Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Факультет інформатики
Кафедра мультимедійних систем

Кваліфікаційна робота

освітній ступінь – бакалавр

на тему: **«РОЗРОБКА ГРИ НА UNITY/C# З ІМПЛЕМЕНТАЦІЄЮ ШТУЧНОГО ІНТЕЛЕКТУ**

Виконав: студент 4-го року навчання,

Освітньої програми «Прикладна математика», 113

Кириняченко Ростислав Вадимович

Керівник

Борозенний С.О.

Рецензент

_____

Кваліфікаційна робота захищена з оцінкою

_____

Секретар ЕК

_____

«____» _____
20____ р.

Київ 2023

Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Факультет інформатики

Кафедра мультимедійних систем

ЗАТВЕРДЖУЮ
Зав.кафедри мультимедійних систем
доц Жежерун О.П.
_____
“____” _____2023 року

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ
для кваліфікаційної роботи

студента Кириняченка Ростислава Вадимовича факультету інформатики 4-го курсу
Тема: Розробка гри на Unity/C# з імплементацією штучного інтелекту

Зміст ТЧ до кваліфікаційної роботи:

1. Індивідуальне завдання
2. Зміст
3. Вступ
4. Огляд технологій розробки ігор та штучного інтелекту
5. Розробка гри шьогі на unity/c# з імплементацією штучного інтелекту
6. Висновки
7. Список використаних джерел

Дата видачі „___” _____ 2023 р.

Керівник Борозенний С.О._____
(підпис)

Завдання отримав _____
(підпис)

ЗМІСТ

ВСТУП

1.1 Актуальність теми

Актуальність теми «Розробка гри на Unity/C# з імплементацією штучного інтелекту» зумовлена багатьма факторами.

На сьогоднішній день відбувається активний розвиток штучного інтелекту та його застосування в різних сферах людської діяльності. Використання якісного AI для розробки гри дозволяє покращити геймплей та забезпечити більш високу конкурентоспроможність розробленого продукту на ринку ігрових застосунків.

Платформа Unity та мова програмування C# є широко використовуваними інструментами в ігровій індустрії, що забезпечують гнучкість та ефективність розробки. Вивчення їхніх особливостей та застосування для розробки гри з імплементацією штучного інтелекту є актуальним для отримання практичного досвіду та навичок роботи з сучасними технологіями.

Крім того, актуальність теми підкреслюється постійним розвитком технологій та потребою в освіті нових спеціалістів, здатних працювати з сучасними інструментами розробки. Вивчення платформи Unity, мови програмування C# та алгоритмів штучного інтелекту дозволяє формувати професійні навички, необхідні для успішної кар'єри в галузі ігрової розробки.

Таким чином, актуальність даної теми полягає в дослідженні можливостей сучасних технологій штучного інтелекту для розробки гри, відтворенні складної поведінки штучного інтелекту та наданні високоякісного ігрового досвіду. Дослідження такої теми сприяє розвитку ігрової індустрії та підвищенню якості комп'ютерних ігор.

1.2 Мета та завдання роботи

Завданням даної кваліфікаційної роботи є розробка комп'ютерної гри на платформі Unity, з використанням мови програмування C# та імплементації штучного інтелекту на основі математичних алгоритмів. Робота спрямована на вивчення теоретичних основ гри шьогі, ознайомлення з сучасними технологіями розробки ігор та штучного інтелекту, а також розробку застосунку, який демонструє можливості штучного інтелекту в ігровому середовищі.

Результатом успішного виконання даної кваліфікаційної роботи стане розроблена комп'ютерна гра з імплементованим штучним інтелектом на платформі Unity.

1.3 Об'єкт та предмет дослідження

Об'єктом дослідження є математичні алгоритми та методи штучного інтелекту, які можуть бути застосовані для реалізації комп'ютерної гри. Особлива увага приділяється аналізу, проектуванню, реалізації, тестуванню та оптимізації алгоритмів штучного інтелекту.

Предметом дослідження у даній роботі є комп'ютерна версія гри шьоґі, розроблена на платформі Unity з використанням мови програмування C#.

У дослідженні розглядаються питання, пов'язані з аналізом сучасних технологій розробки ігор, використання платформи Unity та мови програмування C# для створення гри сьогі з штучним інтелектом, а також переваги та недоліки різних математичних алгоритмів для імплементації штучного інтелекту в грі.

ОГЛЯД ТЕХНОЛОГІЙ РОЗРОБКИ ІГОР ТА ШТУЧНОГО ІНТЕЛЕКТУ

2.1 Платформа Unity: особливості, переваги та недоліки

Unity - це потужна та гнучка платформа для розробки ігор та інтерактивного контенту, яка пропонує широкий спектр можливостей для розробників.[1] Unity підтримує багато платформ, мов програмування та апаратних засобів, що робить її популярним вибором серед професіоналів ігрової індустрії.

Особливості Unity:

1. Підтримка багатьох платформ: Unity дозволяє розробляти ігри для різних платформ, таких як PC, консолі, мобільні пристрої та VR/AR.
2. Єдине інтегроване середовище розробки (IDE): Unity пропонує зручний інтерфейс для розробки, що спрощує процес створення ігор.
3. Скриптування на мові C#: Unity дозволяє розробляти ігрову логіку за допомогою мови програмування C#, яка широко використовується в ігровій індустрії.
4. Багатий асортимент ресурсів: Unity пропонує велику кількість плагінів, бібліотек та інших ресурсів, які спрощують розробку ігор.[3]

Переваги Unity:

1. Швидкість розробки: Unity надає інструменти, які дозволяють швидко прототипувати та розробляти ігри.
2. Спільнота розробників: Unity має велику активну спільноту, яка допомагає у вирішенні проблем та навчанні.[2]
3. Безкоштовна версія: Unity пропонує безкоштовну версію для індивідуальних розробників та невеликих команд, що знижує вхідний поріг для початківців.

4. Гнучкість: Unity дозволяє легко інтегрувати різні технології, такі як штучний інтелект, фізичні двигуни, аудіо системи тощо.

Недоліки Unity:

1. Відносно висока вимогливість до ресурсів: Unity може вимагати відносно високі апаратні ресурси, особливо при роботі з великими і складними проектами.
2. Ліцензійні обмеження: Преміальні функції та підтримка доступні тільки за додаткову плату, що може бути обтяжливим для невеликих студій та незалежних розробників.
3. Вивчення: Хоча Unity має досить зручний інтерфейс, потребує деякого часу для опанування всіх його можливостей, особливо для початківців.
4. Оптимізація: У деяких випадках, розроблені ігри на Unity можуть мати проблеми з продуктивністю на слабкіших пристроях через відсутність автоматичної оптимізації. Розробники повинні додатково працювати над оптимізацією для досягнення бажаної продуктивності.[4]

Програмні інструменти Unity:

1. Unity Hub - це інструмент для управління версіями Unity, допомагає зберігати та керувати проектами Unity. Він також дозволяє легко встановлювати різні версії Unity та додаткові компоненти.

2. Unity Editor - це інтегроване середовище розробки, що дозволяє розробникам створювати відеоігри з використанням візуального редактора та мови програмування C#.

3. Visual Studio - це інтегроване середовище розробки програмного забезпечення, яке дозволяє розробляти та налагоджувати програми на мові C#. Воно може бути інтегровано з Unity, щоб додати підтримку мови програмування C# для розробки Unity-проектів.[5]

4. Git - це система контролю версій, яка дозволяє розробникам зберігати та керувати кодом своїх проектів.[6]
5. AI або машинне навчання бібліотеки для C# - такі бібліотеки, як TensorFlow.NET[7], Accord.NET, CNTK[8] та ML.NET[9], можуть допомогти розробникам реалізувати алгоритми штучного інтелекту для гри.

Загалом, платформа Unity є потужним та гнучким інструментом для розробки ігор та інтерактивного контенту. Її особливості, переваги та активна спільнота роблять її привабливим вибором для розробників на різних етапах своєї кар'єри. Однак, необхідно враховувати деякі недоліки, зокрема вимоги до ресурсів[4] та ліцензійні обмеження, перед тим як обрати Unity для свого наступного проекту.

2.2 Програмування на мові C#: основні характеристики та застосування

C# є сучасною, об'єктно-орієнтованою мовою програмування, розробленою компанією Microsoft. Вона є частиною платформи .NET і забезпечує простоту, гнучкість та високий рівень продуктивності для розробників. Основні характеристики та застосування C# включають наступне:

Основні характеристики:

1. Об'єктно-орієнтованість: C# підтримує основні принципи об'єктно-орієнтованого програмування, такі як інкапсуляція, наслідування та поліморфізм, що сприяє створенню чіткої та зрозумілої структури коду.
2. Синтаксис: C# використовує синтаксис, схожий на мови програмування C++ та Java, що спрощує навчання та перехід між мовами для розробників.
3. Статична типізація: C# використовує статичну типізацію, яка дозволяє забезпечити безпечність типів, покращує продуктивність та спрощує відлагодження коду.
4. Підтримка різних парадигм програмування: C# підтримує не тільки об'єктно-орієнтоване програмування, але й інші парадигми, такі як функціональне, паралельне та асинхронне програмування.
5. Багатоплатформність: Завдяки платформі .NET, C# дозволяє розробляти застосунки для різних платформ, включаючи Windows, macOS, Linux, мобільні пристрої та веб-застосунки.

Застосування:

1. Розробка ігор: C# є однією з основних мов програмування, що використовуються для розробки ігор на платформі Unity.
2. Розробка веб-застосунків: За допомогою ASP.NET, розробники можуть створювати потужні та масштабовані веб-застосунки на основі C#.

3. Розробка мобільних додатків: C# використовується для розробки мобільних додатків на платформах Android та iOS через Xamarin, який є частиною платформи .NET.
4. Розробка настільних застосунків: За допомогою C# та .NET можна створювати настільні застосунки для Windows, macOS та Linux.
5. Розробка програмного забезпечення для робототехніки та IoT: C# використовується для розробки програмного забезпечення для роботів, дронів, вбудованих систем та IoT-пристроїв, завдяки гнучкості та масштабованості мови.

Загалом, C# є потужною та гнучкою мовою програмування, яка має великий спектр застосувань. Вона популярна серед розробників завдяки своїй простоті, продуктивності та широким можливостям, що надаються платформою .NET.

2.3 Штучний інтелект: визначення, класифікація та застосування

Штучний інтелект (ШІ) – це галузь комп'ютерних наук, яка вивчає способи створення машин та програм, здатних виконувати задачі, які зазвичай вимагають людського розуму. Ці задачі можуть включати рішення проблем, розпізнавання мови, навчання, планування, сприйняття та інше. З розвитком технологій та алгоритмів, штучний інтелект стає дедалі більш впливовим та всеосяжним.

Штучний інтелект можна класифікувати за різними критеріями:

1. За ступенем автономності:
    - Напівавтономний ШІ: використовується для підтримки людини у вирішенні проблем чи покращення роботи систем.
    - Автономний ШІ: здатний працювати без втручання людини, самостійно приймаючи рішення.
2. За способом навчання:
    - Символічний ШІ: базується на знаннях, представлених у формі символів та правил.
    - Підкріплення навчання: машина навчається на основі взаємодії з середовищем та отримання відповідних винагород.
    - Машинне навчання: системи, які здатні вивчати закономірності та залежності на основі аналізу даних.
3. За рівнем загальності:
    - Вузький (слабкий) ШІ: спеціалізований на виконанні одного або декількох визначених завдань.
    - Загальний (сильний) ШІ: має потенціал виконувати будь-яку роботу, яку може виконати людський розум.

У даній роботі будуть розглянуті алгоритми штучного інтелекту), які можна використати для розробки гри шьоґі на платформі Unity, зокрема мінімакс, альфа-бета відсічення та Monte Carlo Tree Search (MCTS).

2.4 Математичні алгоритми в ігровій розробці

Математичні алгоритми в ігровій розробці використовуються для різних цілей, таких як обчислення фізики, графіки та штучного інтелекту. У цьому розділі розглянемо деякі з основних математичних алгоритмів, які використовуються в ігровій розробці.[10]

1. Обчислення фізики

В іграх зазвичай необхідно мати точне обчислення руху об'єктів, зіткнень та зіткнень з поверхнею. Для цього використовуються алгоритми фізики, такі як розрахунок імпульсу, динаміка тіл, та інші.

2. Обчислення графіки

У сучасних іграх використовується велика кількість різних ефектів та рендерингу, що вимагає значного обчислювального потужності. Основними математичними алгоритмами в цьому випадку є алгоритми трасування променів, тіней, зіткнень та зорової перспективи.

3. Штучний інтелект

Штучний інтелект використовується для створення комп'ютерних противників, що здатні до вивчення та прийняття рішень відповідно до ситуації в грі. Основними математичними алгоритмами, що використовуються в цьому випадку, є алгоритми машинного навчання, генетичні алгоритми та алгоритми пошуку шляху.

4. Анімація

Анімація в іграх вимагає точних обчислень, щоб створити плавний та реалістичний рух об'єктів в грі. Для цього використовуються алгоритми, такі як інтерполяція, кінематика та інші.

Усі ці математичні алгоритми необхідні для створення реалістичної та динамічної гри, що здатна забезпечити користувачеві захоплюючий ігровий досвід. Крім того, з точними обчисленнями та алгоритмами можна створювати

більш складні та витончені ігрові ефекти, які допомагають залучати більше користувачів до гри.

Важливим аспектом при використанні математичних алгоритмів є їх ефективність та швидкість. Ігрові движки та програми, такі як Unity, зазвичай мають вбудовані бібліотеки математичних алгоритмів, які оптимізовані для роботи в ігровому середовищі. Однак, для складних та нестандартних випадків може бути необхідним самостійно розробляти та оптимізовувати алгоритми.

У цілому, математичні алгоритми в ігровій розробці є необхідним інструментом для створення реалістичної та захоплюючої гри. Використання правильних алгоритмів може значно покращити якість та продуктивність гри, що дозволяє розробникам створювати більш складні та витончені ігри.

2.5 Алгоритм Мінімакс

Алгоритм мінімакс є одним з найпоширеніших алгоритмів для розробки ігрового штучного інтелекту в іграх з нульовою сумою для двох гравців.[13] Цей алгоритм обчислює найкращий хід, моделюючи всі можливі результати гри і вибираючи той, який максимізує мінімальний виграш гравця (звідси і назва "мінімакс"). Алгоритм працює, рекурсивно оцінюючи дерево гри, щоб присвоїти оцінку кожному стану гри, з метою мінімізації максимального виграшу суперника.[11][13][14]

Опис роботи алгоритму:[12][17]

1. Визначення поточного стану гри та його оцінки (оцінка може бути представлена числом, яке відображає перевагу одного гравця над іншим).
2. Генерування списку можливих ходів від поточного стану гри.
3. Для кожного можливого ходу симулюється гра, а також визначається оцінка кожного нового стану гри.
4. Якщо досягнуто максимальної глибини пошуку або гра закінчилась, то повертається оцінка поточного стану гри.
5. Якщо глибина пошуку не досягнута, алгоритм мінімакс рекурсивно запускається для кожного з нових станів гри, від яких можна зробити хід, та повертається максимальне (для гравця, який має хід) або мінімальне (для суперника) значення серед всіх нових станів гри.
6. На кожному рівні дерева, коли хід належить гравцю-максимізатору, він вибирає найвище значення серед своїх нащадків, а коли хід належить гравцю-мінімізатору, він вибирає найменше значення серед своїх нащадків

Переваги алгоритму Мінімакс:

1. Гарантує знаходження оптимального ходу.
2. Відносно простий в реалізації

3. Можливість використання евристичної інформації (матеріальний баланс, контроль над центром дошки, безпеку короля та інші фактори) для поліпшення пошуку.

Недоліки алгоритму Мінімакс:

1. вимагає побудови повного дерева гри
2. Низька ефективність у випадку гри з великою кількістю можливих ходів, що ускладнює пошук оптимального рішення.
3. Потреба у значних обчислювальних ресурсах, особливо при великій глибині пошуку та складних ігрових ситуаціях.

Часова складність алгоритму мінімаксу може бути проаналізована з урахуванням розміру дерева гри. У цьому контексті розмір визначається коефіцієнтом розгалуження (b), який є середньою кількістю дозволених ходів у кожній позиції, та глибиною дерева (m), яка є максимальною кількістю ходів до кінця гри. Найгірша часова складність алгоритму мінімаксу у випадку повного перебору всіх ходів без використання евристик та оптимізацій, становить O (b^m), оскільки він досліджує всі можливі стани гри.

Просторова складність визначається максимальною глибиною дерева рекурсії, яка дорівнює глибині дерева гри. Отже, просторова складність алгоритму мінімаксу дорівнює O(bm).

Це означає, що кількість обчислень зростає експоненційно зі збільшенням глибини пошуку та кількості гілок на кожному рівні дерева. В такому випадку, алгоритм мінімакс може бути повільним і вимагати значної обчислювальної потужності. Отже, сирий алгоритм мінімаксу часто непрактичний для ігор з великими просторами пошуку, таких як шахи. Для оптимізації його роботи можна застосувати такі методи, як альфа-бета відсікання, ітеративне поглиблення та використання евристик для оцінювання.

Ці методи можуть значно зменшити кількість досліджуваних вузлів у дереві гри, таким чином зменшуючи часову складність алгоритму.

2.6 Методи оптимізації алгоритму мінімакс

Альфа-Бета відсічення - це метод оптимізації алгоритму мінімаксу, який має на меті усунути гілки дерева гри, які гарантовано не впливають на остаточне рішення. Він робить це, відстежуючи два значення, альфа і бета, які представляють найкращий можливий результат для гравця і найгірший можливий результат для опонента, відповідно. Коли алгоритм проходить по дереву гри, він оновлює ці значення і використовує їх для обрізання гілок, які не вплинуть на результат.

Завдяки альфа-бета відсіченню, часова складність мінімакса може бути знижена до O(b^(m/2)) у найкращому випадку, що робить алгоритм значно швидшим та ефективнішим.[15]

Ітеративне поглиблення(Iterative deepening) - це ще один метод оптимізації, який поєднує в собі переваги пошуку в глибину та в ширину. Вона передбачає запуск алгоритму мінімаксу кілька разів зі збільшенням глибини пошуку, починаючи з поверхневого пошуку і поступово поглиблюючи його. Ця стратегія дозволяє алгоритму швидко знаходити хороші ходи, одночасно досліджуючи більш глибокі рівні ігрового дерева. Ітеративне поглиблення також покращує ефективність альфа-бета обрізки, надаючи кращі початкові оцінки значень альфа і бета.Ф

РОЗРОБКА ГРИ ШЬОҐІ НА UNITY/C# З ІМПЛЕМЕНТАЦІЄЮ ШТУЧНОГО ІНТЕЛЕКТУ

3.1. Визначення, особливості та правила гри шьоґі

Шьоґі є традиційною японською стратегічною настільною грою, що виникла в 16-му столітті в Японії і відома також як японські шахи. Гра стала популярною у всьому світі завдяки своїм складним правилам та безлічі стратегічних можливостей.

Особливості гри:

1. Розміщення фігур: Гра шьоґі грається на дошці розміром 9x9 клітинок. Кожен гравець має 20 фігур, розміщених на перших трьох рядах дошки.
2. Рух фігур: У шьоґі кожна фігура має свої власні правила руху, подібно до класичних шахів. Однак, фігури можуть також здійснювати "продвинуті" рухи, залежно від їх позиції на дошці.
3. Захоплення та повернення фігур: У шьоґі існує унікальна можливість повернення захоплених фігур противника на дошку, що додає стратегічну глибину грі.
4. Апгрейд фігур: Фігури можуть отримати "продвинуті" характеристики, якщо вони досягають одного з трьох крайніх протилежних рядків дошки. Це надає нові можливості атаки та захисту, змушуючи гравців адаптуватися до змінної ситуації на дошці.
5. Перемога: Ціль гри - поставити короля противника під шах і мат або змусити противника здатися.

Шьоґі відрізняється від інших настільних ігор своєю стратегічною глибиною, великою кількістю можливих ситуацій на дошці та необхідністю адаптації до ходів противника. «Ці характеристики роблять шьоґі найскладнішою формою шахів, в яку зазвичай грають, із середньою тривалістю партії в 115 ходів і коефіцієнтом розгалуження 80, що дає

складність ігрового дерева ~ 10 ^ 226. Для порівняння, складність шахів становить ~ 10 ^ 120»[16]. Успіх у грі залежить від гарного розуміння правил, стратегій, прогнозування наслідків своїх ходів та уміння читати наміри суперника. Незважаючи на свою складність, шьоґі є чудовою грою для розвитку логічного мислення, планування та вироблення стратегічних навичок.

3.2. Проектування архітектури гри та інтерфейсу користувача

3.2.2 Клас Piece

Клас Piece відповідає за функціональність фігури шьоґі в грі. Він містить інформацію про тип фігури, її статус (противник, гравець), чи вона заручник і інші властивості. Клас також містить методи для руху фігур, перевірки можливих ходів та візуалізації фігур на дошці.

Основні елементи класу Piece:

- Enum PieceType: перелік типів фігур шьоґі (Pawn, Bishop, Rook, Lance, Knight, Silver, Gold, King).
- Поля: characterFace (текстова мітка для відображення символу фігури), PieceType piece (тип фігури), promoted (чи фігура «продвинута»), enemy (чи є фігура противником), player (чи належить фігура гравцю).
- Методи:
    - Awake: ініціалізує Piece.
    - raised/deselected: піднімає/опускає фігуру при виділенні.
    - getPiece/isEnemy/isPlayer/isPromoted: гетери для властивостей класу.
    - setPiece: встановлює значення фігури та її статуси.
    - promotion: змінює статус фігури на "перетворена" та запускає анімацію перевороту.

- moveAnimation: Coroutine для анімації руху фігури.
- flipAnimation: анімація перевороту фігури
- renderFace: рендер символу на фігурі
- pieceMoves: розраховує можливі ходи фігури на дошці.
- clearSpecificCollision/sideInverse: фільтрують можливі ходи з урахуванням перешкод та сторони гравця.
- lanceMoves/bishopMoves/rookMoves: розраховують ходи для ладьї, слона та списа.

Цей клас є ключовим елементом гри, який забезпечує роботу з фігурами.

3.2.3 Клас Tile

Клас Tile відповідає за функціональність клітинки на ігровій дошці, на якій можуть розміщуватися фігури. Ось опис ключових частин класу:

1. Змінні:

- row, col - рядок та стовпець для позиції плитки на дошці.
- piece - фігура, яка розташована на клітинці.
- highlight - об'єкт для підсвічування клітинки.

2. Методи:

- Initialize(int, int) - ініціалізація плитки з заданими рядком та стовпцем.
- Гетери та сетери для позиції плитки (getRow, getCol, setPosition).
- raised() та deselected() - методи для взаємодії з фігурою на плитці.
- highlightEnable() та highlightDisable() - методи для управління підсвічуванням плитки.
- isHighlighted(), isEnemy(), isPlayer(), isPromoted() - методи для отримання інформації про стан плитки та фігури на ній.

- getState() - метод для отримання типу фігури на плитці.
- getMoves(Tile[,]) - метод для отримання можливих ходів фігури на плитці.
- setState(PieceType, bool, bool, bool) - метод для встановлення стану фігури на клітинці.
- moveState(Tile) - метод для переміщення фігури на іншу клітинку.
- addPiece(PieceType, bool, bool, bool) та removePiece() - методи для додавання та видалення фігур з клітинки.
- checkPromotion() - метод для перевірки на можливе просування фігури.

Цей клас є ключовим елементом архітектури гри. Він забезпечує логіку для взаємодії з фігурами, підсвічування можливих ходів, перевірки на просування фігур, а також робить можливим переміщення фігур між різними плитками.

3.2.4 Клас Board

Клас **Board** відповідає за функціональність ігрової дошки. Він містить ряд методів для ініціалізації, створення та керування станом дошки, а також відповідає за генерацію списку можливих ходів для гравців.

Основні елементи класу **Board**:

1. **boardSize** - статична змінна, що визначає розмір дошки (9x9).
2. **board** - двовимірний масив класу **Tile**, який представляє собою сітку дошки.
3. **selectedTile** - змінна для зберігання вибраної плитки.

Даний клас містить такі методи:

1. **Awake()**, **Start()**, **initializeBoard()**, **prepareBoard()** - методи, що використовуються для ініціалізації та налаштування дошки при старті гри.
2. **GetAllPossibleMoves(bool isEnemy)** - метод, який повертає список можливих ходів для ворога або гравця в залежності від переданого параметра **isEnemy**.
3. **Board(Board existingBoard)** - конструктор, що створює нову дошку на основі існуючої.
4. **CloneWithMove(Tile startTile, Tile endTile)** - метод, що клонує поточну дошку з виконанням ходу, заданого параметрами **startTile** і **endTile**.

Клас **Board** є важливою частиною архітектури і забезпечує зручний контроль над структурою та станом дошки, а також забезпечує логіку для взаємодії з фігурами та клітинками. Він співпрацює з іншими класами гри, такими як **Tile**, **Piece**, та **GameController**, щоб створити цілісну та зручну архітектуру гри.

3.2.5 Клас GameController

Клас GameController є відповідальним за управління грою, він містить наступні поля та методи:

Поля:

- game, multiplayer, playersTurn, playersWin, endComment: статичні змінні для контролю гри, її стану та результату.
- enemyOffensiveLevel, EnemyMinThinkTime, searchDepth: статичні змінні, які визначають поведінку суперника в грі.
- turnCounter: лічильник ходів.
- validMoves, validKingMoves, validEnemyMoves, validEnemyKingMoves: списки, які зберігають можливі ходи для короля та суперника.

- boardPrefab, facePrefab, piecePrefab: GameObjects для роботи з префабами (шаблонами) дошки та фігур.
- board: об'єкт дошки.

Методи:

- Awake(): завантажує префаби дошки, фігур та клітинок.
- Start(): ініціалізує дошку та запускає гру
- GameLoop(): основний цикл гри, який переключається між ходами гравця та суперника.
- endGame(DeathType, bool): закінчує гру та визначає, хто переміг.
- PlayerControls(): керує ходами гравця.
- EnemyControls(): відповідає за ходи суперника.
- Minimax(TestBoard board, int depth, bool isMaximizingPlayer): Метод реалізує алгоритм Minimax для визначення найкращого ходу. Він приймає копію ігрової дошки, глибину пошуку алгоритму і булеве значення, яке вказує, чи є гравець максимізуючим або мінімізуючим гравцем.
- EvaluateBoard(TestBoard board, bool isMaximizingPlayer): Цей метод оцінює дошку, аналізуючи місцеположення і типи фігур. Він приймає копію ігрової дошки та значення, яке вказує, чи є гравець максимізуючим або мінімізуючим гравцем.
- Dictionary<PieceType, int[,]> positionalScores: Словник, який містить оцінки позицій для різних типів фігур на дошці.
- selectPiece(): Метод відповідає за вибір фігури на дошці та підсвічування можливих ходів для цієї фігури.
- deselectPiece(): Метод відміняє вибір фігури та знімає підсвічування можливих ходів.
- movePiece(Tile targetTile): Метод переміщує вибрану фігуру на цільову клітинку.

Клас GameController є важливим елементом архітектури гри та інтерфейсу користувача, він координує дії гравця та суперника та відповідає за контроль стану гри.

3.3. Реалізація алгоритмів штучного інтелекту для гри шьоґі

У грі було реалізовано два штучних інтелекти. Логіка першого ШІ міститься у функції PlayerControls(), яка виконується кожного ходу 1 гравця.

Ось короткий опис поведінки цього ШІ:

Скрипт переглядає всі клітинки на дошці і для кожної фігури, що належить гравцеві, обчислює можливі ходи і розподіляє їх по трьом спискам: playerTargets (ходи де гравець атакує фігуру іншого гравця), spaceTargets (ходи де гравець переміщує фігуру на порожню клітинку) і pawnSpaceTargets (ходи де гравець переміщує пішака на порожню клітинку).

Якщо король гравця у небезпеці, то ШІ ходить королем, а якщо ні то на основі доступних ходів ШІ вирішує, чи атакувати фігуру гравця, чи піти на вільну клітину. Рішення приймається рандомно, але шанс що ШІ атакує фігуру більше ніж, що він походить на вільну клітинку. Полтім, обраний хід виконується за допомогою функції pieceMovement().

Цей скрипт є дуже простою реалізацією ШІ для шьоґі і обіграти його доволі легко.

Другий скрипт реалізований у функції EnemyControls(), яка виконується кожний хід 2 гравця і знаходить найкращий хід, використовуючи алгоритм Мінімакс.

Цей скрипт створює масив усіх можливих ходів противника, потім для кожного ходу створює клон ігрової дошки і виконує цей хід на клонованій дошці. Після цього він викликає функцію Minimax(), щоб оцінити дошку і отримати оцінку цього ходу. В результаті ШІ обирає хід з найбільшою кількістю балів і виконує його на реальній ігровій дошці.

Функція Minimax() є рекурсивною реалізацією алгоритму Minimax з альфа-бета відсіченням. Вона отримує на вхід копію поточної ігрової дошки, глибину пошуку, і змінну isMaximizingPlayer, яка вказує на те, є поточний

гравець максимізуючим чи мінімізуючим, а також значення альфа і бета для альфа-бета відсічення. Функція повертає найкраще значення дошки на основі поточного гравця (мінімізуючого чи максимізуючого).

Коли поточний гравець максимізує, функція перебирає всі можливі ходи, клонує дошку, симулює хід на клонованій дошці і рекурсивно викликає Minimax() з протилежним гравцем (мінімізуючим) і зменшеною глибиною. Найкраще значення - це максимум поточного найкращого значення і нового значення дошки. Альфа-бета відсічення застосовується шляхом оновлення значення альфа і виходу з циклу, коли бета менше або рівне альфа.

Коли поточний гравець мінімізує, функція робить те ж саме, що і для гравця, який максимізує, але з деякими відмінностями. Найкраще значення - це мінімум поточного найкращого значення і нового значення дошки. Замість альфа оновлюється значення бета, і цикл переривається, коли бета менше або дорівнює альфа.

Якщо глибина пошуку дорівнює 0, то функція Minimax() викликає метод EvaluateBoard, який є евристичною функцією оцінки заданого стану дошки у грі. Метод отримує на вхід копію дошки та змінну isMaximizingPlayer й перебирає усі клітинки на дошці, перевіряючи наявність і тип фігури (пішак, кінь, слон, тура, спис, срібло, золото або король) на кожній клітинці. Кожен тип фігур має певну прописану вартість, і ці вартості підсумовуються для підрахунку оцінки. Крім вартості самих фігур, код також враховує вартість їхніх позицій. Словник positionalScores<PieceType, int[,]> містить двовимірні масиви для кожного типу фігур, що представляють цінність знаходження фігури на тій чи іншій клітинці. Код додає або віднімає позиційну оцінку для кожної фігури, залежно від того, чи належить вона гравцю, який максимізує, чи ні. Після того як метод пройшов по всім клітинкам він повертає обчислений результат для даного стану дошки. Позитивне значення вказує на перевагу гравця, який максимізує, а від'ємне - на перевагу гравця, який мінімізує.

Цей скрипт є реалізацією повноцінного ШІ для шьогі який непросто обіграти.

3.4. Тестування та оптимізація штучного інтелекту

Для оцінки ефективності та продуктивності алгоритму Minimax було проведено тестування на різних рівнях глибини. Глибина в цьому контексті визначає кількість ходів вперед, які розглядає алгоритм при визначенні найкращого ходу.

Результати тестування алгоритму без альфа-бета відсічення, де виміряно середній час, що необхідний штучному інтелекту, щоб зробити хід при відповідній глибині пошуку:

- Глибина 1: 0,0005 секунди
- Глибина 2: 0,03 секунди
- Глибина 3: 0,65 секунди
- Глибина 4: 36 секунд
- Глибина 5: 747 секунд

Результати алгоритму з додаванням альфа-бета відсічення

- Глибина 1: 0,0005 секунди
- Глибина 2: 0,02 секунди
- Глибина 3: 0,25 секунди
- Глибина 4: 5 секунд
- Глибина 5: 67 секунд

Ці дані наглядно демонструють значущість альфа-бета відсічення в оптимізації алгоритму Minimax. На глибині 1 алгоритм здатен лише на прості дії, такі як знищення ворожої фігури, якщо вона входить в зону його ходу. З глибиною 2 та 3 алгоритм починає демонструвати більш стратегічну поведінку, хоча в основному він залишається оборонним і рідко ризикує своїми фігурами.

Починаючи з глибини 4 алгоритм стає ще «розумнішим», але починаються проблеми з його часовою складністю.

Без застосування альфа-бета відсічення час на розрахунок ходу збільшується до середнього значення 36 секунд. Втім, з альфа-бета відсіченням середній час визначення ходу скорочується до 5 секунд, хоча в окремих випадках може досягати і більше 20 секунд. Це відбувається через те, що ефективність альфа-бета відсічення в найгірших сценаріях знижується, внаслідок чого відсікається лише невелика кількість гілок дерева гри. Втім, незважаючи на це, тестові дані свідчать про те, що альфа-бета відсікання є критично важливим для підвищення ефективності алгоритму Minimax, особливо при більшій глибині пошуку.

ВИСНОВКИ

4.1. Основні результати дослідження

В ході написання кваліфікаційної роботи були проаналізовані особливості платформи Unity та мови програмування C#. Розглянуті різні варіанти імплементації штучного інтелекту та використання математичних алгоритмів у відеоігровій розробці. На основі отриманих даних створено гру шьоґі, в якій був успішно інтегрований алгоритм мінімакс для реалізації штучного інтелекту та реалізоване альфа-бета відсічення для оптимізації цього алгоритму.

Під час розробки додатку я детальніше ознайомився можливостями платформи Unity та розвинув свої навички у створенні комп’ютерних ігор з імплементацією штучного інтелекту.

4.2. Рекомендації щодо подальшого вдосконалення гри та штучного інтелекту

У проєкту є великі перспективи для розвитку. Зокрема, можна реалізувати можливість гри проти інших гравців, а також попрацювати над подальшою оптимізацією ШІ шляхом додавання додаткових евристичних функцій та кешування і впровадження нових алгоритмів штучного інтелекту, таких як наприклад "Monte Carlo Tree Search".

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Unity Technologies. (2021). Офіційний сайт Unity. https://unity.com/
2. Unity Forums. (2021). Форуми спільноти Unity для обговорення питань, пов'язаних з розробкою ігор та інтерактивного контенту https://forum.unity.com/
3. Unity Asset Store. (2021). Магазин активів та ресурсів для розробки ігор на платформі Unity. https://assetstore.unity.com/
4. Unity Learn. (2021). Оптимізація ігор на Unity. https://learn.unity.com/tutorial/introduction-to-optimization-in-unity#
5. Microsoft. (2021). Visual Studio для Unity. https://docs.microsoft.com/uk-ua/visualstudio/cross-platform/getting-started-with-visual-studio-tools-for-unity
6. GitHub. (2021). Git для розробників Unity. https://github.com/Unity-Technologies
7. TensorFlow.NET. (2021). TensorFlow.NET для C#. https://github.com/SciSharp/TensorFlow.NET
8. CNTK. (2021). CNTK для C#. https://docs.microsoft.com/en-us/cognitive-toolkit/
9. ML.NET. (2021). ML.NET для C#. https://dotnet.microsoft.com/apps/machinelearning-ai/ml-dotnet
10. Math for Game Developers: Відеоуроки про математичні алгоритми, що застосовуються в ігровій розробці. https://www.youtube.com/watch?v=sKCF8A3XGxQ&list=PLW3Zl3wyJwWOpdhYedlD-yCB7WQoHf-My
11. Пояснення роботи алгоритму мінімакс з альфа-бета відсіченням https://www.youtube.com/watch?v=l-hh51ncgDI
12. Покрокове керівництво по створенню простого штучного інтелекту для шахів https://www.freecodecamp.org/news/simple-chess-ai-step-by-step-1d55a9266977/

13. Minimax https://cs.stanford.edu/people/eroberts/courses/soco/projects/1998-99/game-theory/Minimax.html
14. Minmax (sometimes Minimax, MM or saddle point) https://en.wikipedia.org/wiki/Minimax
15. How to derive the time complexity of alpha-beta pruning https://stackoverflow.com/questions/16328690/how-do-you-derive-the-time-complexity-of-alpha-beta-pruning
16. An AI for Shogi http://web.stanford.edu/class/archive/cs/cs221/cs221.1192/2018/restricted/posters/reguchi/poster.pdf
17. Building an AI for playing Japanese Shogi Chess https://habr.com/ru/articles/168867/
18. An attempt to remake the famous Japanese board game Shogi on Unity https://github.com/NicholasSebastian/Shogi-Game