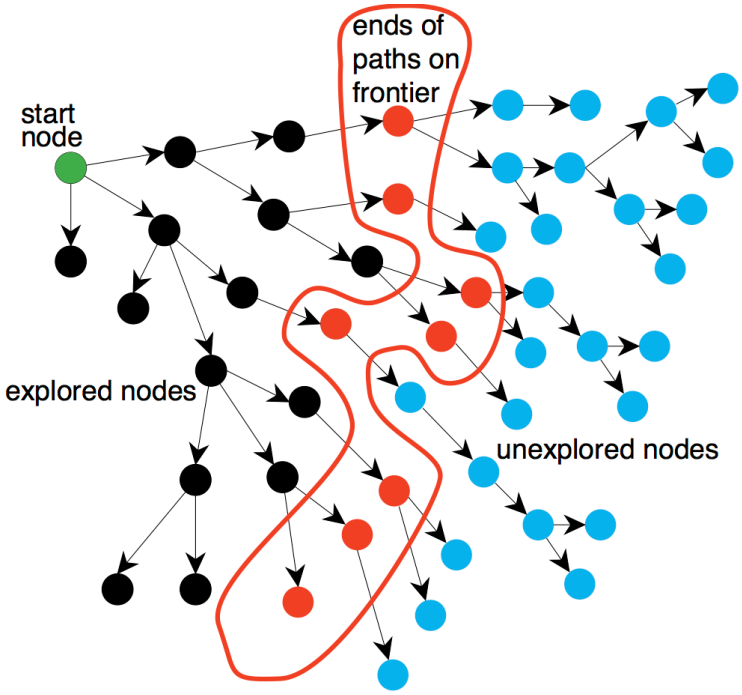

Рисунок 1.2 – Приклад графа шляху штучного інтелекту

Бій декількох NPC між собою - Бій декількох NPC-персонажів між собою, рисунок 1.2 це термін, який популяризували у шутерах від першого обличчя, такими, як Дум, випущеним в 90-х роках. Це можливо охарактеризувати як бій одного штучного інтелекту з іншим у грію, можливо випадково, коли вони не є з самого початку ворогами. Гравці досягали цього коли, ставали в одну лінію з противниками і той стріляв у гравця і попадав в іншого штучного інтелекту, ти

самим провокуючи вогонь на себе після цього вже обидва комп’ютери не будуть вважати себе союзниками, а навпаки ворогами.

Штучний інтелект в іграх до сих пір розвивається щоби досягнути тієї точки, коли гравець не зможе відрізнити його від інших гравців і це для нього буде найкращим ігровим досвідом.

1.3 Аналіз результатів досліджень

На даний час у реальності первинна задача розробки штучного інтелекту, є ціль при якому штучний інтелект який зможе сам навчатися тому чи іншому, бути соціальною істотою, та мати емоції які має і сама людина, іншими словами добитися того щоб він став повністю живим як і людина, чи тварина.

В іграх штучний інтелект це дещо інше він не схожий з тими що розроблюються людьми у реальності і до нього зовсім інші вимоги, в першу чергу щоби, гравець відчув насолоду від самої ігри. Штучному Інтелекту в іграх не потрібні відчуття, та самосвідомість. Йому потрібне тільки самонавчання яке прив’язано до самої гри і ігрового процесу. В першу чергу штучний інтелект в іграх потрібен для того щоби ставити перешкоди для гравця, але тільки ті які не будуть заважати самому ігровому процесу, а саме переконливо і правдоподібно.

Основним принципом, штучного інтелекту це є поведінка у тій чи іншій ситуації. При якому штучний інтелект зможе впливати на об’єкти. При цьому такий вплив може бути організовано у вигляді «мовлення штучного інтелекту» або «звернень об'єктів».

У системах з «мовленням штучний інтелект» система штучного інтелекту зазвичай ізольована у вигляді окремого елемента ігрового типу. Такий тип зазвичай становиться окремою формою, або декількома, у якій штучний інтелект обчислює найкраще рішення для заданих параметрів гри. Коли штучний інтелект

відтаскується від тих чи інших подій у грі. Таке дуже підходить до стратегій реального часу, в ній штучний інтелект аналізує всі події у грі

Системи зі зверненням до об’єктів краще підходить для простих ігор. В таких іграх об'єкти звертаються до системи штучного інтелекту кожен раз, коли об'єкт «думає» або оновлює себе. Такий підхід відмінно використовується в іграх з великою кількістю об’єктів, де штучному інтелекту не потрібно обчислювати багато інформації, наприклад в шутерах. Для цієї системи потрібно набагато більше планування в архітектурі, але вона має свої переваги.

Для того щоби штучний інтелект мав можливість себе проявити, для нього треба різна кількість обчислень, у простих іграх це звичайне положення штучного інтелекту до гравця. У складних іграх, в залежності від типу гри, це багато різних вимог: ситуація, стан гравця, положення до цього гравця, найближчий маршрут до гравця, укриття і багато іншого.

Але ті хто проектують всі ці обчислення повинні і звертати увагу на багато різних способів досягнення. Наприклад, якщо дизайнери поставили на площі ті чи інші укриття, то найближчий шлях до них, так найвірніший спосіб досягнення, та зручність укриття, то положення гравця, Щось вичислюватися повинно на льоту, а щось заздалегідь, бо є та чи інша ситуація спровокована гравцем, як загроза для штучного інтелекту.

Саме простіше для штучного інтелекту яка була розроблена , це звичайна система правил. Така система надалі коштує від справжнього штучного інтелекту. Набір заздалегідь заданих алгоритмів визначає поведінку ігрових об'єктів. З урахуванням різноманітності дій кінцевий результат може бути неявній поведінкової системою, хоча така система насправді зовсім не буде «інтелектуальної».

Класичним іграми платформи, де використовується така система, є Пак-Мен [1]. Гравця переслідують чотири привиди. Кожне привид діє, підкоряючись простому набору правил. Перший привид завжди повертає вліво, другий завжди повертає вправо, третій повертає в довільному напрямку, а останній завжди повертає в бік гравця. Але якщо би на екрані привиди з'являлися не всі, а по одному,

то їх поведінка була б дуже легко відрізнити і гравець зміг би без праці від них рятуватися. Але оскільки з'являється відразу група з чотирьох привидів, їх руху здаються складним і скоординованим вистежуванням гравця.

Наочне представлення набору правил, які керують привидами в грі Пак-Мен, де стрілки представляють прийняті «рішення», рисунок 1.3.

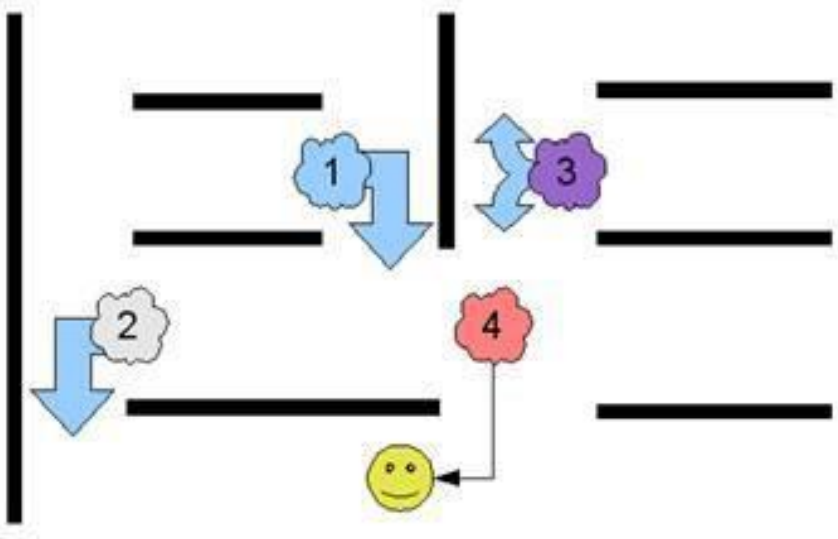

Рисунок 1.3 – Приклад Штучного Інтелекту Pac-Man

Насправді ж тільки останнім з чотирьох привидів враховує розташування гравця.

Тепер розуміємо, що правила можуть бути і не точно задані. Вони можуть ґрунтуватися на сприймається стані (як у останнього привиди) тут використовують такі змінні як положення до гравця, рисунок 1.4, чи рівень сміливості, агресія, швидкість прийняття рішень, це дає можливість отримати деякі правила на яких і ґрунтується штучний інтелект і його поведінка до гравця.

У складних іграх основною перевагою є задані правила ігровими розробниками. Наприклад тактичні ігри це правила які зосередженні на тій чи іншій тактиці. У стратегіях реального часу цими правилами являються реакції на ті чи інші явища у грі. Це є фундамент штучного інтелекту.

Кінцевий автомат це є спосіб моделювання і реалізації об'єкта, що володіє різними станами протягом свого життя.

Схема станів у кінцевого автомату, представляють можливі зміни стану

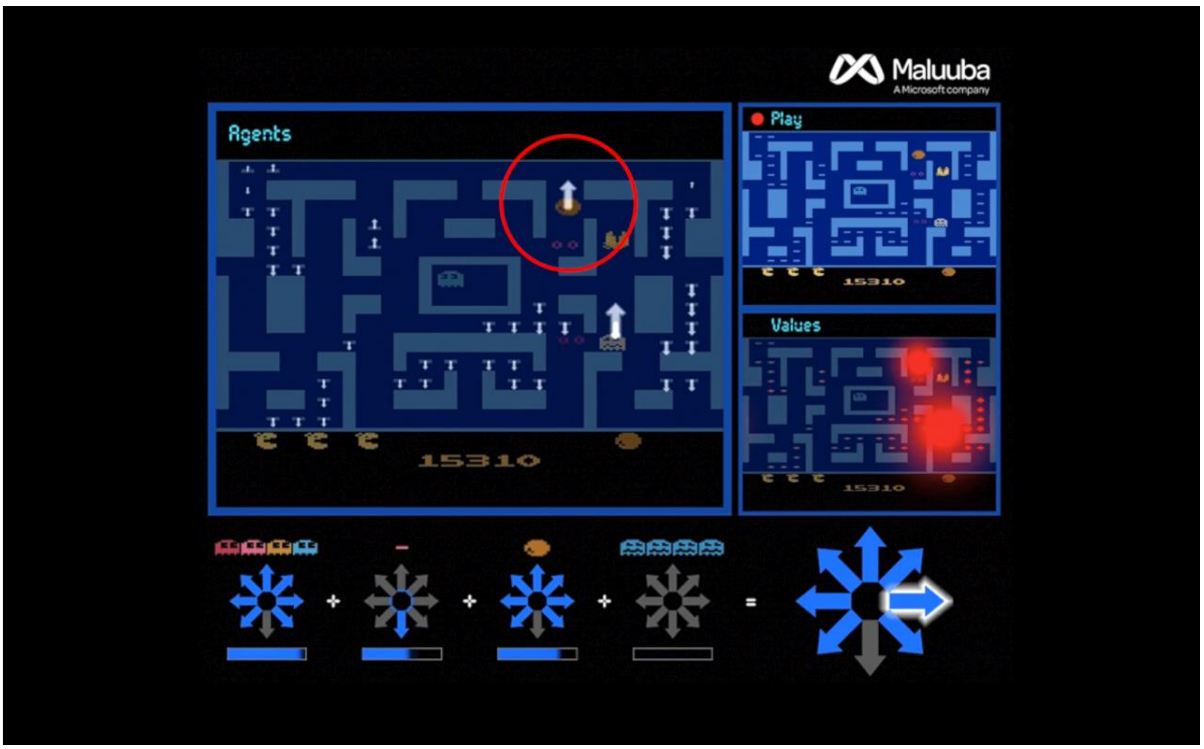

Рисунок 1.4 – Приклад поведінки у Пак Мен

Всі стани можуть представляти різні умови фізичного характеру в яких знаходиться об'єкт у даний час, або емоції які використовує об’єкт чи його стан. Емоційний стан не має подібності до емоцій штучного інтелекту, ці емоції відносяться до правил які насамперед задані поведінковим моделям штучного інтелекту, вписується в контекст до тієї чи іншої гри яка повинна вписуватися у основу частину.

Існує як мінімум два простих способи реалізації кінцевого автомата з системою об'єктів. Перший спосіб: це коли кожній стан є змінною яку можливо перевірити за допомогою інструкцій. Інший спосіб: використовувати функції у мовах програмування C,C++ та інші мови програмування які використовуються у ігровій розробці.

Якщо у грі потрібна більша динамічність, то для неї потрібен штучний інтелект який буде пристосовуватися до різної поведінки і різних станів, а точніше адаптуватися, приклад адаптування можемо побачити на рисунку 1.5.

Рисунок 1.5 – Кінцевий автомат

Штучний інтелект який може адаптуватися вперше використовується у стратегіях реального часу, тактичних іграх, різних бойових іграх, та інших.

Розглянемо цей штучний інтелект. Інтелектуальний агент потрібен сприйняти різне середовище, а саме місцезнаходження об’єкта чи гравця при його пересуванні чи у спокійному стані. Для нас потрібно те ж саме, але декілька змінене. До того ж, по відношенню до ігрового світу можливо використовувати так звані шахрайські методи, але це потрібно робити так, щоб це працювало і все правильно робило і штучний інтелект через це не зламався.

Ця ситуація складається з двох проблем, по перше це розпізнавання самого укриття на заданому ландшафті чи геометрії; і друга проблема, це правильне розпізнавання того самого укриття на площині чи ландшафті.

Визначення штучним інтелектом, чи видно буде його за цим укриттям, чи воно буде підходити для нього по розмірам і зможе він помістится у нього. Це можливо вислідити за геометрією укриття та моделлю самого штучного інтелекту.

1.4 Постановка задачі

На основі виконаного аналізу предметної галузі метою роботи стає розробка свого штучного інтелекту який буде самонавчатися та аналізувати поведінку гравців для того щоб поліпшувати самого себе, а не за допомогою програміста.

Поставлена задача може бути розділена на підзадачі:

- провести аналіз проблем, що існують у сучасному штучному інтелекті;
- дослідити існуючий само-розвинений штучний інтелект;
- розробити свій штучний інтелект;
- виявити більшу ефективність штучного інтелекту;
- сформулювати рекомендації щодо розробки само-розвиненого штучного інтелекту.

Це дозволить організувати найбільшу взаємодію гравця з комп’ютерною грою, бо основна задача штучного інтелекту, це щоб гравець відчував, що проти нього грає не бот, а справжня людина

2 АНАЛІЗ ІСНУЮЧИХ ПРОСУНУТИХ ШТУЧНИХ ІНТЕЛЕКТІВ

2.1 AlphaStar

с В останні роки StarCraft вважається однією з найбільш багатогранних і найскладніших стратегій у режимі реального часу та однією з найпопулярніших електронних спортів на сцені в історії, і тепер StarCraft також став головним викликом для досліджень штучного інтелекту.

AlphaStar - перша система штучного інтелекту, здатна перемогти найкращих професійних гравців. У серії матчів, які відбулися 19 грудня, AlphaStar здобув перемогу над Grzegorz Komincz (MaNa) від команди. Один із найсильніших гравців світу, з рахунком 5: 0. До цього вдалий демонстраційний матч також був зіграний проти його товариша по команді Даріо Вюнша (TLO). Матчі проходили за всіма професійними правилами на спеціальній картці турніру та без будь-яких обмежень.

Незважаючи на значний успіх у таких іграх, як Atari, Mario, Quake III Arena та Dota 2, методи безуспішно боролися зі складністю StarCraft. Найкращих результатів було досягнуто шляхом вручну побудови основних елементів системи, накладення різних обмежень на правила гри, надання системі надлюдських здібностей або гри на спрощених картах. Але навіть ці нюанси унеможливили наближення до рівня професійних гравців. Зважаючи на це, AlphaStar, рисунок 2.1 грає в повноцінну гру, використовуючи глибокі нейронні мережі, які навчаються на основі необроблених даних про ігри, використовуючи методи навчання з викладачем та підкріплення.

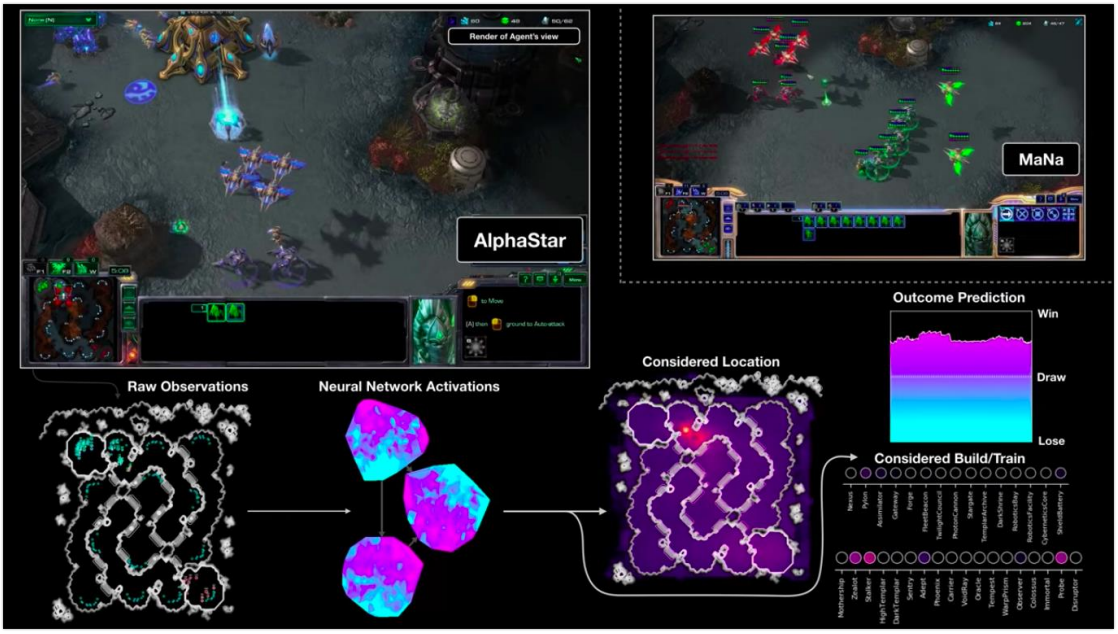

Рисунок 2.1 Побудова штучного інтелекту AplphaStar

Головна проблема StarCraft II - це вигаданий фантазійний всесвіт з багатим, багаторівневим ігровим процесом. Поряд з оригінальним виданням, це найбільша та найуспішніша гра всіх часів, яка веде боротьбу на турнірах більше 20 років.

Існує багато способів гри, але найпоширенішими в електронному спорті є турніри один на один, що складаються з 5 матчів. Для початку гравець повинен вибрати один з трьох розпусників, протосів або терен, кожен з яких має свої особливості та можливості. Тому професійні гравці найчастіше спеціалізуються на одній гонці. Кожен гравець починає з декількох робочих підрозділів, які витягують ресурси для будівництва будівель, інших підрозділів або для розвитку технології. Це дозволяє гравцеві захоплювати інші ресурси, будувати все більш складні бази та розвивати нові здібності перехитрити опонента[3]. Щоб виграти, гравець повинен дуже елегантно збалансувати картину загальної економіки, яка називається макро, і низького рівня управління окремими одиницями, що називається мікро .

Необхідність збалансувати короткострокові та довгострокові цілі та адаптуватися до непередбачених ситуацій є великим викликом для систем, які часто є абсолютно негнучкими. Вирішення цієї проблеми вимагає прориву в декількох областях штучного інтелекту:

Теорія ігор: StarCraft - це гра, де, як і в комені, ножицях, папері, немає єдиної стратегії виграшу. Тому в процесі навчання штучний інтелект повинен постійно досліджувати та розширювати горизонти своїх стратегічних знань.

Неповна інформація: на відміну від шахів або гри, де гравці бачать все, що відбувається, важлива інформація в StarCraft часто приховується і повинна бути активно отримана шляхом дослідження.

Довгострокове планування: Як і в реальних проблемах, причинно-наслідкові зв'язки можуть бути не миттєвими. Гра також може тривати годину і більше, тому дії, які виконуються на початку гри, можуть не мати взагалі ніякого значення.

Реальний час: На відміну від традиційних настільних ігор, де учасники по черзі роблять по черзі, в StarCraft гравці виконують дії постійно, разом із часом.

Величезний простір дій: за сотнями різних одиниць і будівель слід контролювати одночасно, в режимі реального часу, що дає справді величезний комбінаторний простір можливостей. На додаток до цього, багато дій є ієрархічними і їх можливо змінювати і доповнювати по ходу. Наша параметризація гри дає в середньому приблизно від 10 до 26 дій за одиницю часу.

Завдяки цим викликам StarCraft став великим викликом для дослідників штучного інтелекту. Нинішні змагання StarCraft та StarCraft II починаються з моменту запуску BroodWar API в 2009 році. Серед них - AIIDE StarCraft AI Competition, змагання CIG StarCraft , студентський турнір StarCraft AI та Starcraft II AI Ladder.

Примітка. У 2017 році PatientZero опублікував на Habré чудовий переклад «Історії змагань AI Starcraft».

Щоб допомогти громаді вивчити ці проблеми далі, працюючи разом із Blizzard у 2016 та 2017 роках, опублікували журнал-інструментарій PySC2, що включає найбільший масив відтворення, який коли-небудь публікував. На основі цієї роботи поєднали наші інженерні та алгоритмічні досягнення для створення AlphaStar.

Візуалізація AlphaStar під час боротьби з MaNa демонструє гру як агент початкові спостережувані дані, активність нейронної мережі, деякі запропоновані

дії та необхідні координати та очікуваний результат матчу. Показано також зовнішній вигляд гравця MaNa, але, звичайно, він недоступний для агента.

Як проходить навчання Поведінка AlphaStar генеруються з допомогою глибокого навчання нейронної мережі, яка отримує вихідні дані через інтерфейс (список вузлів і їх властивості) і дає послідовність інструкцій, які дії в грі. Більш конкретно, архітектура нейронної мережі використовує підхід « трансформувати тулуб до одиниць, поєднаний з глибоким ядром LSTM, авторегресивною головою політики з мережею вказівника

А та централізованою базовою лінією значення» (для точності термінів, залишених неперекладеними). Ці моделі надалі допоможуть впоратися з іншими важливими завданнями машинного навчання, включаючи довгострокове моделювання послідовностей та великі простори виводу, такі як на рисунку 2.2,

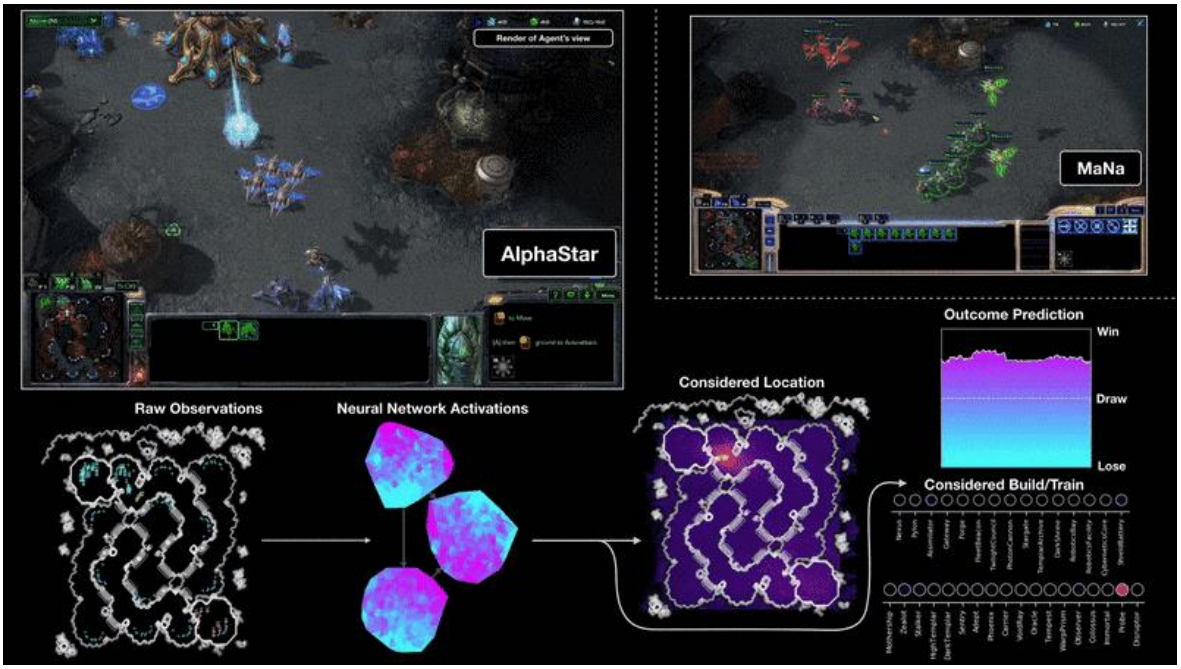

Рисунок 2.2 візуалізація AplphaStar

переклад, моделювання мови та візуальні уявлення AlphaStar також використовує новий алгоритм навчання з кількома агентами Ця нейронна мережа спочатку була навчена за допомогою методу навчання, заснованого на вчителях, на основі анонімованих повторів, доступних через Blizzard. Це дозволило AlphaStar вивчити

та моделювати основні мікро - та макро стратегії, які використовуються гравцями в турнірах. Цей агент переміг вбудований штучний інтелект рівня Еліта, що еквівалентно рівню гравців золотої ліги, в дев'яносто п'яти процентах тестових ігор.

MMR (Match Making Rating) - приблизний показник майстерності гравця. Для суперників у лізі AlphaStar, рисунок 2.3 та рисунок 2.4, порівняно з онлайн-лігами.

Ліга AlphaStar. Агенти спочатку навчалися на основі повторів людських матчів, а потім на основі змагальних матчів між собою. На кожній ітерації нові суперники розщеплюються, а початкові суперники замерзають. Шанси на зустріч з іншими опонентами та гіперпараметрами визначають цілі навчання кожного агента, що збільшує складність, що зберігає різноманітність. Параметри агента оновлюються тренуванням підкріплення на основі результату гри проти супротивників. Кінцевий агент вибирається (без заміни) на основі розподілу.

-

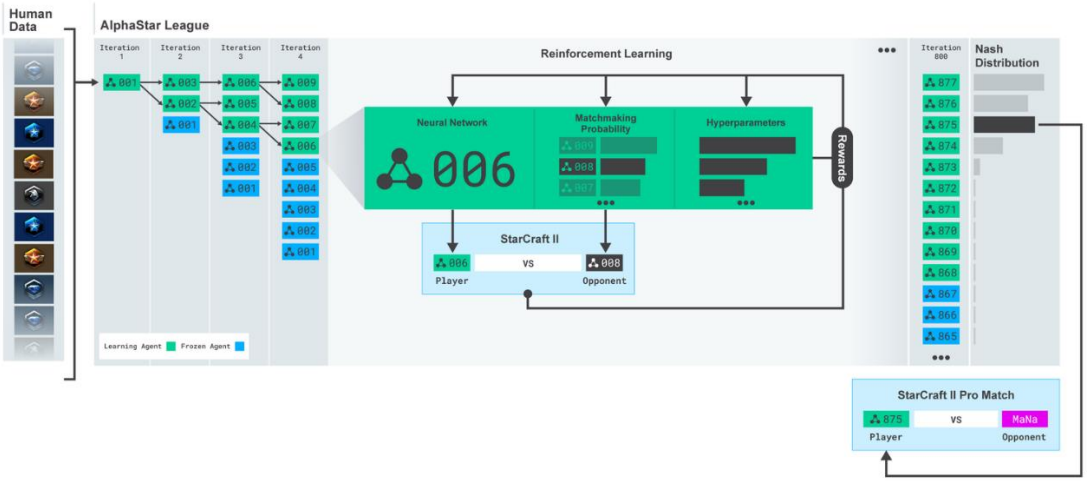

Рисунок 2.3 візуалізація тестових ігор AlphaStar

Ці результати потім використовуються для початку багатоагентного процесу навчання із підкріпленням. Для цього була створена ліга, де опоненти грають один проти одного, як і люди отримують досвід гри в турніри. До ліги додалися нові суперники шляхом дублювання діючих агентів. Така нова форма навчання, запозичивши деякі ідеї з методу тренування з підкріпленням елементами алгоритмів на основі населення, дозволяє створити безперервний процес

дослідження величезного стратегічного простору ігрового процесу StarCraft і бути впевненим, що агенти можуть протистояти найсильніші стратегії.

З розвитком ліги та створенням нових агентів з'явилися контрстратегії, які змогли перемогти попередні. У той час як деякі агенти лише вдосконалювали стратегії, які були задоволені раніше, інші агенти створювали абсолютно нові, включаючи нові незвичайні замовлення збірки, склад підрозділів та макроуправління. Наприклад, на ранній стадії сир процвітав - швидко поспішайте з фотонами (Photon Cannons) гарматами або темними тамплієрами (Dark Templars). Але по мірі прогресування навчального процесу ці ризиковані стратегії були відкинуті, поступившись місцем іншим. Наприклад, виробництво надлишкової кількості робітників для отримання додаткового притоку ресурсів або пожертвування двох оракулів (Oracles) завдати удару проти робітників противника і підірвати їх економіку. Цей процес схожий на те, як звичайні гравці відкривали нові стратегії та перемагали старі популярні підходи протягом багатьох років.

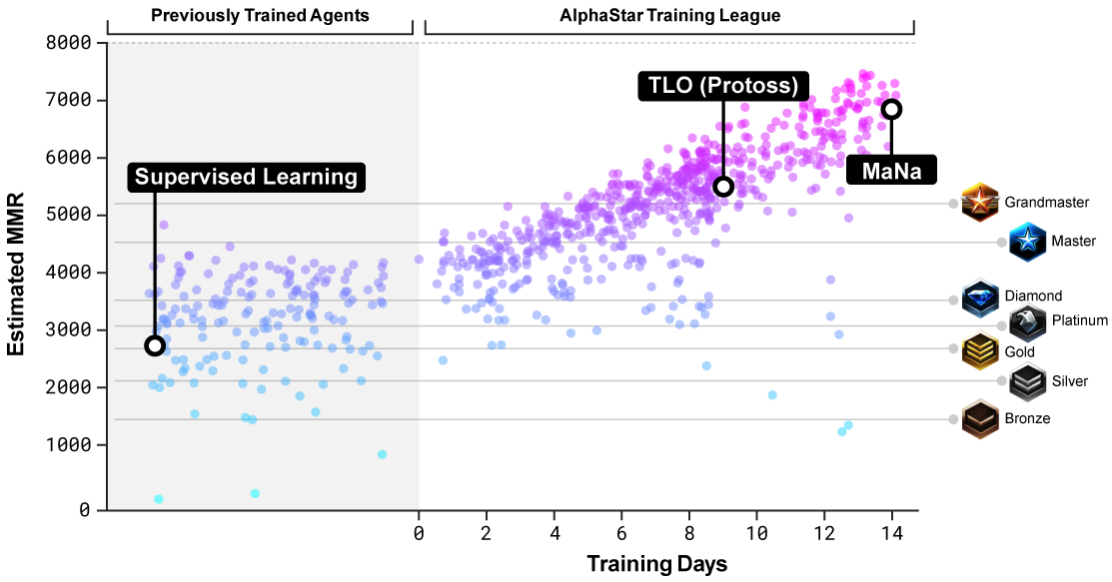

Рисунок 2.4 Match Making Rating показник майстерності

По мірі прогресування тренінгу було помітно, як змінювався склад одиниць. Для забезпечення різноманітності кожен агент наділений своєю власною метою

навчання. Наприклад, які опоненти повинні виграти цього агента, або будь-яка інша властива мотивація, яка визначає гру агента. У певного агента може бути мета перемогти одного конкретного противника, а інший - цілу вибірку супротивників, але робити це лише певними одиницями, як на рисунку 2.5.

Інтерактивна візуалізація (інтерактивні функції доступні в оригінальній статті), де показані суперники з ліги AlphaStar. Окремо відзначається агент, який грав проти TLO та MaNa.

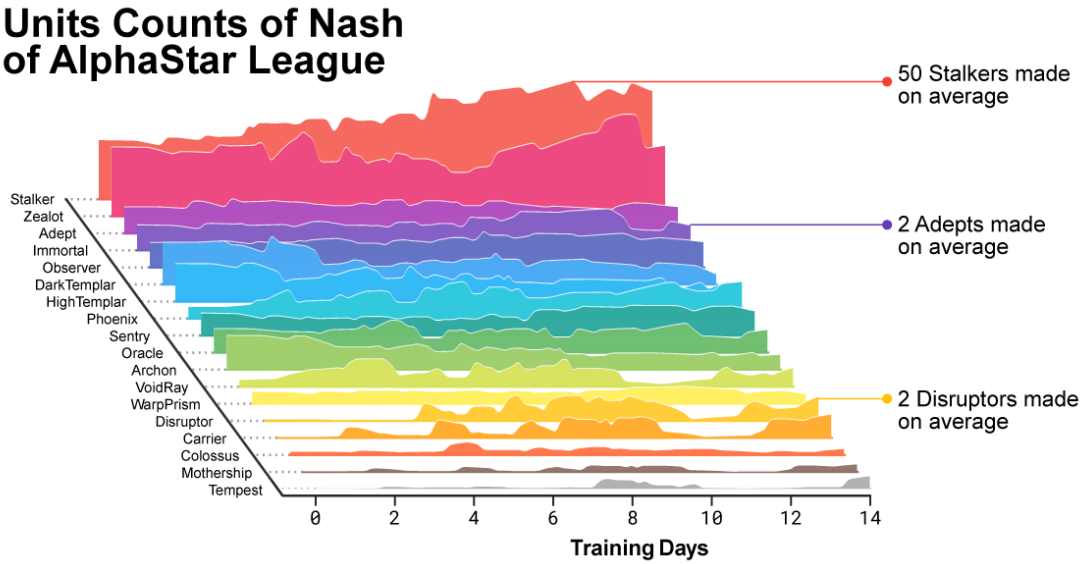

Рисунок 2.5 Match Making Rating показник майстерності у тренуванні

Коефіцієнти (ваги) нейронної мережі кожного агента були оновлені за допомогою посиленого навчання на основі ігор з супротивниками з метою оптимізації їх конкретних навчальних цілей. Правило оновлення ваги - новий ефективний алгоритм для вивчення «акторського режиму, що не відповідає політиці, -алгоритм навчання критики підкріплення з переглядом досвіду, навчання само імітації та дистиляція політики» (для точності термінів, що залишилися без перекладу).

Зображення показує, як один агент (чорна крапка), який в кінцевому підсумку був обраний для гри проти MaNa, розробляв свою стратегію порівняно зі своїми опонентами (кольоровими точками) у процесі навчання. Кожен очко

представляє суперника в лізі. Положення точки показує стратегію, а розмір показує частоту, з якою вона вибирається як опонент для агента MaNa в процесі навчання.

Як AlphaStar діє і бачить гру - Професійні гравці, такі як TLO або MaNa, здатні виконувати сотні дій в хвилину (APM), рисунок 2.6 та 2.7. Але це набагато менше, ніж у більшості існуючих ботів , які незалежно керують кожною одиницею і генерують тисячі, якщо не десятки тисяч дій.

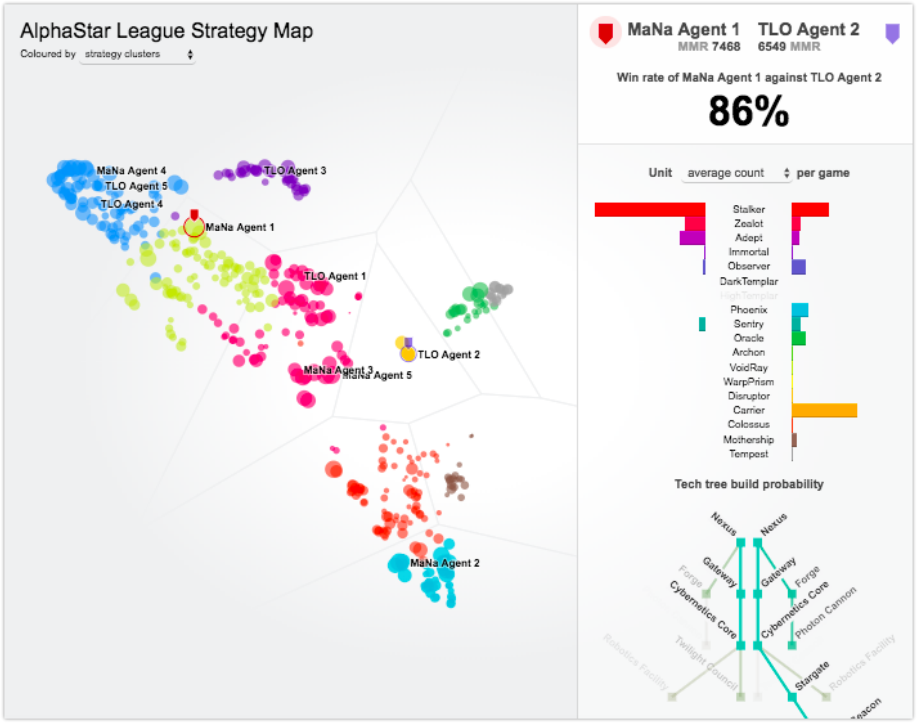

Рисунок 2.6 Агенти MaNa в процесі навчання

Розподіл між суперниками під час розвитку ліги та створення нових опонентів. Розподіл, який встановлюється додатковими конкурентами, високо оцінює нових гравців, тим самим демонструючи постійний прогрес порівняно.

Для тренінгу AlphaStar створили масштабовану розподілену систему на базі Google TPU3, який забезпечує паралельний процес навчання для всієї групи агентів з тисячами запущених копій StarCraft II. AlphaStar League працював 14 днів, використовуючи 16 ТПУ для кожного агента. Під час тренінгу кожен агент мав до 200 років досвіду гри в StarCraft в режимі реального часу. Остаточна версія агента

AlphaStar містить компоненти розподілу по всій лізі. Іншими словами, найефективніший поєднання стратегій, які були виявлені під час ігор і цю конфігурацію можливо запустити на одному стандартному настільному графічному процесорі. Повний технічний опис зараз готується до публікації у рецензованому науковому журналі.

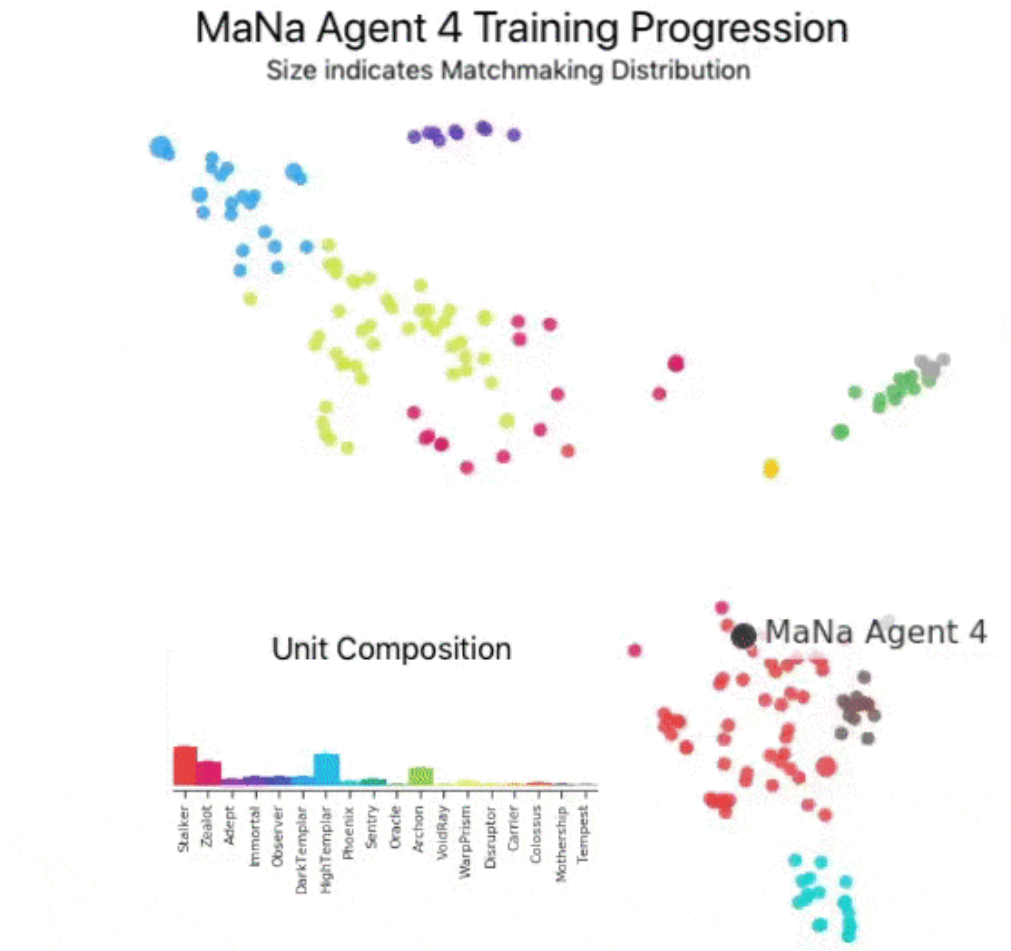

Рисунок 2.7 Процес навчання

У наших іграх проти TLO та MaNa AlphaStar утримував APM на середньому рівні 280, що набагато менше, ніж у професійних гравців, хоча його дії можуть бути більш точними.Такий низький APM частково пояснюється тим, що AlphaStar почав вчитися на повторах звичайних гравців і намагався наслідувати манері людської гри. На додаток до цього, AlphaStar відповідає затримкою між спостереженням та дією в середньому близько 350 мс.

Розподіл APM AlphaStar у матчах проти MaNa та TLO та загальна затримка між спостереженням та дією.

Крім того, розробили другу версію AlphaStar. Як людські гравці, ця версія AlphaStar чітко вибирає, коли і куди рухати камеру. У цьому варіанті його сприйняття обмежується інформацією на екрані, а дії також допустимі лише на видимій частині екрана, рисунок 2.8.

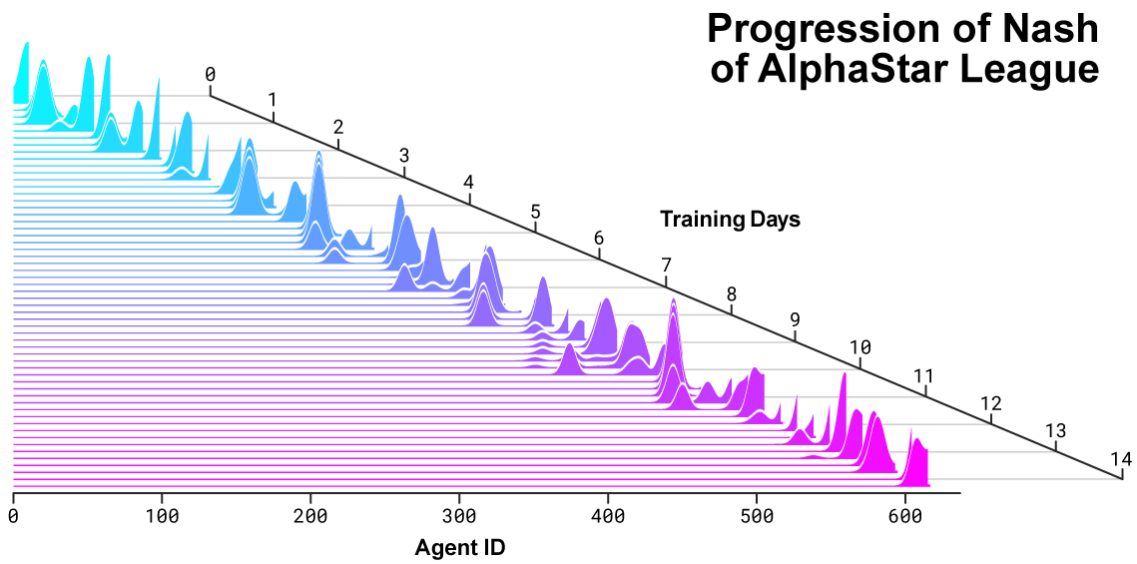

Рисунок 2.8 Навчальні дні

Під час матчів проти TLO та MaNa AlphaStar взаємодіяв з ігровим двигуном StarCraft через основний (сирий) інтерфейс, тобто він міг бачити атрибути своїх і видимих ворожих підрозділів на карті безпосередньо, не потребуючи переміщення камери - ефективно грати зі зменшеним видом на всю територію. Всупереч цьому, живі люди повинні чітко керувати економією уваги, щоб постійно вирішувати, куди слід сфокусувати камеру. Однак аналіз ігор AlphaStar показує, що вона неявно контролює фокус уваги. В середньому агент перемикає свій контекст уваги приблизно 30 разів на хвилину, як MaNa та TLO.

Крім того, розробили другу версію AlphaStar. Як людські гравці, ця версія AlphaStar чітко вибирає, коли і куди рухати камеру. У цьому варіанті його

сприйняття обмежується інформацією на екрані, а дії також допустимі лише на видимій частині екрана.

Продуктивність AlphaStar при використанні базового інтерфейсу та інтерфейсу з камерою. На графіку видно, що новий агент, що працює з камерою, швидко досягає порівнянних характеристик агента за допомогою базового інтерфейсу, рисунок 2.9,

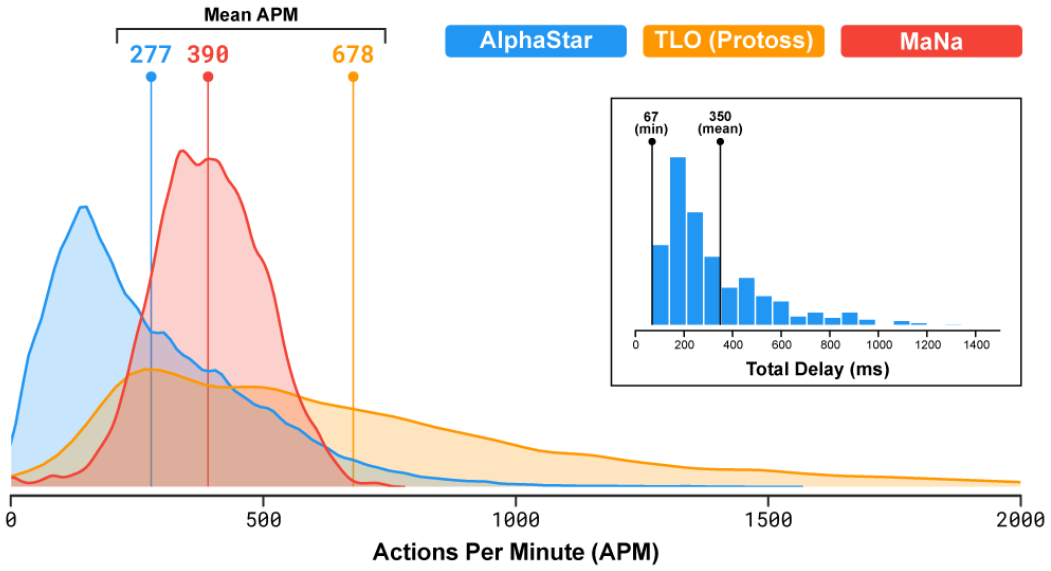

Рисунок 2.9 Дії за хвилину

підготували двох нових агентів, одного за допомогою базового інтерфейсу та одного, який повинен був навчитися керувати камерою, граючи проти ліги AlphaStar. На початку кожен агент проходив навчання з викладачем на основі людських сірників, після чого проводилося навчання з підкріпленням, описане вище. Версія AlphaStar, що використовує інтерфейс камери, досягла майже таких же результатів, як і версія базового інтерфейсу, перевищуючи позначку 7000 MMR на нашому внутрішньому лідері. У зразковому поєдинку MaNa перемогла прототип AlphaStar за допомогою камери. Тренували цю версію лише 7 днів.

Ці результати показують, що успіх AlphaStar у матчах проти MaNa та TLO насамперед результат хорошого макро- та мікро-управління, а не лише великої швидкості натискань, швидкої реакції чи доступу до базової інформації інтерфейсу.

Результати гри AlphaStar проти професійних гравців - StarCraft дозволяє гравцям вибирати одну з трьох рас, зерг або протос. Вирішено, що AlphaStar на даний момент спеціалізується на одній конкретній гонці - Protoss, щоб скоротити час тренувань і відхилення в оцінці результатів нашої внутрішньої ліги. Але слід зазначити, що подібний процес навчання може бути застосований до будь-якої раси. Наші агенти були навчені грати в StarCraft II версії 4.6.2 в режимі протос, на карті CatalystLE. Щоб оцінити ефективність AlphaStar, спочатку у матчах проти TLO - професійного гравця за Zerg та гравця за Protoss рівня GrandMaster. AlphaStar виграв матчі 5-0, використовуючи широкий спектр одиниць та будуючи замовлення. «AlphaStar приймає відомі стратегії і перевертає їх догори ногами. Агент показав такі стратегії, про які я ніколи навіть не думав. І це свідчить про те, що ще можуть існувати способи гри, які ще не були вивчені повністю.

Після додаткового тижня тренувань зіграли проти MaNa, одного з найсильніших гравців StarCraft II у світі, та увійшли до топ-10 найсильніших гравців протосу. Цього разу AlphaStar виграв з рахунком 5: 0, продемонструвавши сильні навички мікро-менеджменту та макростратегії. Побачивши, як AlphaStar використовує найсучасніші підходи та різні стратегії в кожній грі, демонструючи дуже людський стиль гри, AlphaStar та інші складні питання - незважаючи на те, що StarCraft - це лише гра, навіть якщо це дуже важко, методи, що лежать в основі AlphaStar.

Наприклад, цей тип архітектури нейронної мережі здатний імітувати дуже довгі послідовності ймовірних дій, в іграх, які часто тривають до повної години і містять десятки тисяч дій на основі неповної інформації. Кожен кадр у StarCraft використовується як один крок введення. У цьому випадку нейронна мережа кожного такого кроку передбачає очікувану послідовність дій для решти гри. Основне завдання складання складних прогнозів для дуже довгих послідовностей даних зустрічається у багатьох проблемах реального світу, таких як прогнозування погоди, моделювання клімату, розуміння мови тощо, Величезний потенціал як на рисунку 2.10.

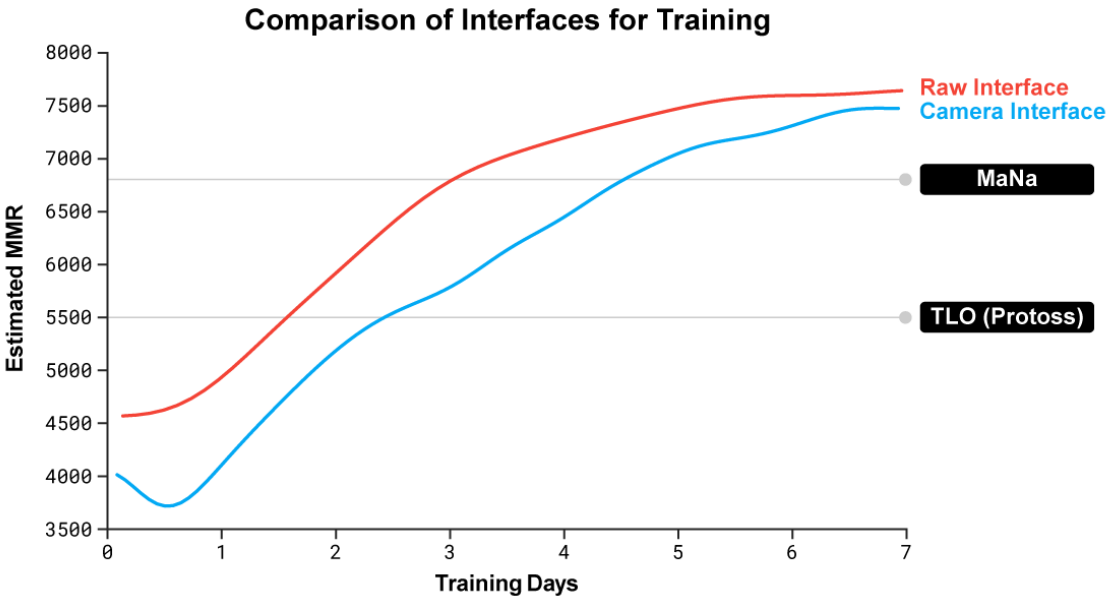

Рисунок 2.10 Порівняння інтерфейсів для навчання

Деякі наші методи навчання можуть бути корисними при вивченні безпеки та надійності штучного інтелекту. Однією з найскладніших проблем у галузі штучного інтелекту є кількість варіантів, за які система може помилитися. А професійні гравці в минулому швидко знаходили способи обійти штучний інтелект, спочатку використовуючи його помилки. Інноваційний підхід AlphaStar, заснований на лізі, знаходить такі підходи та робить загальний процес більш надійним та безпечним від таких помилок. Потенціал такого підходу може допомогти підвищити безпеку та надійність систем штучного інтелекту в цілому. Зокрема, у таких критичних сферах, як енергетика, де вкрай важливо правильно реагувати у складних ситуаціях.

Досягнення такого високого рівня гри StarCraft - це великий прорив в одній із найскладніших відеоігор, що коли-небудь створених. Ці досягнення, поряд із успіхами в інших проектах, чи то AlphaZero, чи AlphaFold , - це крок вперед до реалізації нашої місії створення інтелектуальних систем, яка одного дня допоможе нам знайти рішення для найскладніших та фундаментальних наукових питань.

2.2 Аналіз результатів досліджень у прикладі

Дослідники оприлюднили новини про те, щоб їх амбіції штучні інтелекти грали у грізну гру в хованки з грізними результатами. У середовищі агентів були стіни та пересувні ящики для виклику, де одні були хованками, а інші - шукачами. Багато чого відбулося по дорозі, з сюрпризами.

Зазначаючи те, що було вивчено, автори ведуть блог: Спостерігали, як агенти виявляють прогресивніше складніше використання інструментів, граючи в просту гру в хованки, де агенти побудували серію з шести різних стратегій і контрстратегій, деякі з яку не знали, що наше середовище підтримує.

У новому документі, опублікованому на початку цього тижня, команда виявила результати. У їх документі Невідкладне використання інструменту з багатовалентних автокурсів було сім авторів, у шести з яких було вказано представництво OpenAI, а на одному - Google Brain.

Автори прокоментували, який виклик вони беруть на себе. Створення розумних штучних агентів, здатних вирішити широкий спектр складних завдань, що стосуються людини, стало давнім завданням для спільноти штучного інтелекту.

Агенти створюють автоконтроль само нагляду, індукуючи декілька чітких раундів нової стратегії, багато з яких потребують складного використання інструментів та координації.

За допомогою хованок, шукачі навчилися переслідувати ховальників, а хованки навчилися тікати (2) Хідники навчились основного використання інструментів - ящиків та стін для будівництва фортів. шукачі навчилися використовувати пандуси, щоб стрибати у притулок для ховальників Прихованці навчилися переміщати пандуси туди, куди вони будуватимуть свій форт, і замикають їх на місці. Шукачі дізналися, що можуть перестрибувати із замкнених пандусів до ящиків і прибігати до коробки до притулку для ховальників і Прихованці навчилися фіксувати невикористані ящики перед тим, як побудувати свій форт.

Ці шість стратегій з'явилися як агенти, навчені один одному в хованках кожна нова стратегія створювала раніше неіснуючий тиск для агентів переходити до наступного етапу без будь-яких прямих стимулів для агентів взаємодіяти з об'єктами чи досліджувати. Стратегії були результатом автокурсу, викликаного багатоагентською конкуренцією та динамікою хованок.

Автори в блозі зазначають, що вони дізналися, що часто трапляється так, що агенти знаходять спосіб ненавмисно використовувати середовище, яке ви будуєте, або фізичний двигун.

Те, що відбувалося, було складною само наглядною складною ситуацією. І це ще більше говорить про те, що багатоагентна адаптація може одного дня спричинити надзвичайно складну та розумну поведінку. Автори аналогічно заявляли у своїй роботі, що спонукання автокурсу у фізично заземлених та відкритих середовищах може врешті-решт дати агентам можливість придбати необмежену кількість вмінь, що стосуються людини.

Спочатку хованки просто втекли. Але незабаром вони розробили, що найшвидшим способом провокувати шукачів було знайти предмети в навколишньому середовищі, щоб сховатися від зору, використовуючи їх як певний інструмент. Наприклад, вони навчилися що скриньки можуть бути використані для блокування дверних прорізів і побудови простих сховищ. Шукачі дізналися, що вони можуть переміщати пандус і використовувати його, щоб перелізти стіни. Тоді боти виявили, що є гравцем команди - передаючи об'єкти один одному або співпрацюючи на приховування – було найшвидшим способом перемогти».

Рисунок 2.11 це амбітний проект, MIT Technology Review зазначив, що штучний інтелект навчився користуватися інструментами після майже 500 мільйонів ігор в хованки[5]. За допомогою гри в хованки сотні мільйонів раундів дві протилежні команди агентів штучного інтелекту розробили складні стратегії приховування.

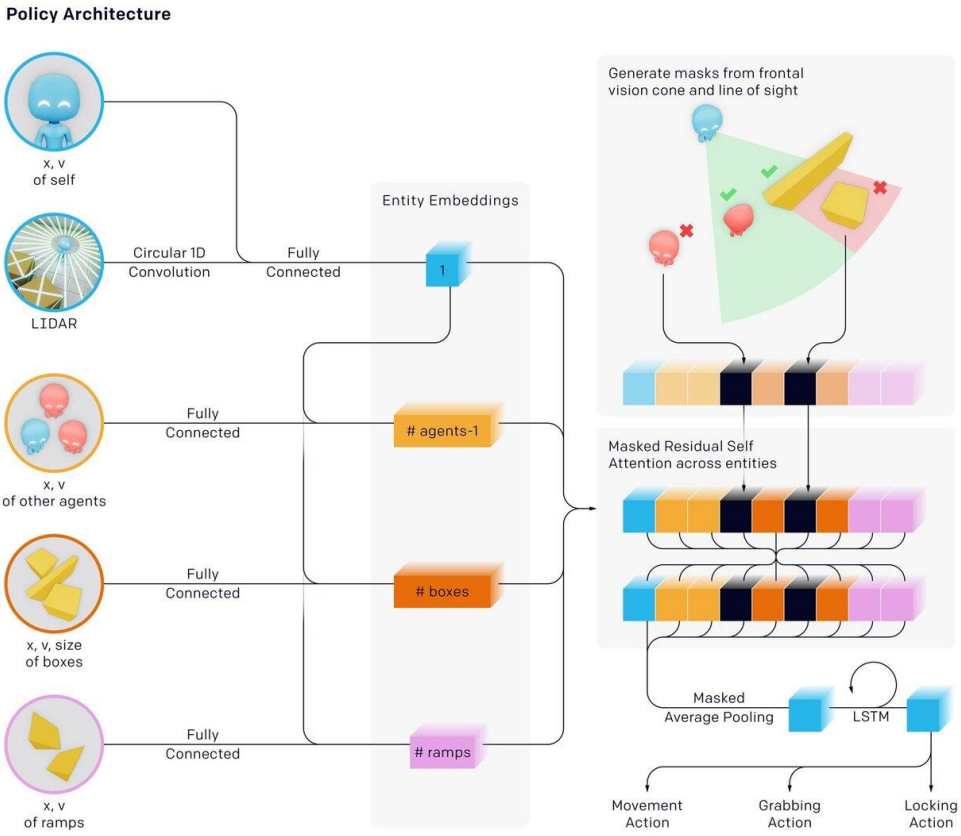

Рисунок 2.11 Поведінка штучного інтелекту у грі у хованки

Більше мільйонів патронів: шукачі виявили контр-стратегію, коли вони навчилися рухати пандус поруч із фортом ховальників і використовувати його, щоб перелізти через стіни. Більше турів пізніше, хованки навчилися фіксувати пандуси на місці, перш ніж будувати свій форт. Ще 380 стратегій виграли ще більше стратегій з'явилися ще дві стратегії. Шукачі розробили стратегію вторгнення у форт ховальників, використовуючи заблоковану пандус[6], щоб піднятися на незамкнутий ящик, а потім проїхати по коробці до форту та над його стінами. На завершальній фазі хованки ще раз навчилися фіксувати всі пандуси та ящики на місці.

Дослідження передбачало і успішно вивчало можливість машинного навчання агентів освоювати складні , реальну методику, без втручання пропозицій дослідників.

3 ДОСЛІДЖЕННЯ І ІННОВАЦІЇ ШТУЧНОГО ІНТЕЛЕКТУ

3.1 Прогрес штучного інтелекту

Штучний інтелект відкриває нові можливості у відеоіграх, навіть старих відеоіграх. Цього місяця команда італійських дослідників під керівництвом Едоардо Джакомелло з Politecnico di Milano використовувала штучний інтелект для створення нових рівнів Doom, новаторська гра-шутер від першої особи, вперше випущена в 1993 році. Аналізуючи 1000 офіційних рівнів Doom, що повертаються до Doom 1 та Doom 2, а також 9000 рівнів, створених ігровим співтовариством, команда змогла визначити ключові особливості рівнів Doom. і використовувати їх для автоматичного генерування нових рівнів. Після 36000 ітерацій рівень штучний інтелект, що генерується штучний інтелект, досягнув відтворюваної якості, порівнянної з рівнем, призначеним людиною. Дослідники сподіваються, що ця програма штучний інтелект полегшить компанії для відеоігор створення нового контенту, одного з багаторічних викликів для ігрових дизайнерів. штучний інтелект генерував рівень приреченості

Досягнення команди Джакомелло ілюструє, як штучний інтелект змінює світ відеоігор. Ось подивіться на три інші способи штучний інтелект перетворює дизайн та гру відеоігор. Робити розумніших персонажів, які не грають

Найбільш поширене використання штучного інтелекту у відеоіграх це генерування більш інтелектуальних персонажів, які не грають Традиційні NPC поводяться за заздалегідь запрограмованими правилами, роблячи їх передбачуваними для досвідчених гравців. штучний інтелект може змусити NPC поводити себе більш розумно, використовуючи алгоритм, який називається Кінцева машина, популяризований у відеоіграх протягом 90-х років через такі ігри, як Dragon Quest IV Warcraft та Half-Life. FSM передбачає всі можливі ситуації, з якими може зіткнутися програма штучний інтелект гри, а потім генерує реакцію на кожну подію. FSM передбачає всі можливі ситуації, з якими може зіткнутися програма гри, а потім генерує реакцію на кожну подію, рисунок 3.1.

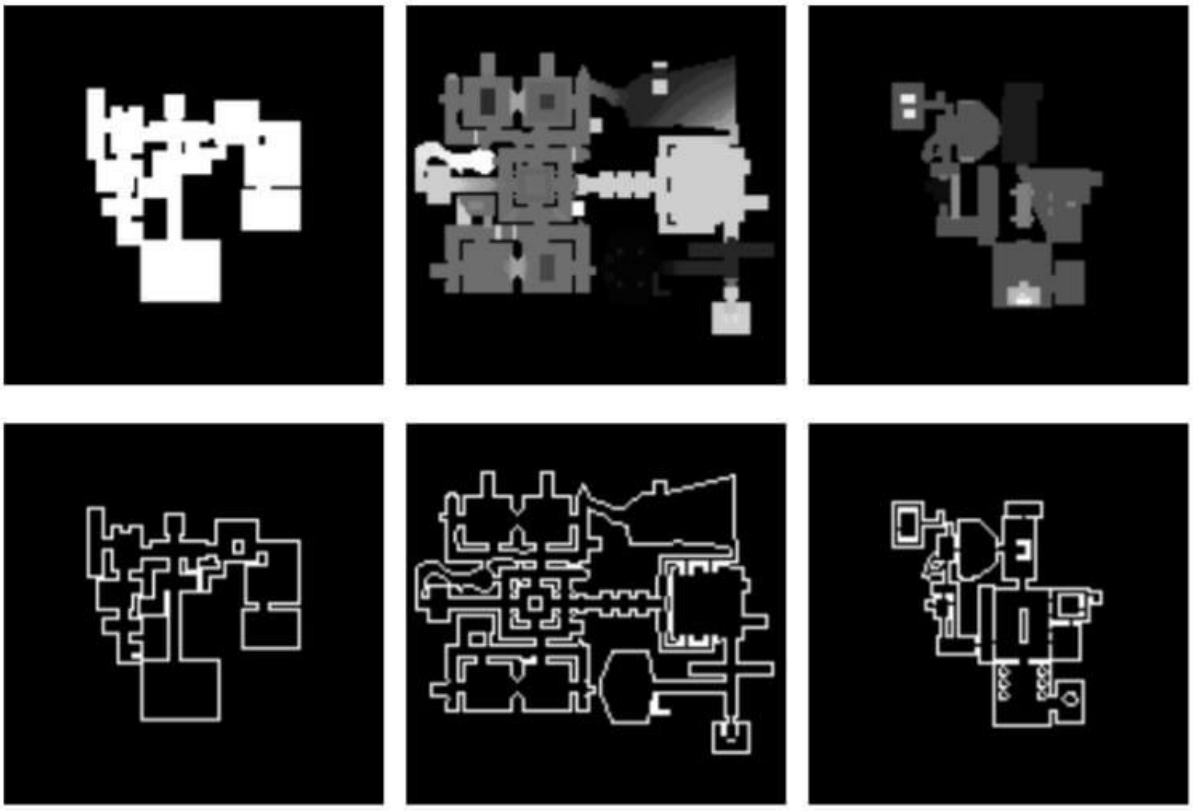

Рисунок 3.1 Штучний інтелект генерує рівні

Наприклад, у фантазійній рольовій грі FSM може мати атаку NPC, якщо гравець з’явиться, а потім відступить, коли пункти ударів впадуть нижче певного рівня. Більш просунуті FSM використовують машинне навчання, щоб змусити NPC адаптуватися до ігрових звичок гравців, роблячи їх більш непередбачуваними та складнішими.

З ранніх днів штучний інтелект ігри були популярним шляхом для програм штучного інтелекту та додатків для самостійного навчання. Сьогодні деякі найбільш інноваційні технології, які використовуються для повного домінування в ретро та старих шкільних іграх. Одна з найвідоміших програм штучний інтелект в іграх була створена філією Google DeepMind. DeepMind вперше почав створювати AlphaGo, штучний інтелект -бот, який раніше грав, Go, китайську настільну гру. Go був винайдений понад 2000 років тому і є найстарішою грою в світі, яка все ще грається в сучасний час. У 2016 році DeepMind створив AlphaGo, штучний інтелект -бота, який використовував складні алгоритми та самонавчання, щоб перемогти

одного з найкращих гравців Go у світі, ставши одним із перших випадків, коли бот штучний інтелект зміг перемогти гравця вищого класу . Але програма AlphaGo не зупинилася на Go. AlphaGo також застосовується до інших багатокористувацьких ігор, таких як шахи. Зараз називається AlphaZero, програма Google використовувала ті самі методи самонавчання, що і їх колега Go, і навчилася грати в шахи лише за чотири години. Тоді AlphaZero був висунутий проти програми світових чемпіонів Stockfish 8. Крім шахів, AlphaZero також міг грати в інші настільні ігри, такі як шогі, швидко навчатись грі, а потім бити у верхній програмі. Неймовірні подвиги AlphaZero[6] змогли заволодіти і домінувати в іграх усі самостійно без допомоги людини. У кожній грі, яку він грав, технології штучний інтелект давали лише основні правила кожної гри. Провівши гру протягом декількох годин, штучний інтелект зміг посилити своє навчання, стаючи все кращим та кращим з кожним.

Поряд зі старими іграми, ретро-ігри, такі як Breakout, Pong і навіть Space Invaders, стали наступними мішенями для штучний інтелект. Ретро-ігри були так само легкими, якщо не простішими підкорити для AlphaGo. Пройшовши лише пару годин гри в обидві ці ігри, AlphaGo зміг досягти навичок, подібних до богів.

В останніх новинах DeepMind та інші програми штучний інтелект впроваджували години в більш складні та сучасні ігри. Одні з найпопулярніших напрямків - це ігри Starcraft II, DOTA 2, і особливо онлайн-покер. Однією з найбільших проблем для штучного інтелекту було подолання обчислювальної сили та ресурсів, необхідних для гри в складні ігри. Раніше програми штучного інтелекту, застосовувані до великих багатокористувацьких стратегічних ігор, вимагають багато енергії. Особливо, граючи наживо проти людей, програми штучні інтелекти повинні створювати рішення в реальному часі та обчислювати нескінченні стратегії за лічені секунди. Обчислювальні ресурси, необхідні для гри в складні ігри, є астрономічними, і програмам ще важче зрозуміти.

Все це змінилося зі створенням програми Starcraft DeepMind Starcraft, яку вони назвали AlphaStar. У Starcraft головна мета полягає в тому, щоб збирати матеріали для збору різних армій та військ. Ці війська можуть бути використані для нападу на інших гравців, поки один не залишиться стояти. Мета DeepMind своєю

новою програмою полягала в інноваційному програмуванні штучний інтелект для подолання цих попередніх проблем. AlphaStar зміг піднятися на гори, що вважалися занадто крутими для штучного інтелекту, і став одним з найкращих гравців Starcraft II у світі. Він розпочався з гри та побиття професійного гравця Даріо TLO Wünsch, а потім перейшов до свого наступного суперника людини, MaNa. AlphaStar був на відміну від своїх попередників, вміючи легко обробляти обчислювальну потужність, коли він тупотів проти своїх ворогів людини. На відміну від людських гравців, AlphaStar рухався з точністю, роблячи швидкі та рішучі дії, на які багато людських гравців не здатні.

Поряд з програмою AlphaStar штучний інтелект є боти DOTA 2 OpenAI. OpenAI - це стартап Elon Musk, який зосередився на створенні програми штучного інтелекту для підкорення світу DOTA 2. Аналогічно Starcraft II, DOTA 2 - це багатокористувацька гра в Інтернеті, яка в минулому була конкуренцією штучного інтелекту. У квітні 2019 року OpenAI провела змагання Final Five, збігаючи свою програму штучний інтелект проти команди з електронних видів спорту OG, чемпіонів світу з DOTA 2. Аналогічно AlphaStar, OpenAI використовував версію посиленого навчання для програмування свого штучного інтелекту. Замість того, щоб кодувати його для використання певних стратегій, їх програма була закодована, щоб навчитися та покращити понаднормово. Оскільки це була програма, вони могли грати в прискореному середовищі, граючи до 180 років щоб підготуватися проти своєї людської опозиції. Боти будуть постійно розумнішими та кращими, як грали. Під час змагань OpenAI змогла оптимізувати використання штучного інтелекту та самонавчання, щоб стати богами проти людських гравців.

Тому боти повинні створювати ігрові плани з невеликою кількістю інформації, яку вони мають.

Але боти не безпорадні. На відміну від звичайних ботів, боти машинного навчання здатні оцінити та вивчити дії своїх противників людини, а також оцінити результат певних кроків. Через здатність вчитися, більш просунуті боти здатні вирішити ці ігри краще, ніж люди могли коли-небудь. То чому це має значення? Здатність перемагати людей в іграх сама по собі може бути не дуже вагомою, але

алгоритми та дані, що стоять за цими ботами, є важливими. Доведення того, що боти мають можливість вчитися та ставати кращими на таких складних заходах, - це перші кроки для створення все більш потужного штучного інтелекту.

Штучний інтелект продовжує бути великим гравцем в іграх, і це скоро не зупиниться. Ігри були величезною і невід'ємною частиною розвитку штучного інтелекту. Роблячи це, щтучному інтелекту дозволено самостійно впроваджувати інновації, навчаючи себе нових стратегій та методів оптимізації ігрового процесу. В останні роки штучний інтелект перейшов від базових ретро-ігор до теперішніх складних багатокористувацьких ігор. Інновації в штучний інтелект дозволили йому враховувати нескінченні змінні, а потім з легкістю приймати розділені.

Покер став одним із тематичних сайтів для тестування ботів з однієї конкретної причини: недосконала інформація. На відміну від ретро-ігор та навіть багатокористувацьких ігор, таких як DOTA 2, покер надає дуже мало інформації для роботи штучного інтелекту. Програма в покерному матчі нічого не знає про карти противника і має бути лише їх карти та карти таблиці. У покері є нескінченні змінні, до яких штучний інтелект повинен враховувати. Маючи так мало інформації, розробники штучного інтелекту зосереджуються на створенні алгоритмів, які розробляють успішні стратегії з невеликими та неповними наборами даних. Постійно вдосконалюючи штучний інтелект, особливо в галузі ігор, ці інноваційні технології можуть бути застосовані в інших сферах і галузях.

З прискоренням штучний інтелект та машинного навчання багато людей виходять на поле, не маючи великого досвіду. Розробка ігрових ботів - це цікавий та захоплюючий спосіб для нових ентузіастів та розробників штучного інтелект, щоб отримати досвід. Навіть для розробників-ветеранів створення бота з машинним навчанням - це відмінний спосіб перевірити нові алгоритми та теорії.

4 ОПИС ПРОГРАМНОЇ СИСТЕМИ ШТУЧНОГО ІНТЕЛЕКТУ

4.1 Програмна реалізація та оцінювання ефективності

Створюємо штучний інтелект для гри у двигуні UnrealEngine 4. Для створення штучного інтелекту вам знадобляться три речі: Тіло: це фізичне уявлення персонажа. В цьому випадку маффин це тіло, рисунок 4.1. Душа: душа - це сутність, що управляє персонажем. Це може бути гравець або штучний інтелект. Мозок: Мозок - це те, як штучний інтелект приймає рішення. Можливо створити це різними способами, такими як код C ++, Blueprints або дерева поведінки. Оскільки у вас вже є тіло, все, що вам потрібно, це душа і мозок. Спочатку ви створите контролер, який стане душею.

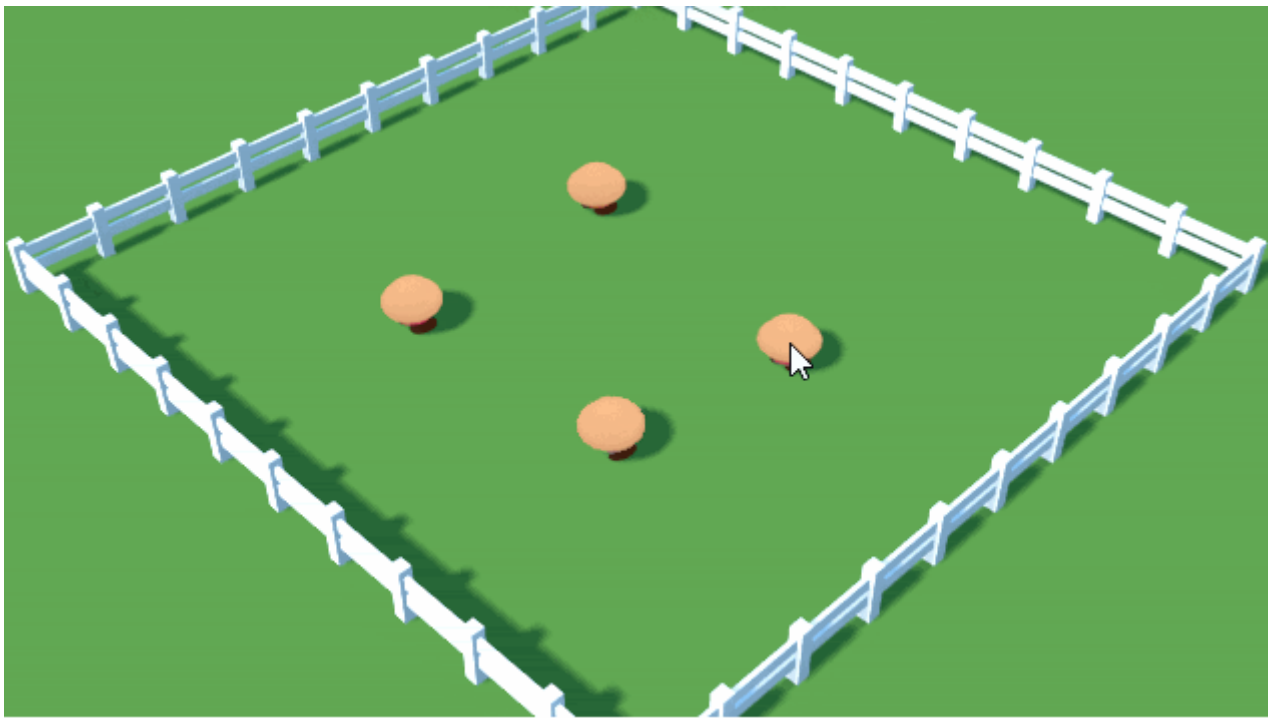

Рисунок 4.1 Тіло штучного інтелекту

Контролер - це нефізичний актор, який може володіти Пішаком. Володіння дозволяє контролеру - як ви вже здогадалися - контролювати пішки. Але що означає «контроль» в цьому контексті? Для гравця це означає натиснути кнопку і змусити

пішки щось зробити. Контролер отримує вхідні дані від гравця, а потім може відправляти вхідні дані в пішака. Контролер також може обробляти вводи і потім доручати Пішці виконати дію, рисунок 4.2.

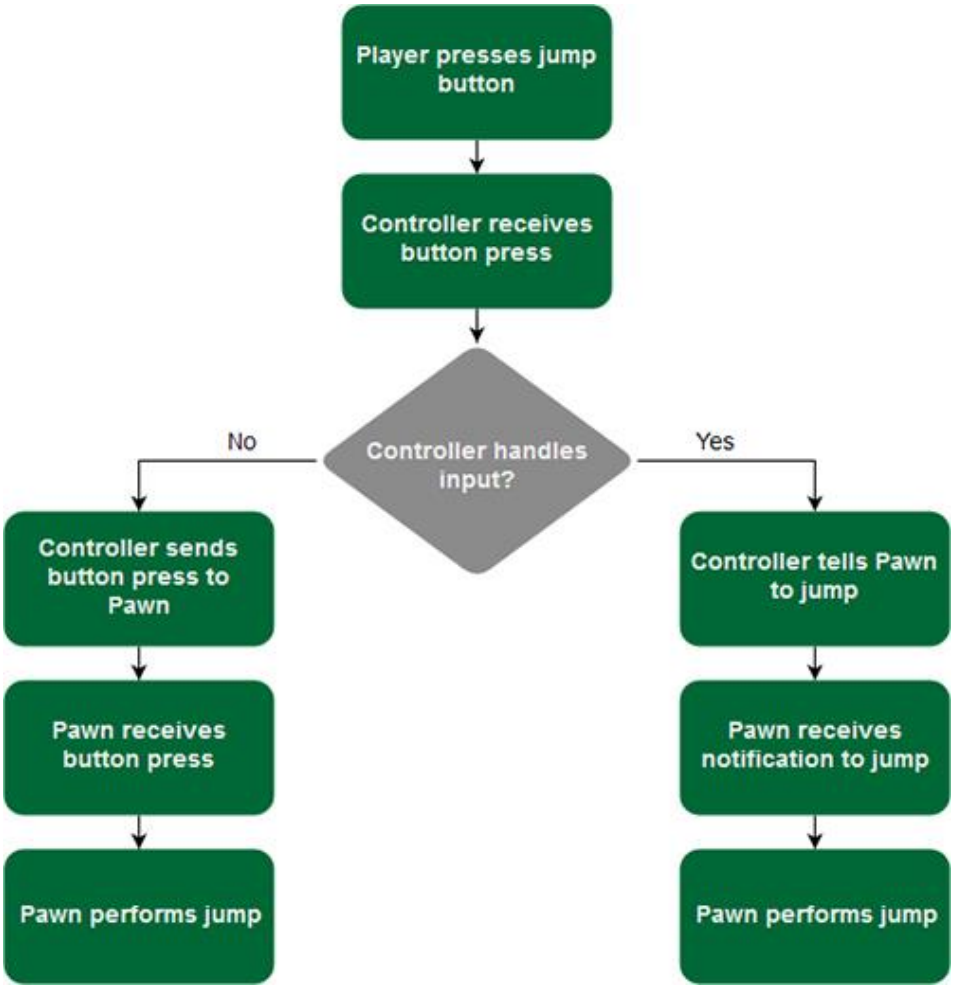

Рисунок 4.2 Контролер

У разі штучний інтелект, Пішак може отримувати інформацію від контролера або мозку (в залежності від того, як ви її запрограмуєте).

Щоб керувати кексами з допомогою штучного інтелекту, вам потрібно створити спеціальний тип контролера, відомий як контролер.

Дерева поведінки щоб виконати кожну дію в послідовності, ви повинні використовувати складову послідовність. Це тому, що послідовність виконує свої дочірні елементи зліва направо.

Якщо який-небудь з нащадків Послідовності зазнає невдачі, Послідовність зупиниться.

Наприклад, якщо Пішак не може переміститися на ворога, Move To Enemy зазнає невдачі. Це означає, що Rotate Toward Enemy і Attack не будуть виконані. Проте, вони виконаються, якщо Пішці вдасться перейти до ворога.

Далі створюємо дошку - це актив, єдина функція якого - зберігати змінні (відомі як ключі). Можливо думати про це як про пам'ять штучного інтелекту.

Хоча ви не зобов'язані їх використовувати, класні дошки пропонують зручний спосіб читання і зберігання даних. Це зручно, тому що багато вузлів в деревах поведінки приймають тільки ключі класної дошки, рисунок 4.3.

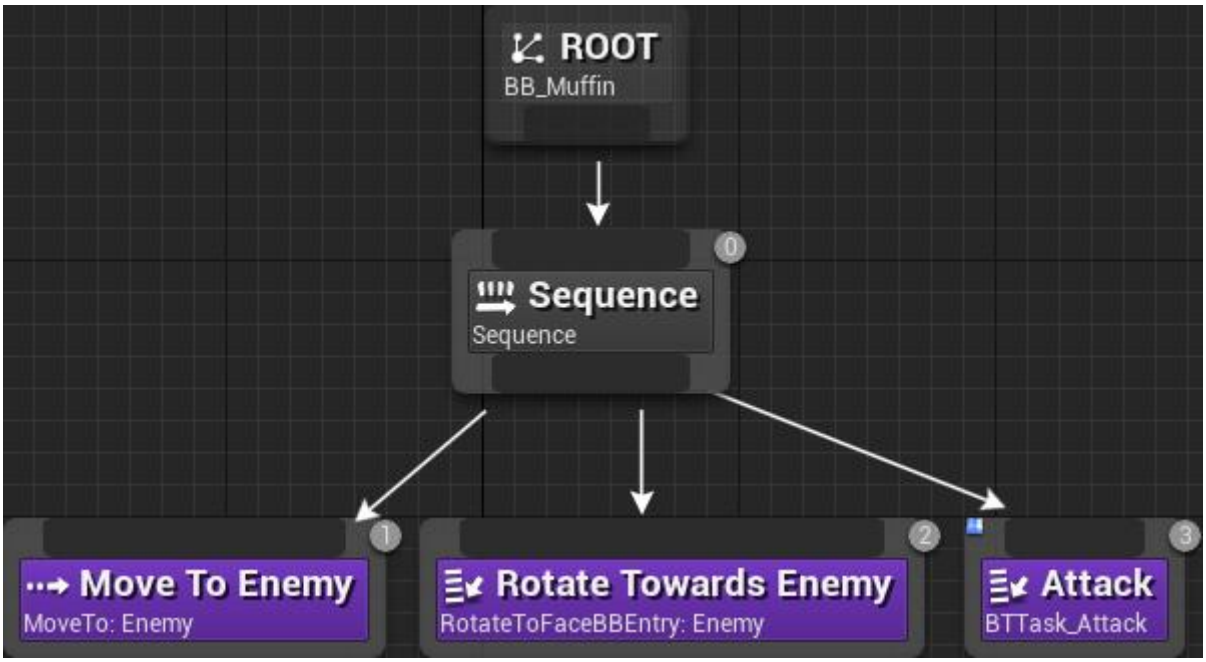

Рисунок 4.3 Дерево поведінки

І надалі створюємо сервіс Служби подібні завданням в тому, що ви використовуєте їх, щоб щось робити. Проте замість того, щоб змусити пішки виконати дію, ви використовуєте сервіси для перевірки або поновлення дошки.

Послуги не є окремими вузлами. Замість цього вони прикріплюються до завдань або композитам. Це призводить до більш організованого дереву поведінки, тому що у вас менше вузлів для роботи.

Вказати ключ, використовуючи його ім'я в вузлі Make Literal Name Вставляємо змінну в дереві поведінки. Це дозволить вам вибрати ключ зі списку. Ви будете використовувати другий метод. Створіть змінну типу Blackboard Key Selector. Назвіть його BlackboardKey і включіть Instance Editable. Це дозволить змінної з'являтися при виборі служби в дереві поведінки. Потім створіть виділені вузли, рисунок 4.4

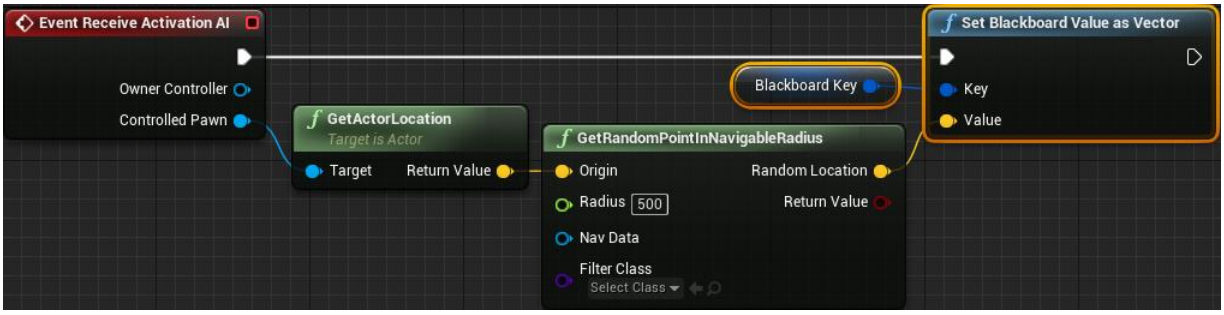

Рисунок 4.4 Вузли

Активація прийому подій штучного інтелекту виконується, коли активується батьківський (у цьому випадку MoveTo). GetRandomPointInNavigableRadius повертає випадкове навігаційне місце в межах 500 одиниць керованого кексу. Встановити значення Blackboard як вектор встановлює значення клавіші дошки (надається BlackboardKey) на випадкове місце. Далі налаштовуємо сприйняття штучного інтелекту Сприйняття штучного інтелекту - це компонент, який можливо додати акторам. Використовуючи його, можливо давати відчуття (наприклад, зір і слух) вашому штучному інтелекту. Відкрийте AIC Muffin та додайте компонент AIPerception. Існує три основні настройки для зору: Радіус видовища: Максимальна відстань, яку може побачити булочка. Залиште це на 3000 . Втратити радіус прицілу: Якщо булочка побачила ворога, це те, наскільки ворог повинен відійти, перш ніж кекс втратить зір. Залиште це на 3500 .Половина кута периферійного зору: наскільки широке бачення кекси. Встановіть це 45 . Це дасть кексу 90-градусний діапазон зору. За замовчуванням AI Perception виявляє лише ворогів (акторів, призначених для іншої команди). Однак актори не мають команди за замовчуванням. Коли у актора немає команди, AI Perception вважає це нейтральним

. На момент написання, не існує способу призначення команд за кресленнями. Натомість можливо просто сказати AI Perpcetion для виявлення нейтральних акторів. Для цього розгорніть програму Detection by Affiliation та увімкніть Detect Neutrals

Далі все це компілюємо, щоб отримати явну картину і бачимо, що у штучного інтелекту з’явився радіус виявлення супротивника як на рисунку 4.5.

Рисунок 4.5 Радіус виявлення

Далі створюємо ворога, відкриваємо BB_Muffin та додайте ключ типу Object. Перейменуйте його на Enemy Зараз не можливо використовувати Enemy в MoveTo. Це тому, що ключ є Об'єктом, але MoveTo приймає лише ключі типу Vector або Actor. Щоб виправити це, виберіть Enemy[2] та розгорніть Тип ключа. Встановіть базовий клас на актора. Це дозволить дереву поведінки розпізнати ворог, як актор. Закрийте BB_Muffin. Тепер потрібно створити поведінку, щоб рухатися до ворога.

Далі налаштовуємо рух до ворога BT_Muffin, а потім від'єднайте послідовність і кореневий. Можливо зробити це за допомогою Alt-натиснувши на провід, що з'єднує їх. Відсуньте піддерево роуму в бік. Далі створіть виділені вузли та встановіть їх клавішу Blackboard на Enemy

Це перемістить пішака до ворога. У деяких випадках пішак не буде повністю звернутись до своєї цілі, тому ви також використовуєте Поворот, щоб зіткнутися з входом BB. Тепер вам потрібно встановити Enemy, коли AI Perception виявить інший кекс, як на рисунку 4.6.

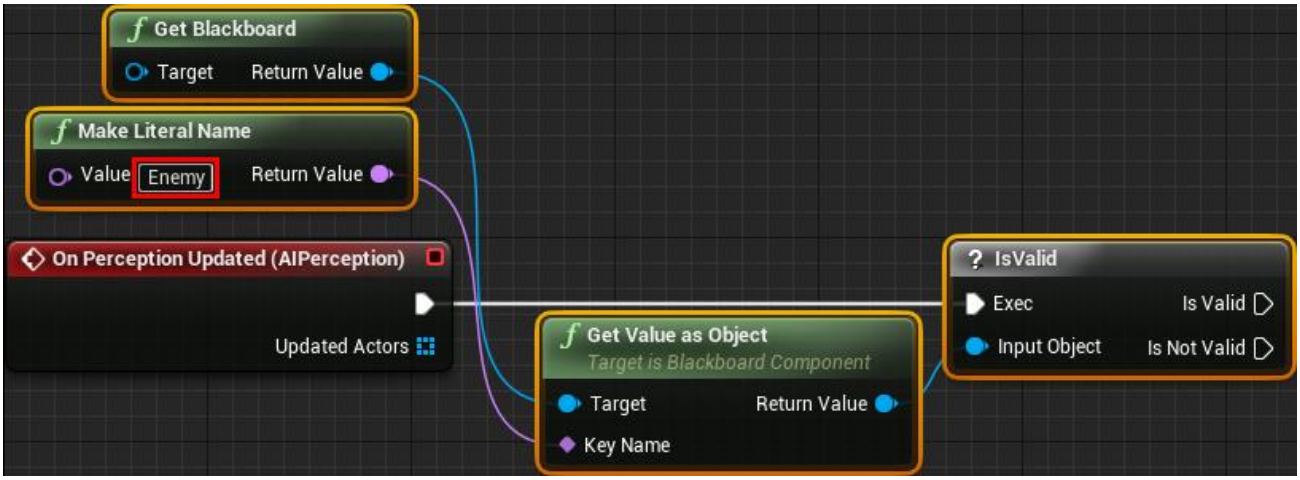

Рисунок 4.6 Код для визначення ворогу

Це дозволить перевірити, чи штучний інтелект вже має ворога. Якщо цього немає, вам потрібно надати його IsValid перевірить, чи встановлено ключ EnemyЯкщо це не встановлено, переведіть петлю на всіх сприймаються в даний час акторів Cast To BP_Muffin перевірить, чи є актор кекси

Якщо це булочка, перевірте, чи вона мертва Якщо IsDead поверне помилкове значення, встановіть кекс як новий Enemy, а потім перервіть цикл[4]. Компылюэмо, а потім закрийте AIC_Muffin. Натисніть кнопку Відтворити, а потім наклейте два кекси, щоб один був перед іншим. Булочка позаду автоматично піде назустріч іншому кексу.

Створюємо завдання атаки штучного інтелекту отримання події Execute Execute буде виконуватися, коли дерево поведінки виконує BTTask_Attack Cast To BP_Muffin перевірить, чи є контрольована пішачка типу BP_Муффін Якщо він є, встановлена його змінна IsAttacking Finish Execute дозволить дереву поведінки знати, що завдання успішно завершено, рисунок 4.7.

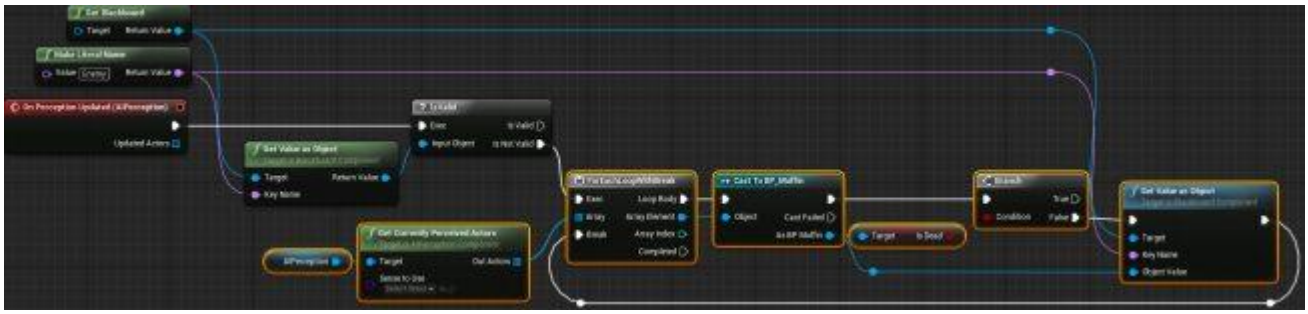

Рисунок 4.7 код перевірки ключа та дії

Тепер вам потрібно додати BTTask_Attack до дерева поведінки Wait до кінця послідовності. Встановіть його час очікування на 2. Це дозволить переконатися, що кекс не буде постійно атакувати. Поверніться до головного редактора і натисніть Play. Ікру два кекси, як минулого разу. Булочка рухатиметься і обертається до ворога. Після цього він нападе і чекає дві секунди. Потім він знову виконає всю послідовність, якщо побачить іншого ворога, рисунок 4.8.

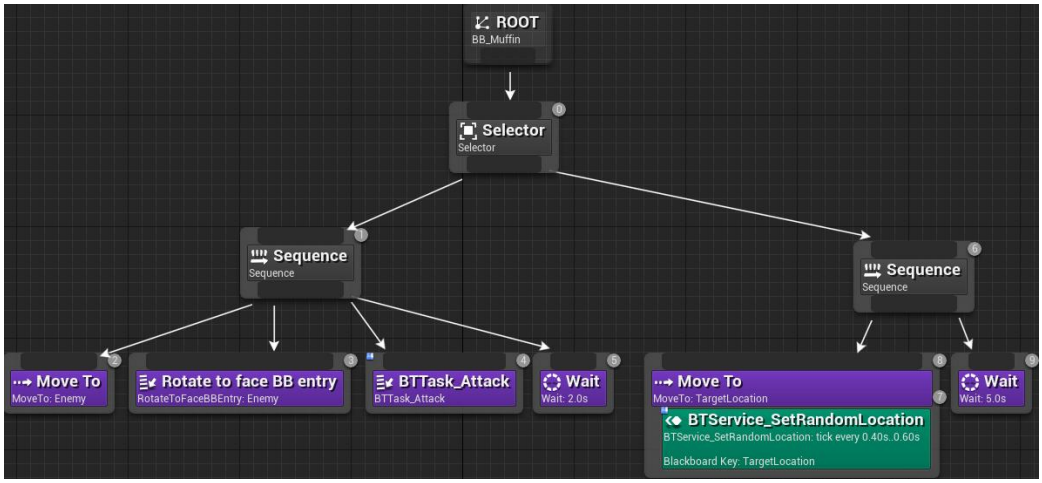

Рисунок 4.8 Дерево поведінки з налаштуванням

Для комбінування підрядів використовуємо композитор Selector. Як і послідовності, вони також виконуються зліва направо. Однак Селектор зупиниться, коли дитина досягне успіху, а не зазнає невдачі[7]. Використовуючи цю поведінку, можливо переконатися, що дерево поведінки виконує лише одне під дерево. Відкрийте BT_Muffin, а потім створіть Selector після вузла Root. Це налаштування дозволить одночасно запускати лише одне під дерево, Як приведено на рисунку 4.9,

так буде працювати кожне піддерево: Атака: Селектор спочатку запустить під дірець атаки.

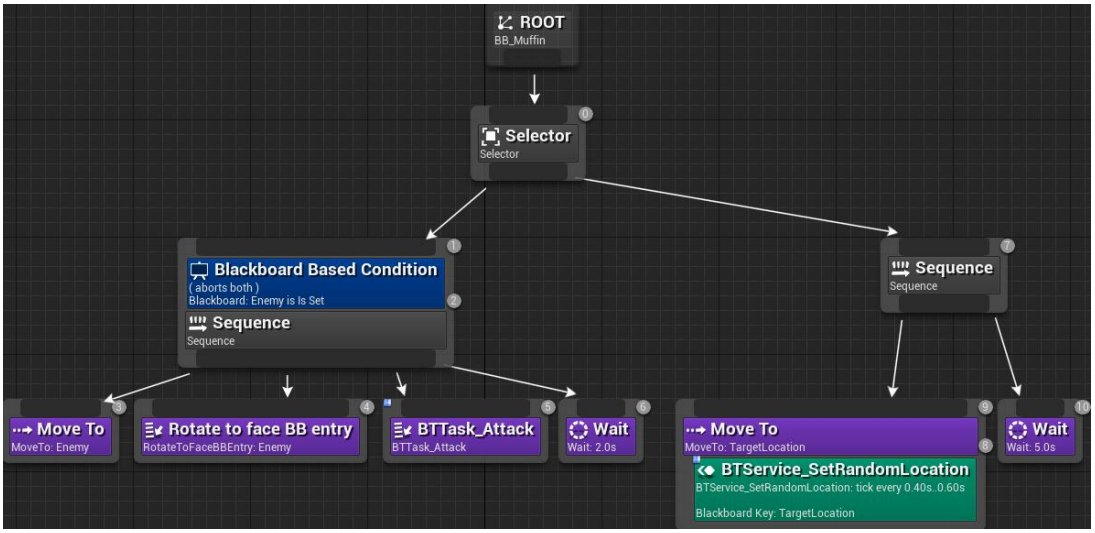

Рисунок 4.9 Кінцеве дерево поведінки

Якщо всі завдання вдаються, Послідовність також буде успішною. Selector виявить це і потім припинити виконання. Це не дозволить робряти підгрупу роуму. Роум: Селектор спробує запустити підрівень атаки першим. Якщо Enemy не встановлено, MoveTo вийде з ладу. Це також призведе до виходу з ладу послідовності . Оскільки атаки виявився невдалим, Selector виконає наступне дочірнє зображення – під дерево роуму. У традиційних деревах поведінки виконання починається з кореня кожного оновлення. Це означає, що кожне оновлення, воно спробує спочатку під дерево атаки, а потім під дерево. Це означає, що дерево поведінки може миттєво змінювати підкреслення, якщо змінюється значення Enemy.Однак дерева поведінки Unreal працюють не так. У Unreal виконання виконується з останнього виконаного вузла. Оскільки AI Perception не відчуває інших акторів негайно, під дерево починає працювати. Дерево поведінки тепер має зачекати, поки під робота закінчиться, перш ніж він зможе переоцінити атаку. правило, ви використовуєте декоратори для здійснення перевірок.[8]. Якщо результат правдивий, декоратор також поверне справжнє і навпаки. Використовуючи це, можливо контролювати, чи може виконати батько декоратора.

Декоратори також мають можливість перервати під дерево. Це означає, можливо зупинити під дерево після встановлення Enemy . Це дозволить кекси напасти на ворога, як тільки його виявлять. Для використання абортів можливо використовувати декоратор Blackboard . Вони просто перевіряють, чи не встановлена клавіша для дошки[9]. Відкрийте BT_Muffin, а потім клацніть правою кнопкою миші на підпорядкуванні послідовності атаки. Виберіть Додати декоратор дошку . Це додасть декоратор дошки до послідовності.

Аборти спостереження відміняють під дерево, якщо вибраний клавіатурний ключ змінився. Існує два типи абортів:

Самостійно: Цей параметр дозволить субкредиту атаки перервати себе, коли Enemy стане недійсним. Це може статися, якщо Ворог загине до завершення підзахідного атак.

Нижній пріоритет: Цей параметр призведе до переривання дерев нижчого пріоритету, коли встановлено Enemy . Оскільки піддерево роуму знаходиться після атаки, воно є нижчим пріоритетним.

Короткий підсумок атаки: Селектор буде запускати атаки поддерева , якщо ворог знаходиться безліч[10]. Якщо він встановлений, Пішак рухатиметься та обертається до ворога Після цього він здійснить атаку Нарешті, Пішак чекатиме дві секунди Короткий підсумок підрозвитку роуму: Selector запустить піддерево, якщо підстворення атаки не вдасться. У цьому випадку він вийде з ладу, якщо Враг не встановлений. BTService_SetRandomLocation створить випадкове місцеположення Пішак переміститься в створене місце Після цього буде чекати п'ять секунд.

5 ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ ТА МАТЕМАТИКА ШТУЧНОГО ІНТЕЛЕКТУ

5.1 Математика штучного інтелекту

У грі мета комп'ютера на боці супротивника є знищення гравця. Тому потрібно нанести йому максимальну можливу шкоду. У кожного супротивника є кількість одиниць шкоди, який він наносить замку, у тому разі якщо дійде до гравця. Не завжди важливо, щоб тільки один взятий супротивник дістався гравця, бо він може і допомогти іншим супротивникам дістатися гравця, тобто супротивники залежать від інших супротивників.

Таким чином, всі супротивники що знаходяться у даний момент на полі для кожного з потенційних шляхів. З отриманих значень обираємо найближчий шлях. Програмна система повинна розрахувати потенційна шкода, яка може бути завданий гравцю, у разі запуску супротивника на поле[12]. Задача вибору супротивника може бути вирішена за допомогою методу лінійних адитивних згорток.

Лінійна адитивна згортка з ваговими коефіцієнтами обирає найкращий варіант, що обчислюється по формулі:

$$Z = \sum_{j=1}^{n} \alpha_j \beta_j a_{ij}$$

де α_j - нормуючі множники,

β_j - вагові коефіцієнти, що залежать від відносного вкладу всіх критеріїв в загальний критерій,

a_{ij} - значення оцінки певного критерію для даного варіанту. При цьому, вагові коефіцієнти прийнято вказувати вже нормованими величинами.

Цей метод дає можливість обирати варіанти на основі найкращого вибору шляху.

Серед методів прийняття рішень на базі теорії дослідження операцій потрібно відзначити задачі лінійного та нелінійного програмування. [16].Математичне програмування використовується в задачах, де можливо

побудувати однозначну математичну модель та існує один критерій оптимізації і набором обмежень на наведені змінні

Зазначимо, що методи лінійного програмування застосовуються, якщо поставлена одна ціль. Коли ціль не одна, використовується цільове програмування. Якщо задача вирішується не в один етап чи з інтервалом[13], слід використати метод динамічного програмування.

Можливо поставити задачу таким чином: знайти розподіл ресурсів за супротивниками, при яким сумарна кількість супротивників буде мінімальна, а шкода була максимальною.

Таким чином, в оптимізаційній моделі використовуються:

– Cij - витрати або прибуток, щодо виділення однієї одиниці ресурсу до роботи;

– невідомі Xij, як правило це об’єм і-го ресурсу, необхідного для виконання j-ї роботи.

Загальні витрати, чи прибуток, на використання і-го ресурсу для виконання j-ї роботи обчислюється як Xij*Cij. Маємо маємо лінійну розподільну задачу, цільова функція:

$$F = \sum_{j=1}^{n}\sum_{i=1}^{m} C_{ij} * X_{ij} \rightarrow extremum$$

Система обмежень задається таким чином:

$$\sum_{j=1}^{n} X_{ij} \leq x_i, \quad \forall i = \overline{1,m}$$

,

$$\sum_{i=1}^{m} X_{ij} \geq y_j, \quad \forall j = \overline{1,n}$$

$$X_{ij} \geq 0$$

Важливо зазначити, що на практиці до типового застосування методів лінійного програмування належать сиплексний та транспортний методи, угорський алгоритм, жадібний алгоритм.

Для моделювання штучного інтелекту взята поведінка комп’ютера на боці супротивників з метою знищення гравця. Для знищення, штучному інтелекту необхідно нанести максимальну шкоду гравцю[14].

Таким чином, для усіх супротивників, що можуть знаходитись у даний момент на полі потрібно розрахувати шкоду для кожного з потенційних шляхів, при цьому враховуючи їх кількість. З отриманих значень обираємо найбільше. Штучний інтелект потрібен розрахувати потенційну шкоду, що може бути завданий гравцю у разі запуску супротивника на поле.

Для подальшого моделювання штучного інтелекту введемо такі математичні позначення для ігрових сутностей:

– $\{U_i\}_{i=1}^{m}$ – множина супротивників U_i, що доступні для запуску на поле в грі, всього доступно m супротивників;

– $P = \{P_i\}_{i=1}^{n}$ – множина шляхів P_i на ігровому полі;

– $T = \{T_i\}_{i=1}^{m}$ – список усіх ресурсів, що доступні на поле в грі, всього доступно m в ресурсів;

– $P^{*} = \{P^{*}_{(x,y)}\}_{x=1,y=1}^{X,Y}$ – множина клітин на ігровому полі X×Y.

– X – ширина ігрового поля;

– Y – довжина ігрового поля;

– C_i – ціна і-го супротивника у внутрішніх ресурсах;

– Q_i – сила атаки супротивника і;

– H_i – кількість одиниць життя супротивника і;

– V_i – сила атаки гравця і;

– M_M – кількість ресурсів гравця за монстрів; – M_T – кількість ресурсів гравця.

Задача супротивника та його кращого шляху на полі може бути промодельована за допомогою теорії корисності, а саме моделі лінійної адитивної згортки, де α_j - нормуючі множники[15], β_j - вагові коефіцієнти, що залежать від відносного вкладу всіх критеріїв в загальний критерій,

$$Z = \sum_{j=1}^{n} \alpha_j \beta_j K_j(P_i)$$

$K_j(P_i)$ – критеріальна оцінки за критерієм K_j для варіанту шляху P_i.

При цьому, вагові коефіцієнти[17]. прийнято вказувати вже нормованими величинами

Задача призначення супротивника на той чи інший шлях може бути віднесена до оптимізаційних задач про призначення. Вона може бути сформульована наступним чином: розподілити наявні на ігровому полі супротивники на шляхи таким чином, щоб мінімізувати пошкодження, яке понесуть супротивники під час проходження цими шляхами. Табличне надання цієї оптимізаційної задачі надано в таблиці

Даний метод дозволяє найбільш ефективно обирати варіанти на основі багатьох критеріїв, тому саме цей метод був розглянутий для застосування в ігровій системі.

Для вирішення задачі призначення супротивника до певного шляху в розробленій моделі (1) було запропоновано наступні критерії з ваговими коефіцієнтами:

– K_1 – загальна кількість захисних сутичок героя, де $\beta_1 = 0{,}1$;

– K_2 – безпека шляху: кількість одиниць втрат, який нанесуть нападнику на шляху, де $\beta_2 = 0{,}5$;

– K_3 – кількість ресурсів, які може отримати супротивник, пройшовши по шляху або його частині, де $\beta_3 = 0{,}3$;

– K_4 – відношення на шляху з особливими ефектами при атаці до загальної кількості героїв на шляху, де $\beta_4 = 0{,}1$.

Такі обчислення при виборі супротивника, якого потрібно додати на ігрове поле, потрібно виконувати для кожного супротивника, та обрати найкращій варіант.

Таблиця 5.1 – Табличне надання оптимізаційної задача призначення супротивника на шлях

Супротивники	Шляхи				
	P_1	P_2	…	P_n	Обмеження за супротивниками Ui
U_1 (C_1)	D11, X11	D12, X12	…	D1n, X1n	1
U2 (C2)	D21, X21	D22, X22	…	D2n, X2n	1
…	…	…	…	…	
Um (Cm)	Dm1, Xm1	Dm2, Xm2	…	Dmn, Xmn	1
Обмеження за шляхами P_j	1	1		1	

Цей метод дає можливість ефективно обрати варіанти на основі багатьох критеріїв, для застосування в даній системі.

Математичне програмування - широко поширений метод оптимізації[16] використання обмежених ресурсів.

Для моделювання задачі про призначення супротивників на шляхи введемо наступні позначення:

– $U = \{U_i\}_{i=1}^{m}$ – множина супротивників Ui, що знаходяться на початку у грі ;

– Ci – кількість і-го супротивнику у максимально допустимому значенні;

– $P = \{P_i\}_{i=1}^{n}$ – список усіх шляхів на ігровому полі;

– Dij – шкода, яка завдана гравцю, у випадку, якщо супротивник *i* буде запущений на ігрове поле *j*.

Задача призначення супротивника на той чи інший шлях може бути віднесена до оптимізаційних задач про призначення. Вона може бути сформульована наступним чином: розподілити наявні на ігровому полі

супротивники на шляхи таким чином, щоб мінімізувати шкоду, яку понесуть супротивники під час проходження цими шляхами. Табличне надання цієї оптимізаційної задачі надано в таблиці 1.1.

Введемо змінну *Xij* , що символізує, чи був запущений *i*-й супротивник по *j*-му шляху, змінна може приймати значення 1 або 0.

Потенційна шкода *Dij* розраховується за формулою:

$$D_{ij} = Q_i L_{ij} + \sum_{u=0}^{U} Q_u L_{uij}$$

де Q_i – сила атаки супротивника i,

L_{ij} – факт того, дійде супротивник до гравця j, чи ні, може приймати значення 0 чи 1:

$$L_{ij} = \begin{cases} 0, H_i > T_{ij} \\ 1, H_i \leq T_{ij} \end{cases}$$

де H_i – кількість одиниць життя супротивника *i*,

T_{ij} – кількість шкоди, що буде завдано супротивнику *i* на шляху *j*.

Для даної постановки задачі повинні виконуватися наступні умови:

– кількість *i*-го супротивника повинна бути менша або дорівнювати кількості кількісті життя у гравця $Ci < MM$, де *MM* – кількість життя, що є в наявності в гравця за супротивника;

– кожен супротивник може бути призначений не більше ніж на один шлях, отже

$$\sum_{j=1}^{n} x_{ij} \leq 1, \ \forall_i = \overline{1, m};$$

– на кожен шлях може бути призначений лише один супротивник, отже

$$\sum_{i=1}^{m} x_{ij} = 1, \ \forall_j = \overline{1, n};$$

– i-й супротивник може бути призначений або не призначений на j шлях, отже $X_{ij} = 1|0, \forall i = \overline{1,m}, \ \forall j = \overline{1,n}$.

Цільова функція повинна забезпечити таке призначення супротивників за шляхами, при якому би досягався максимальний потенційна шкода гравцю супротивника, що грає за супротивників, отже

$$F = \sum_{i=1}^{m}\sum_{j=1}^{n}\frac{D_{ij}X_{ij}}{C_i} \to \max,$$

$$F = \sum_{i=1}^{m}\sum_{j=1}^{n}\frac{X_{ij}(Q_i L_{ij} + \sum_{u=0}^{U} Q_u L_{uij})}{C_i} \to \max.$$

Таким чином, математична модель оптимізаційної задачі про призначення штучним інтелектом супротивників на шляхи приймає наступний вигляд:

$$F = \sum_{i=1}^{m}\sum_{j=1}^{n}\frac{X_{ij}(Q_i L_{ij} + \sum_{u=0}^{U} Q_u L_{uij})}{C_i} \to \max,$$

$$\sum_{j=1}^{n} x_{ij} \le 1,\ \forall_i = \overline{1,m};$$

$$\sum_{i=1}^{m} x_{ij} = 1,\ \forall_j = \overline{1,n};$$

$$X_{ij} = 1|0, \forall i = \overline{1,m}, \forall j = \overline{1,n}.$$

Введемо змінну $B_i(x,y)$, що символізує гравця на полі *i*-та його позиція *(x,y)*, $B_i(x, y)$ може приймати значення 1 або 0.

Таким чином, цільова функція може бути виражена як:

$$F = \sum_{i=1}^{m}\sum_{x=1}^{X}\sum_{y=1}^{Y}\frac{(A_{i(x,y)} + 1)(K_{i(x,y)} + 1)D_{i(x,y)}X_{i(x,y)}}{C_i} \to \max.$$

Отже, математична модель оптимізаційної задачі про призначення штучного інтелекту супротивників на позиції на ігровій мапі приймає наступний вигляд:

$$F = \sum_{i=1}^{m}\sum_{x=1}^{X}\sum_{y=1}^{Y}\frac{(A_{i(x,y)} + 1)(K_{i(x,y)} + 1)D_{i(x,y)}X_{i(x,y)}}{C_i} \to \max$$

$$\sum_{x=1}^{X}\sum_{y=1}^{Y} b_{ixy} \le 1,\ \forall_x = \overline{1,X}, \forall_y = \overline{1,Y},$$

$$\sum_{i=1}^{m} b_{ixy} = 1,\ \forall_j = \overline{1,n},$$

$$B_{ij} = 1|0, \forall i = \overline{1,m}, \forall_x = \overline{1,X}, \forall_y = \overline{1,Y}$$

Для практичної реалізації математичних моделей потрібно розробити алгоритми вирішення задачі про призначення[18] з використанням методів, що були досліджені.

5.2 Розробка алгоритмів

В практиці операційного менеджменту типовими для вирішення задач лінійного програмування є симплексний та транспортний методи, жадібний алгоритм, угорський алгоритм та інші. Симплекс-метод – можливо вважати базовим для вирішення цього класу задач. Це алгебраїчна процедура, в результаті якої дослідник послідовно наближається до знаходження оптимального рішення. В теорії даний метод можливо використовувати для вирішення задач, які включають будь-яку кількість обмежень і змінних, але якщо в них, більше чотирьох змінних чи обмежень, то обчислення стає досить складним.

На рисунку 5.1 наведено схему розробленого алгоритму для знаходження супротивнику який буде введений на поле.

Жадібний алгоритм – евристичний алгоритм, досить простий для застосування, тому і був обраний. При тому він найефективніший для використання штучного інтелекту у цій грі, адже він не потребує багато ресурсів комп’ютера гравця, що є великим показником для оптимізації.

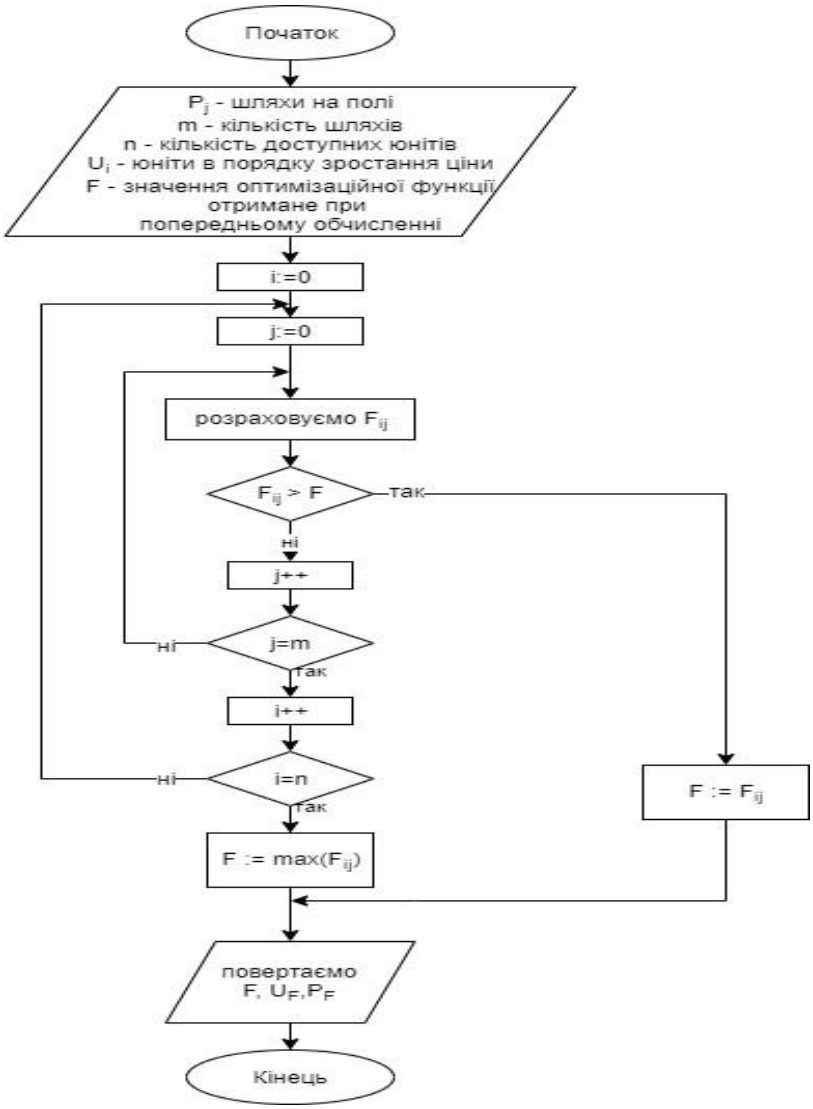

Рисунок 5.1 – Алгоритм призначення супротивників на шляхи

Для вирішення задачі для призначення супротивників та задачі по призначення гравця було обрано, розроблено, та реалізовано для різних алгоритми на основі симплекс методу і жадібного алгоритму.

5.3 Реалізація програмної частини

Для реалізації програмної частини було обрано Unreal Engine 4 – двигун для розробки ігор і віртуальних світів.

Під час реалізації гри було обрано технологію Blueprints, що використовує мову програмування C++, завдяки цьому програмується гра за допомогою схем які містять частини коду C++.

Нижче на рисунку 5.2 представлений фрагмент коду, що демонструє метод завдяки якому з’являються супротивники та корисні ресурси на ігровому полі.

Ігровий клієнт працює за таким чином: На початку битви гравець завантажує дані поля битви[11]. Далі, коли гравець змінює стан битви, тобто з’являється на полі гри відправляється команда до появлення супротивників. Таким чином реалізована синхронізація.

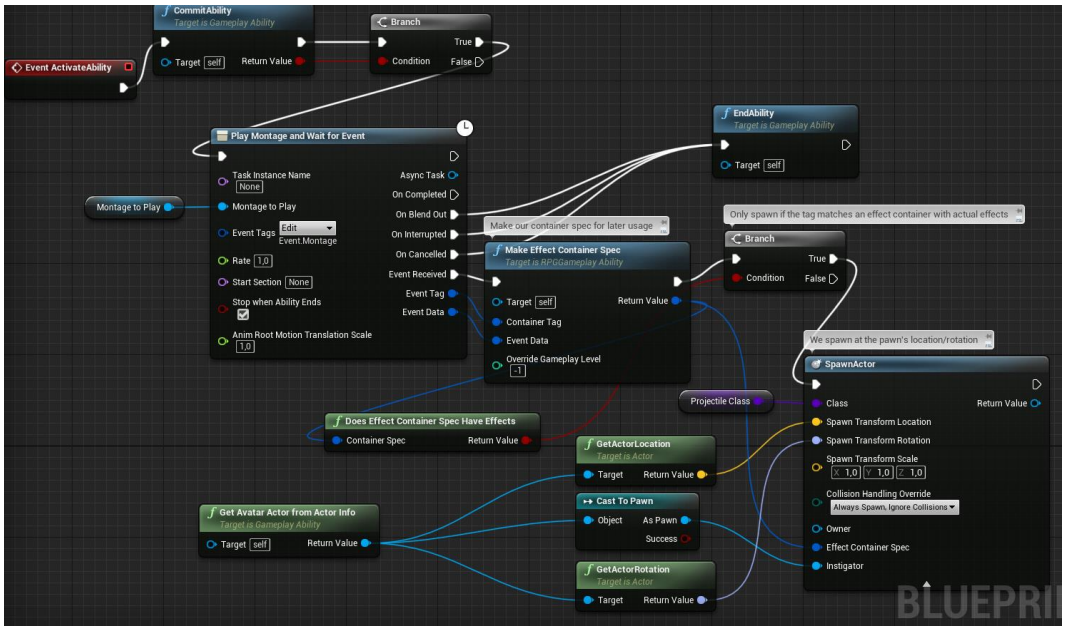

Рисунок 5.2 – Спавн супротивників та ресурсів

На рисунку 5.3 зображено форму геймплею де зображені супротивники, корисні ресурси, моделі на поли та всі основні елементи керування

Знизу екрану зображені інформаційні панелі, які показують поточний рівень здоров’я гравця, активні навички гравця, та ресурси для поповнення здоров’я. Вгорі розташований таймер при якому гравець повинен встигнути знищити супротивника, або буде кінець гри.

Рисунок 5.3 – Геймплей

Далі можливо побачити нижче на рисунку 5.4 всі елементи розробки гри, завдяки яким і була розроблена сама гра та всі її елементи.

Можемо бачимо інтерфейс розробки, та елементи які знаходяться на самому полі гри, а саме: внизу місце, де з’являється гравець при старті гри, 3 точки де з’являються супротивники, та всі інші елементи які відповідають за все що знаходиться на полі гри.

Рисунок 5.4 – Інтерфейс розробки гри

Приведено далі фрагмент коду логіки, в якому обирається кращий для переміщення супротивника. Вибір виконується на основі розрахунку кращого шляху на основі багатьох критеріїв.

```
public int GetOptimalPath(Field field,
Unit unit){  var paths =
field.StaticData.Path;
 var tableToAnalize = new double[paths.Length,
_additiveConvolutionCalculator.NumberOfCriterias];  for (int i = 0; i <
paths.Length; i++){
   var heroesOnPath = FindHeroesThatCanAttackPath(field,
i);    var heroes = heroesOnPath.Select(t =>
field[t]).ToArray();    var heroesTypes = heroes.Select(t
=> t.Type).ToArray();    tableToAnalize[i, 1] =
GetTotalAttackDamage(heroesTypes, unit);
tableToAnalize[i, 2] = GetAvgHeroesRemoteness(paths[i],
heroes);    tableToAnalize[i, 3] =
GetHeroesWithSpecialEffectRate(heroesTypes);
 }
 return _additiveConvolutionCalculator.FindOptimalVariantIndex(tableToAnalize);
}

private double GetTotalAttackDamage(IEnumerable<GameObjectType> heroTypes, Unit
unit){  return 1 - (double) heroTypes.Sum(t
=>_gameCalculator.CalculateDamage(unit.Type, t))/unit.Health;
}
```

```
        private double GetAvgHeroesRemoteness(Path path, ICollection<GameObject>
        heroes){  double remotenessSum = 0;  foreach (var heror in heroes){
        var pos = path.First(point => _gameCalculator.IsHeroCanAttack(hero,
        point));    remotenessSum += (double)path.PointOnThePathPosition(pos) /
        path.Length;
         }

         return remotenessSum / heroes.Count;
        }

        private HashSet<int> FindHeroesThatCanAttackPath(Field field,
        int pathId){  var path = field.StaticData.Path[pathId];  var
        heroes = new HashSet<int>();  foreach (var point in path)
           heroes.UnionWith(field.FindHeroesThatCanAttack(point,
        _statsLib));   return heroes;
```

Частина коду жадібного алгоритму для вирішення задачі про призначення ігрових об’єктів на поле:

```
public (GameObjectType type, TDest destination)?
GetOptimalVariant(Func<GameObjectType, TDest, double> func)
{
  if (!_availableGameObjects.Any() || !_availableDestinations.Any())
  {
    return default;
  }
  var maxWeight = 0.0;
  var variants = new List<(GameObjectType type, TDest dest)>();   foreach (var
gameObject in _availableGameObjects)
  {
    _availableDestinations.AsParallel().ForAll(dest =>
    {
      var weight = func(gameObject, dest);
      if (Math.Abs(weight - maxWeight) < double.Epsilon)
      {
        variants.Add((gameObject, dest));
      }
      if (weight > maxWeight)
      {
        variants.Clear();
        variants.Add((gameObject, dest));
maxWeight = weight;
      }
    });
```

5.4 Планування експериментів над штучним інтелектом

Для того щоб створити ефективний розвинутий штучний інтелект, було прийнято рішення провести дослідження за допомогою експерименту над штучним інтелектом. Для планування експериментів використовувалися наступні етапи:

- Стандартний Штучний інтелект не розглядався, його дослідження було проведено окремо, хоча він і є не дуже вимогливим до обчислюваних ресурсів комп'ютера, але він являється не дуже інтерактивним і не дає необхідного рівня перемог над гравцем.
- в якості сильного штучного інтелекту розглядалася поведінка штучного інтелекту на базі розроблено багатокритеріальної моделі вибору
- в якості розвинутого штучного інтелекту розглядалася поведінка штучного інтелекту на базі моделей оптимізаційних задач про призначення

Заплановано 2 серії експериментів.

Під час цих першої серії необхідно дослідити наступне:

– призначення супротивників на поле шляхом багатокритеріального вибору на базі лінійної адитивної згортки;

– призначення супротивників на шляхи на основі жадібного алгоритму;

– призначення супротивників на шляхи на основі симплекс-методу підрахунку значень оптимізаційної моделі.

Під час цих другої серії експериментів необхідно дослідити наступне:

– призначення героя на клітини шляхом багатокритеріального вибору на базі лінійної адитивної згортки;

– призначення героя на клітини на основі жадібного алгоритму; призначення героя на шляхи на основі симплекс-методу.

Експерименти необхідно проводити на іграх різною розмірності:

– Ігра 1: кількість супротивників - 5, розмір поля 10X10, кількість гравців 3, кількість шляхів – 2;

– Ігра 2: кількість супротивників - 8, розмір поля 20X20, кількість гравців, кількість шляхів – 5;

– Ігра 3: кількість супротивників - 10, розмір поля 50X50, кількість героїв 7, кількість шляхів – 10.

Для кожної серії експериментів було заплановано по 100 ігрових сесій, у яких ефективність обраних методів перевірялася за допомогою наступних показників:

– процентне співвідношення успіхів штучного інтелекту по відношенню до іншої сторони;

– використання ресурсів пристрою, а саме час у Мілі секундах, затрачуваний для прийняття одного рішення; – кількість використаної пам’яті.

Усі дослідження необхідно проводити на одному фізичному сервері, який має наступні характеристики:

– центральний процесор Intel Core i3;

– частота процесору 3,4 Ghz;

– кеш 8 Mb;

– 4 ядра (8 логічних ядер); – ОЗУ 16 Gb 2666 MHz.

Щоб забеспечити кращу серію, досліди відбувались автоматично без участі людини, тобто штучний інтелект змагався сам з собою на базі розроблених моделей.

В серії 100 ігор в якості ворога для штучного інтелекту в рівних пропорціях використовувалися всі 3 розроблені моделі. Це дало змогу змоделювати досить досвітченого гравця.

Результати дослідження першої серії експериментів для «Ігра 1», «Ігра 2» та «Ігра 3» наведено в таблицях.

Ігра 1: кількість супротивників - 5, розмір поля 10X10, кількість гравців 3, кількість шляхів –2, як приведено на таблиці 5.2

Таблиця 5.2 - Результати першого експерименту для «Ігра 1»

Алгоритм/Показники	Середній час однієї роботи алгоритму, мс	Середне використання оперативної пам’яті, Кб	Процент перемог монстрів, %
АЗ 1	8	311	33
БА 1	71	1090	37
ЖА 1	50	845	36

Ігра 2: кількість супротивників - 8, розмір поля 20Х20, кількість гравців, кількість шляхів – 5, як приведено в таблиці 5.3.

Таблиця 5.3 - Результати першого експерименту для «Ігра 2»

Алгоритм/Показники	Середній час однієї роботи алгоритму, мс	Середне використання оперативної пам’яті, Кб	Процент перемог монстрів, %
АЗ 1	31	414	31
БА 1	185	1487	36
ЖА 1	106	1045	35

Ігра 3: кількість супротивників - 10, розмір поля 50Х50, кількість героїв 7, кількість шляхів – 10, як на таблиці 5.4.

Тепер зробимо висновок, що результати достатньо закономірні та для ігор різної розмірності модифікований жадібний алгоритм є найбільш оптимальним, адже він дає значну перевагу у продуктивності, та майже не програє у ефективності у порівнянні із базовим.

Таблиця 5.4 - Результати першого експерименту для «Ігра 3»

Алгоритм/Показники	Середній час однієї роботи алгоритму, мс	Середне використання оперативної пам’яті, Кб	Процент перемог монстрів, %
АЗ 1	172	429	39
БА 1	556	1899	35
ЖА 1	468	1465	34

При цьому метод лінійних адитивних згорток хоч і використовує значно менше ресурсів, він не дає достатнього рівня ефективності.

Результати дослідження другої серії експериментів для «Ігра 1», «Ігра 2» та «Ігра 3».

Другий експеримент для «Ігра 1», наведений у таблиці 5.5

Таблиця 5.5 - Результати другого експерименту для «Ігра 1»

Алгоритм/Показники	Середній час однієї роботи алгоритму, мс	Середне використання оперативної пам’яті, Кб	Процент перемог героїв, %
БА 2	799	6000	41
ЖА 2	480	4567	42
АЗ2	43	877	36

Другий експеримент для «Ігра 2», наведений у таблиці 5.6.

Таблиця 5.6 - Результати другого експерименту для «Ігра 2»

Алгоритм/Показники	Середній час однієї роботи алгоритму, мс	Середнє використання оперативної пам’яті, Кб	Процент виграшів героїв,
БА 2	1469	7318	44
ЖА 2	1850	5460	44
АЗ 2	760	1381	32

Другий експеримент для «Ігра 3», наведений у таблиці 5.7.

Таблиця 5.7 - Результати другого експерименту для «Ігра 3»

Алгоритм/Показники	Середній час однієї роботи алгоритму, мс	Середнє використання оперативної пам’яті, Кб	Процент виграшів героїв
БА 2	3021	8110	45
ЖА 2	2995	6110	44
АЗ 2	1070	1801	30

Для оцінки похибки вимірювань були обчислені середньоквадратичні відхилення. Для нормального закону розподілу оцінка генерального середньоквадратичного відхилення (S) результатів спостережень визначається:

$$S = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{X})}$$

де n – кількість експериментів для кожного окремого випадку, тобто 100 в нашому випадку,

x_i – результат і-го експерименту, $\bar{X}$

– середнє арифметичне.

Значення середньоквадратичного відхилення для показнику середнього часу однієї роботи алгоритму для всіх серій експериментів склало не більше 70 мс, при цьому чим більший показник, тим більше його середньоквадратичне відхилення, тобто цей показник є достатньо нестабільним та залежить від багатьох факторів у різні моменти часу.

Значення середньоквадратичного відхилення для показнику середнього використання оперативної пам’яті програмою для всіх серій експериментів склало не більше 60 Кб, тобто цей показник є остатньо стабільним, але тут потрібно відзначити, що оперативна пам’ять використовується програмою не тільки для досліджуваних обчислень.

Значення середньоквадратичного відхилення для показнику проценту перемог не перебільшує 0,5 %, тобто цей показник є достатньо ефективним.

За результатами проведених експериментів можливо зробити висновок, що результати достатньо закономірні. Для ігор різної розмірності модифікований жадібний алгоритм є найбільш ефективним, адже він дає значний вигріш у продуктивності порівняно зі згортковим та майже не програє у ефективності у порівнянні із базовим.

При цьому метод лінійних адитивних згорток хоч і використовує значно менше ресурсів, але не дає достатнього рівня ефективності. Модель, побудована для поведінки штучного інтелекту на стороні гравця, показала більш значні обчислювальні витрати у зв’язку з її складністю, що зумовлено значним ігровим навантаженням.

Звичайно, відсутність реальної людини призвела до певного зниження процентного співвідношення перемог штучного інтелекту по відношенню до такого більш сильного супротивника, але в сукупності дозволило отримати цілком адекватні результати.

ВИСНОВКИ

Метою було дослідити існуючі методи реалізації штучного інтелекту в іграх.

Проаналізувавши створення штучного інтелекту і програмування його для ігор. Було отриману всю інформацію про поводження його у іграх.

В процесі аналізу було визначено та порівняно декілька жанрів ігор для аналізу цієї області.

Основна задача штучного інтелекту це імітація поведінки персонажа яка була б прийнята гравцем як реальна. І це є не нездійсненою задачею якщо придержуватися деяких основних правил і логікою пошуку шляху на якому і працює сам штучний інтелект який є притаманний різним жанрам ігор. Саме основне це оптимізування штучного інтелекту для комп'ютерів різних потужностей.

Отже, домоглися автономності всіх окремих підсистем штучного інтелекту і надали їм можливість оптимізуватися до різних систем

Найголовніше це оптимізування штучного інтелекту на ігровому двигуні так щоби не було разсинхронності коли обчислюється один етап, а анімація не встигає за діями штучного інтелекту і штучний інтелект не був зламаним, та через те не стояв на місці.

Основний цикл ігрового движка займається двома класами дій: отрисовка дії і оновлення її. Якщо все це послідовно програмувати, то це не зламає штучний інтелект, але якщо є паралельне обчислення то це є складнішим для обробки даних.

Основна дія при русі штучного інтелекту щоб не було таке що дія відбувається, але не сама анімація. Через те буває порушення анімації і відбувається хаос, коли рух відбувається не плавно.

Тому потрібно синхронізувати дії штучного інтелекту і анімацію, щоб не було розсинхронізації.

Таким і робота усього штучного інтелекту полягає у синхронізації дій та оптимізації роботи у різних станах і найголовніше щоб кожний гравець отримував

якомога більше ігрового досвіду і відчував повне поринання у ігровий світ, бо в багатьох іграх штучний інтелект є найважнішою складовою цього самого ігрового досвіду.

ПЕРЕЛІК ПОСИЛАНЬ

1. Brian Schwab. AI Game Engine Programming. — Charles River Media, 2004. p. 624

2. Джон Дэвид Фунге AI для игр и анимации: подход когнитивного моделирования М. 1999 С 30-74.

3. Muska & Lipman. Buckland AI Techniques for Game Programming. Muska & Lipman 2002 p. 94-108.

4. New Rider Champandard AI Game Development. 2003 p 134-156.

5. Bourg; Seemann AI for Game Developers. O'Reilly & Associates 2004 p. 238-241

6. Bob Scott. The Illusion of Intelligence AI Game Programming Wisdom editor Steve Rabin. — Charles River Media, 2002 p 13-97.

7. Pamela McCorduck. Machines Who Think.- 2nd edition - Natick, MA: A. K. Peters, Ltd., 2004 p. 118-158.

8. Funge. AI for Animation and Games: A Cognitive Modeling Approach. A K Peters 1999 p. 41-49.

9. Funge. Artificial Intelligence for Computer Games: An Introduction. A K Peters 2004. – p 21-37.

10. Millington. Artificial Intelligence for Games. Morgan Kaufman. 2005. p 56-83.

11. Schwab. AI Game Engine Programming. Charles River Media. 2004. p 9-68.

12. Smed and Hakonen. Algorithms and Networking for Computer Games. 2006. p 49-89.

13. Buckland. Programming Game AI Example. Wordware Publishing. 2004. p 3-17.

14. Ігровий штучний інтелект [Електронний ресурс]. — Режим доступу: https://pages.fandom.com/ru (дата звернення: 26.04.2020).

15. Ігровий ШІ [Електронний ресурс]. — Режим доступу: https://dic.academic.ru/dic.nsf/ruwiki/936246 (дата звернення: 15.04.2020).

16. Лотів А.В., Поспєлова І.І. Багатокритеріальні задачі прийняття рішень: Навчальний посібник. Москва, 2008. – 197 с.

17. Мазурова О.А. Метод автоматизированной поддержки формирования множества критериев для задач принятия решений // Вісник Національного Технічного університету ХПІ. Тематичний збірник наукових праць Нові рішення у сучасних технологіях. Харків: НТУ ХПІ. 2001. № 15. с. 36-42

18. Oleksandr Topchii, Oleksandr Samantsov, Oksana Mazurova, Mariia Shirokopetleva. A Study of Optimization Models for Creation of Artificial Intelligence for The Computer Game in The Tower Defense Genre. Problem of Infocommunications. Science and Technology (PIC S&T′2020), Kharkiv, Ukraine. 6-9 October 2020.