Міністерство освіти і науки України
Харківський національний університет імені В.Н. Каразіна
Факультет математики і інформатики
Кафедра теоретичної і прикладної інформатики

Пояснювальна записка

до кваліфікаційної роботи

На тему «Розробка мобільного додатку з використанням Reinforcement Learning»

Виконав: студент 4 курсу, групи МФ-41,
спеціальність 122 – Комп'ютерні науки та інформаційні технології, освітня програма: «Інформатика»
Деревянко Олег Андрійович
Керівник: викл. Панченко А. С.
Рецензент: викл. Дейнега О. А.

Харків, 2020 рік

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

1. ШІ – штучний інтелект
2. МН – машинне навчання
3. НП (RL) – навчання з підкріпленням
4. MCTS – пошук дерев Монте-Карло
5. DQN – Deep Q-Network

1. ВСТУП

1. 1. Актуальність теми

У сучасному світі мобільні додатки стали невід’ємною частиною нашого повсякденного життя, забезпечуючи користувачів широким спектром послуг, від розваг до бізнес-додатків. Щодня мільйони людей використовують свої мобільні пристрої для доступу до різноманітних програм, що надають їм можливість ефективніше керувати своїм часом, отримувати нові знання, спілкуватися з іншими та розважатися. У зв'язку з цим, розробка інноваційних та розумних додатків стає все більш важливим завданням для розробників.

Останніми роками розвиток штучного інтелекту та машинного навчання значно вплинув на спосіб створення та функціонування цих додатків. Зокрема, методи ШІ використовуються для створення адаптивних систем, здатних підлаштовуватися під потреби користувача, надавати персоналізовані рекомендації та автоматизувати складні процеси. Особливу увагу привертає навчання з підкріплення, яке дозволяє агентам навчатися та вдосконалюватися на основі досвіду, отриманого шляхом взаємодії з середовищем. НП є одним з провідних методів МН, який знайшов широке застосування в різних галузях, від робототехніки до ігрової індустрії.

Гра в шашки завжди була випробувальним майданчиком для алгоритмів ШІ через її складність та глибину стратегій. Історія розробки ШІ для шашок починається ще з середини XX століття, коли вперше були запропоновані алгоритми, здатні грати в шашки на досить високому рівні. З того часу технології значно розвинулися, і сьогодні комп'ютерні програми здатні перемагати навіть найсильніших гросмейстерів. Використання RL у цій сфері відкриває нові можливості для створення ще більш потужних і ефективних алгоритмів.

Актуальність теми зумовлена стрімким розвитком мобільних технологій та потребою у створенні більш адаптивних і розумних додатків. Інтеграція RL у мобільні додатки може значно покращити користувацький досвід, забезпечуючи персоналізовані рекомендації, оптимізацію ресурсів і підвищену автономність додатків. Крім того, застосування RL для розробки мобільного додатку для гри в шашки дозволяє не тільки підвищити рівень гри, але й створити нові навчальні інструменти, які допоможуть гравцям вдосконалювати свої навички та розуміти складні стратегії гри. Таким чином, дослідження та розробка мобільних додатків з використанням RL є надзвичайно актуальною темою, що поєднує передові технології ШІ з практичними потребами користувачів.

1. 2. Мета і завдання роботи

Метою цієї роботи є дослідження та розробка мобільного додатку для гри в шашки з використанням НП. Ця мета обумовлена прагненням створити інноваційний додаток, який не тільки забезпечить високу якість гри, але й надає користувачам можливість взаємодіяти з розумним супротивником, що постійно вдосконалюється на основі досвіду гри.

1. 3. Об’єкт і предмет дослідження

Об’єктом дослідження є мобільні додатки для гри в шашки з використанням алгоритмів НП. Це включає вивчення різних аспектів розробки мобільних додатків, від вибору платформи до реалізації функціоналу, що базується на ШІ. Особливу увагу приділено аспектам інтеграції моделей RL у мобільні додатки для забезпечення адаптивності та високої якості гри.

Предметом дослідження є методи та технології розробки мобільних додатків з використанням RL, включаючи аналіз існуючих алгоритмів гри в шашки, процес навчання агентів RL, та інтеграція цих методів у мобільні додатки. Це охоплює широкий спектр технологій, від класичних алгоритмів ШІ до сучасних методів глибокого навчання, а також практичні аспекти розробки мобільного програмного забезпечення.

1. 4. Постановка задачі

Потрібно розробити алгоритм для гри в шашки та реалізувати гру з алгоритмом у мобільному додатку. Були виділені основні задачі, які потрібно виконати для досягнення цілі:

- Аналіз існуючих алгоритмів для гри в шашки: для того щоб розробити ефективну модель гри в шашки за допомогою RL, необхідно спершу провести детальний аналіз існуючих методів та алгоритмів, що використовуються в цій галузі. Це включає вивчення класичних алгоритмів, таких як Мінімакс із відсіченням альфа-бета, пошук дерев Монте-Карло, а також сучасних підходів, що використовують нейронні мережі.
- Розробка моделі гри в шашки: на основі проведеного аналізу потрібно вибрати найбільш підхожий алгоритм навчання з підкріплення для реалізації моделі гри в шашки. Важливо визначити оптимальну архітектуру нейронної мережі, налаштувати параметри навчання та забезпечити ефективний процес тренування агента.
- Реалізація процесу навчання моделі: після вибору алгоритму та налаштування параметрів необхідно провести процес навчання моделі. Це включає багаторазові ітерації гри, в ході яких агент буде навчатися та вдосконалювати свою стратегію. Важливо забезпечити належну оцінку ефективності моделі на кожному етапі навчання.

- Створення мобільного додатку для гри в шашки: наступним кроком є розробка мобільного додатку, який інтегруватиме розроблену модель. Це потребує вибору платформи для розробки, розробки зручного інтерфейсу користувача та забезпечення стабільної роботи додатку.

РОЗДІЛ 2. АНАЛІЗ ІСНУЮЧИХ МЕТОДІВ ДЛЯ ВИРІШЕННЯ ЗАДАЧІ ГРИ В ШАШКИ

2. 1. Алгоритм Мінімакс із відсічення альфа-бета

Алгоритм Мінімакс є одним із найстаріших і найбільш фундаментальних підходів до вирішення задач гри в шашки. Основна ідея алгоритму полягає у пошуку оптимальної стратегії для гравця, з урахуванням того, що суперник також грає оптимально. Алгоритм визначає найкращий хід, оцінюючи всі можливі ходи гравця і його суперника, створюючи дерево ігрових станів, шляхом рекурсивного аналізу можливих результатів гри.

Алгоритм Мінімакс працює шляхом рекурсивного обходу дерева можливих ходів у грі. На кожному кроці гравець намагається максимізувати свій виграш (максимізуючий гравець), тоді як його суперник намагається мінімізувати виграш гравця (мінімізуючий гравець).

Таким чином, алгоритм намагається мінімізувати потенційну втрату в найгіршому випадку, що гарантує найбільш оптимальний хід у будь-якому стані гри.

Альфа-бета відсічення є технікою оптимізації для алгоритму Мінімакс, яка дозволяє значно зменшити кількість розглянутих ходів, не змінюючи кінцевого результату обчислення. Вона працює шляхом обрізки тих гілок дерева ігрових станів, які не можуть вплинути на остаточне рішення. Це досягається за допомогою двох параметрів: альфа (найкраща знайдена оцінка для максимізуючого гравця) і бета (найкраща знайдена оцінка для мінімізуючого гравця), що на самому початку встановлюються на мінус нескінченність та плюс нескінченність відповідно.

Під час обходу дерева станів алгоритм відслідковує ці значення і використовує їх для прийняття рішення про те, чи варто продовжувати обхід

гілки. Якщо виявляється, що поточний вузол не може призвести до кращого результату, ніж вже знайдений, гілка обрізається.

Щодо переваг та недоліків алгоритму, їх можна визначити декілька:

- Оптимальність: гарантує знаходження оптимального ходу за умови, що суперник грає оптимально.
- Простота: відносно простий для розуміння та реалізації.
- Обчислювальна складність: повний обхід дерева можливих ходів може бути дуже витратним за часом і ресурсами.
- Обмеженість: залежить від глибини обходу дерева, що може обмежувати його ефективність у дуже складних іграх.

2. 2. Пошук дерев Монте-Карло

Пошук дерева за методом Монте-Карло – це сучасний підхід до розв'язання задач гри в шашки, який використовує випадкові симуляції для оцінки можливих ходів. На відміну від класичних методів, таких як Мінімакс, MCTS дозволяє ефективно досліджувати великий простір станів, що робить його особливо корисним для ігор з великою кількістю можливих комбінацій ходів. MCTS поєднує випадковий пошук і інтелектуальний підрахунок очок, щоб створити баланс між вивченням нових ігор і використанням знайомих виграшних стратегій.

MCTS складається з чотирьох основних етапів:

1. Вибір: вузол, який потрібно розширити, обирається на основі політики, що враховує як поточне значення вузла, так і потенційний виграш від подальшого дослідження.
2. Розширення: нові вузли додаються до дерева пошуку на основі можливих ходів з вибраного вузла.

3. Симуляція: випадковим чином виконується гра до кінця з нових вузлів, що додаються.
4. Зворотне оновлення: оновлюються оцінки вузлів на шляху до кореня дерева на основі результатів симуляції.
5. MCTS здатний забезпечити хорошу продуктивність навіть при обмежених обчислювальних ресурсах, що робить його привабливим для використання в мобільних додатках.

Отже, зробимо висновки щодо алгоритму MCTS:

- Гнучкість: підходить для широкого спектру ігор і задач.
- Ефективність: добре балансує між дослідженням і експлуатацією, що дозволяє знаходити оптимальні стратегії з меншими ресурсами.
- Випадковість: через використання випадкових симуляцій може не завжди знайти оптимальне рішення.
- Часові витрати: MCTS є більш ефективним, ніж традиційні методи, але може зайняти багато часу для складних ігор.

2. 3. Нейронні мережі: контрольоване навчання та навчання з підкріпленням

Нейронні мережі стали одним із найпотужніших інструментів у сфері штучного інтелекту і машинного навчання, зокрема у вирішенні задач для ігрової індустрії. Використання нейронних мереж дозволяє створювати складні ігрові моделі, що здатні виявляти та використовувати приховані закономірності ігрових стратегій. Існує два основних підходи до навчання нейронних мереж для ігрових завдань: контрольоване навчання та навчання з підкріпленням.

2. 3. 1. Контрольоване навчання

Контрольоване навчання – це підход, який навчає нейронну мережу на основі наявного набору даних, що містить вхідні приклади, наприклад, положення фігур на дошці та відповідні виходи – оптимальні ходи. Цей підхід складається з кількох ключових етапів:

1. Збір даних: необхідно зібрати великий набір даних з попередніх ігор. Це можуть бути партії, зіграні людьми, або ігри між існуючими алгоритмами.
2. Позначення даних: кожен приклад у наборі даних повинен мати відповідну мітку, яка вказує на оптимальний хід у даному положенні.
3. Навчання моделі: нейронна мережа навчається на цих даних шляхом коригування своїх ваг для мінімізації різниці між передбаченими та правильними ходами.

Переваги та недоліки контрольованого навчання:

- Простота: контрольоване навчання є відносно простим у реалізації, оскільки воно вимагає лише наявності набору даних з відповідними мітками.
- Ефективність: цей підхід може бути дуже ефективним, якщо у вас є великий і якісний набір даних.
- Залежність від даних: якість моделі сильно залежить від якості та кількості наявних даних. Якщо дані містять помилки або є неповними, модель також буде неточною.
- Обмеженість: модель може не вміти адаптуватися до нових або незнайомих ситуацій, оскільки вона навчена лише на попередніх прикладах.

2. 3. 2. Навчання з підкріпленням

Навчання з підкріпленням є більш гнучким і потужним підходом, при якому агент, що є нейронною мережею, навчається шляхом взаємодії з середовищем, наприклад, ігровою дошкою, у контексті гри в шашки, і отримання винагороди за свої дії. Основні етапи цього підходу включають:

1. Визначення середовища: агент взаємодіє з ігровою дошкою, де кожен стан визначається положенням фігур.

2. Визначення дій: агент може виконувати різні дії, такі як переміщення фігур.

3. Визначення винагороди: агент отримує винагороду за кожну дію, яка наближає його до виграшу, або штраф за дії, що ведуть до програшу.

4. Процес навчання: агент багаторазово грає в гру, поступово вдосконалюючи свою стратегію на основі отриманого досвіду та винагород.

Переваги та недоліки навчання з підкріпленням:

- Адаптивність: агент може навчитися грати в різноманітних ситуаціях, адаптуючи свою стратегію до нових викликів.
- Автономність: модель може навчатися самостійно, без необхідності великого набору попередньо позначених даних.
- Складність: навчання з підкріпленням є складним і вимагає значних обчислювальних ресурсів.
- Тривалість навчання: процес навчання може займати багато часу, особливо для складних ігор, таких як шашки.

2. 4. Гібридні підходи

Гібридні підходи до вирішення задач гри в шашки поєднують різні методи машинного навчання та традиційні алгоритми для досягнення найкращих результатів. Ці підходи використовують переваги кожного з методів, щоб компенсувати їх недоліки та створити більш потужні та ефективні системи.

Гібридні підходи до гри в шашки зазвичай включають наступні компоненти:

1. Нейронні мережі: використовуються для оцінки позицій на дошці та прийняття рішень про наступні ходи. Нейронні мережі можуть бути навчені як за допомогою контрольованого навчання, так і навчання з підкріпленням.

2. Алгоритми пошуку: використовуються для дослідження можливих варіантів ходів та вибору найбільш перспективних. Найпопулярнішим серед таких алгоритмів є MCTS, який дозволяє ефективно досліджувати дерево можливих ходів.

2. 5. Висновки

Було розглянуто чотири основні методи для вирішення задачі гри в шашки: алгоритм Мінімакс із відсічення альфа-бета, MCTS, нейронні мережі, а саме їх основні різновиди – контрольоване навчання та навчання з підкріпленням, і гібридні підходи. Кожен із цих методів має свої переваги та недоліки, які впливають на їхню ефективність та застосовність у різних контекстах.

Отже, можна зробити кілька важливих висновків щодо кожного з методів:

1. Мінімакс із відсіченням альфа-бета є ефективним і оптимальним для задач з невеликою кількістю можливих ходів, але його обчислювальна складність робить його менш придатним для складних ігор.

2. MCTS є важливим інструментом для розробки ігрових алгоритмів, хоча є сучаснішим алгоритм, але все ще може бути витратним за часом у складних іграх та у результаті не принести оптимальних рішень, через базування на випадкових симуляціях.

3. Контрольоване навчання може бути дуже ефективним за наявності великого набору даних з правильними мітками, але цей підхід обмежений залежністю від попередньо зібраних даних та нездатністю адаптуватися до нових ситуацій.

Для розробки ігрового алгоритму він потребує забагато часу аби зібрати та розміти дані.

4. Навчання з підкріпленням, хоча і складний та ресурсомісткий метод, проте пропонує високу адаптивність і здатність вчитися на основі взаємодії зі середовищем. Цей метод є найбільш гнучким та перспективним для створення систем, які можуть самостійно вдосконалюватися та адаптуватися до нових умов гри.

5. Гібридні підходи поєднують переваги різних методів і є дуже ефективними, але їхня складність і вимоги до обчислювальних ресурсів роблять їх менш привабливими для мобільних додатків, які мають надавати користувачам найкращий досвід користування.

З огляду на вищезазначене, для розробки мобільного додатку для гри в шашки було обрано навчання з підкріпленням. Цей метод забезпечує гнучкість, адаптивність та можливість створення інтелектуального агента, здатного вдосконалюватися в процесі гри. Хоча RL вимагає значних ресурсів та часу на навчання, його переваги роблять його найбільш придатним для створення мобільних додатків з високою якістю гри та користувацьким досвідом.

РОЗДІЛ 3. РОЗРОБКА МОДЕЛІ ЗА ДОПОМОГОЮ НАВЧАННЯ З ПІДКРІПЛЕННЯМ

3. 1. Архітектура агент-навколишнє середовище

В системах з навчанням з підкріпленням центральною концепцією є взаємодія між агентом та навколишнім середовищем. Ця взаємодія формується в процесі навчання агента шляхом проб і помилок, де він намагається максимізувати свій довгостроковий кумулятивний нагороджувальний сигнал. Для розуміння принципів роботи навчання з підкріплення необхідно детально розглянути архітектуру агент-навколишнє середовище, що складається з кількох ключових компонентів.

Агент — це автономна сутність, яка приймає рішення на основі отриманої інформації про стан навколишнього середовища. Основна мета агента полягає в знаходженні оптимальної політики, яка визначає, які дії він має виконувати в різних станах середовища для максимізації винагороди. Тож основні функції агента можна описати наступним чином:

1. Спостереження: агент спостерігає за станом навколишнього середовища через сенсори або інтерфейси.

2. Дії: на основі спостережень агент вибирає дії, які він вважає оптимальними.

3. Оновлення: агент оновлює свою політику та ціннісні функції на основі отриманих винагород і нових спостережень.

Навколишнє середовище — це все, що знаходиться поза агентом, і з чим він взаємодіє. Середовище має наступні характеристики:

1. Стан: стан середовища є повним описом поточної ситуації, який агент може використовувати для прийняття рішень.

2. Динаміка: середовище змінюється внаслідок дій агента, а також через власні внутрішні процеси.

3. Винагороди: середовище надає агенту винагороди за його дії, що є сигналом про те, наскільки вдалою була його дія з точки зору досягнення довгострокових цілей.

Взаємодія між агентом і навколишнім середовищем відбувається циклічно в послідовності дискретних часових кроків. Ця взаємодія може бути описана наступним процесом:

1. Агент спостерігає стан середовища S_t.

2. На основі поточного стану S_t агент вибирає дію A_t, використовуючи свою політику π.

3. Дія A_t виконується в середовищі, що призводить до зміни стану середовища на новий стан S_{t+1}.

4. Середовище повертає агенту винагороду R_{t+1}, що відповідає дії A_t і новому стану S_{t+1}.

5. Агент оновлює свою політику на основі отриманої винагороди та нової інформації про стан середовища.

Цей цикл продовжується до досягнення певного критерію завершення, наприклад, до завершення епізоду гри в шашки або досягнення встановленого порогу навчання.

3. 2. Математичні основи навчання з підкріпленням та вибір компонентів моделі

Розуміння математичних основ навчання з підкріплення є критично важливим для ефективної реалізації та вдосконалення алгоритмів навчання з підкріпленням. Тому слід розглянути ключові математичні концепції, які лежать в основі даного методу, включаючи формалізацію задачі у вигляді

Марківського процесу прийняття рішень функції винагород, ціннісні функції та алгоритми оцінки і оптимізації політики.

Марківський процес прийняття рішень є математичною моделлю для опису задач навчання з підкріпленням. Даний процес визначається наступними компонентами:

- S: множина станів середовища.
- A: множина дій, які може виконувати агент.
- P: функція ймовірностей переходу, де $P(s'|s,a)$ визначає ймовірність переходу до стану s' з стану s при виконанні дії a.
- R: функція винагород, де $R(s,a)$ визначає очікувану винагороду при виконанні дії a в стані s.
- γ: коефіцієнт дисконтівання, $\gamma \in [0,1)$, що визначає важливість майбутніх винагород порівняно з поточними.

Функція винагород $R(s,a)$ є критичною складовою RL, оскільки вона визначає, наскільки бажаною є певна дія в конкретному стані. Мета агента полягає в максимізації суми винагород, отриманих у процесі його взаємодії з середовищем. Накопичена винагорода за епізод t визначається як:

$$G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1},$$

де G_t — накопичена винагорода, γ — коефіцієнт дисконтівання, який зменшує вплив майбутніх винагород.

Ціннісні функції використовуються для оцінки ефективності дій агента. Існує два основних типи ціннісних функцій: функція цінності стану та функція цінності дії:

1. Функція цінності стану $V(s)$ визначає очікувану накопичену винагороду, починаючи зі стану s та слідуючи певній політиці π:

$$V_\pi(s) = E_\pi[G_t|S_t = s]$$

2. Функція цінності дії $Q(s, a)$ визначає очікувану накопичену винагороду при виконанні дії a у стані s та подальшому слідуванні політиці π:

$$Q_\pi(s, a) = E_\pi[G_t|S_t = s, A_t = a]$$

Для майбутньої моделі, незважаючи на велику розмірність, через використання кожної пари «стан-дія» та більшу складність навчання, була обрана функція цінності стану через гарну гнучкість, завдяки великому простору дій і політик, що змінюються залежно від стану, а також прямому порівнянню, яку дію в даний момент краще обрати.

Наступним елементом є оцінка політики, що полягає у визначенні функцій цінності для певної політики.

У задачах, де модель середовища відома і розмір простору станів є керованим, динамічне програмування виступає найкращим вибором для оцінки політики. Основною перевагою цього підходу є його здатність надавати точні оцінки функцій цінності завдяки використанню рівняння Беллмана, що дозволяє ефективно оновлювати значення станів до збіжності.

Це особливо корисно у задачах зі складними або витонченими політиками, де точність є критично важливою. На відміну від методу Монте-Карло, динамічне програмування не вимагає завершення повних епізодів, що дозволяє уникнути проблем з високою дисперсією оцінок. Порівняно з методами тимчасових різниць, не стикається з проблемами нестабільності і забезпечує більш швидке і надійне зближення при відомій моделі середовища. Хоча підхід і потребує повної моделі середовища, його ефективність і точність у відносно невеликих або середніх за розміром просторах станів роблять його найкращим вибором у цих умовах.

Отже, такою є кінцевою формулу підрахунку оцінки політики з використання рівняння Беллмана за допомогою динамічного програмування:

$$V(s) = E[R_{t+1} + \gamma V(S_{t+1}) | S_t = s]$$

3. 3. Процес навчання моделі

На початку було описана та реалізовано середовище для гри в шашки, з яким буде спілкуватися агент. Середовище містить два ключових компоненти, які є одними з основних в програмі, нарівні з самою моделлю агента, через те, що від середовища залежить якість аналізу поточного стану та перевірки коректності переходів.

1. Ініціалізація дошки

   В даному методі відбувається ініціалізація ігрової дошки за правилами шашок, помічаючи першого та другого гравців на дошці.

2. Зробити крок

   Метод реалізує перевірку, є запропонований агентом хід, легальний, тобто, чи відповідає він правилам гри, та у разі негативного результату може одразу повернути негативну нагороду для агента.

   Якщо хід є легальним, то аналізується, наскільки він є вдалим. Перш за все перевіряється як поточна дія змінила стан на дошці, наприклад, чи змінилася кількість шашок іншого гравця, це означає, що агент забрав шашку гравця.

   В залежності від цих перевірок агент отримує нагороду за наступним принципом:

- Хід, що не відповідає правилам: -10.
- Хід, що відповідає правилам: +5.
- Хід, що призвів до виводу шашки до короля: +10.
- Перемога: +100.
- Поразка: -100.

Після аналізу моделей, їх компонентів, переваг та недолів кожного, для реалізації моделі гри в шашки був обраний алгоритм Deep Q-Network. DQN є одним з найбільш популярних алгоритмів RL, який поєднує традиційне Q-навчання з глибинними нейронними мережами. Це дозволяє агенту вивчати складні стратегії та приймати рішення в середовищах з великим простором станів і дій, таких як шашки.

Основні причини, через які алгоритм DQN був обраний для реалізації:

1. Ефективність в складних середовищах

Показав високу ефективність у складних середовищах з великою кількістю станів та дій, що свідчить про його потенціал для гри в шашки.

2. Використання досвіду

Використовує досвідчену пам'ять, де зберігаються попередні переходи. Це дозволяє агенту вчитися з попередніх ігрових ситуацій, що покращує стабільність і ефективність навчання.

3. Фіксована цільова мережа.

Використовує фіксовану цільову мережу, яка оновлюється з певною частотою. Це допомагає зменшити кореляцію між поточними Q-значеннями і цільовими значеннями, що робить навчання більш стабільним.

Для реалізації моделі було вибрано такі параметри:

- Розмір досвіду пам'яті: 10000 переходів.
- Міні-пакет для навчання: 32 переходи.
- Коефіцієнт дисконтівання γ: 0.99.
- Епсилон-градієнт: початкове значення ε = 1.0, яке поступово зменшується до 0.1.
- Епсілон зниження: 0.995

3. 4. Технології

Мова програмування Python виявилася ключовим інструментом під час розробки та дослідження у рамках даної дипломної роботи. Python був обраний через свою простоту в освоєнні, широкі можливості в області машинного навчання та штучного інтелекту, а також активну спільноту та велику кількість доступних бібліотек.

Однією з причин вибору Python є одна з кращих бібліотек штучного інтелекту – TensorFlow. Вона є одним із найпопулярніших інструментів у сфері розробки та використання моделей глибокого навчання. Використання TensorFlow дозволяє зручно використовувати найпотужніші інструменти.

3. 5. Висновки

У цьому розділі було реалізовано модель для гри в шашки на основі алгоритму Deep Q-Network, і цей процес включав кілька ключових етапів:

1. Вибір моделі та алгоритмуОбрано алгоритм
   DQN для вирішення задачі гри в шашки, оскільки він добре підходить для складних ігрових середовищ з великим простором станів і дій. DQN поєднує Q-навчання з використанням глибоких нейронних мереж для аппроксимації функції Q.
2. Архітектура нейронної мережі DQN
   Спроектовано архітектуру нейронної мережі, що забезпечує необхідну обчислювальну потужність для ефективного навчання та прийняття рішень.
3. Налаштування параметрів навчання
   Визначено та налаштовано основні гіперпараметри, такі як швидкість навчання, дисконтний фактор, розмір міні-пакету та кількість епох

навчання, що призвело до успішного навчання моделі. Ці параметри були обрані на основі експериментальних результатів та оптимізації.

4. Процес навчання моделі

   Модель навчалася шляхом багаторазових ітерацій та взаємодій з ігровим середовищем. Використовуючи алгоритм DQN, агент поступово вдосконалював свої стратегії шляхом проб і помилок, накопичуючи досвід у вигляді історії ігрових сесій, з якого він навчався для поліпшення своїх рішень.

РОЗДІЛ 4. РОЗРОБКА МОБІЛЬНОГО ДОДАТКУ

4. 1. Вибір платформи

Для розробки мобільного додатку для гри в шашки з використанням натренованої моделі на основі навчання з підкріпленням було обрано платформу Android. Це рішення було прийняте на основі кількох ключових факторів, що роблять Android найкращим вибором для цього проекту:

1. Популярність платформи:

Android є найпоширенішою операційною системою для мобільних пристроїв у світі. Частка ринку Android складає понад 70% у глобальному масштабі, що забезпечує широке охоплення користувачів.

2. Відкритість та доступність:

Android є відкритою платформою, що дозволяє розробникам легше адаптувати та інтегрувати різні технології. Це особливо важливо при роботі з моделями машинного навчання та іншими передовими технологіями.

3. Різноманітність пристроїв:

Широкий спектр пристроїв, що працюють на Android, дозволяє користувачам з різними бюджетами та потребами мати доступ до додатку. Це забезпечує більш високий рівень доступності та поширення додатку.

4. Підтримка машинного навчання:

Android має добре розвинену екосистему для впровадження моделей машинного навчання. TensorFlow Lite, зокрема, є легкою і оптимізованою версією TensorFlow, призначеною для мобільних пристроїв. Це дозволяє ефективно використовувати модель DQN, розроблену в попередніх розділах, без значних втрат продуктивності.

4. 2. Архітектура мобільного додатку

Проектування архітектури мобільного додатку для гри в шашки з використанням натренованої моделі на основі RL є ключовим етапом для забезпечення надійної, масштабованої та ефективної роботи додатку. Архітектура додатку повинна включати всі необхідні компоненти для інтеграції моделі машинного навчання, реалізації ігрового процесу та забезпечення зручного інтерфейсу для користувачів.

Основні компоненти архітектури мобільного додатку:

1. Інтерфейс користувача:

- Екран головного меню: включає опції для початку нової гри, перегляду правил, налаштувань та іншого.

- Ігровий екран: показує шахову дошку, фігури, поточний стан гри, інформацію про гравців та дії, які можна виконати.

- Екран результатів: відображає результати гри, історію ходів, аналіз партії та можливість почати нову гру.

2. Логіка гри:

- Реалізація правил шахів: модуль, що забезпечує перевірку правильності ходів та дотримання правил гри.

- Механізм ходу суперника: реалізація логіки, що визначає хід суперника на основі натренованої моделі DQN.

3. Модель машинного навчання:

- Інтеграція моделі: модуль для завантаження та використання алгоритму для прийняття рішень під час гри.

- API для моделі: інтерфейс для взаємодії логіки гри з моделлю машинного навчання.

4. Дані та збереження стану

- Збереження гри: функціональність для збереження поточного стану гри та можливість продовження пізніше.

- Історія партій: збереження та перегляд історії зіграних партій для аналізу та покращення стратегії.

Архітектура мобільного додатку включає всі необхідні компоненти для інтеграції моделі НП, реалізації логіки гри та забезпечення зручного користувацького інтерфейсу. Це забезпечує створення функціонального та ефективного мобільного додатку для гри в шашки.

4. 3. Інтерфейс користувача

Розробка зручного та інтуїтивно зрозумілого інтерфейсу користувача є критично важливою для забезпечення позитивного користувацького досвіду. Інтерфейс повинен бути простим у використанні, інтуїтивно зрозумілим та візуально привабливим, що дозволить користувачам легко взаємодіяти з грою та насолоджуватися процесом гри в шашки.

4. 4. Інтеграція моделі Reinforcement Learning

Для забезпечення інтелектуального супротивника у нашому мобільному додатку, ми інтегруємо розроблену модель навчання з підкріпленням, побудовану на основі алгоритму Deep Q-Network. Цей процес включає кілька ключових етапів, які забезпечать коректну роботу моделі в межах мобільного додатку.

Кроки інтеграції моделі НП:

1. Після завершення навчання моделі НП, зберігаємо натреновану модель у форматі, який може бути легко імпортований у мобільний додаток.

2. За допомогою, сумісної з платформою Android, бібліотеки машинного навчання TensorFlow Lite імпортуємо збережену модель у проект мобільного додатку.

3. Створюємо інтерфейс для взаємодії мобільного додатку з моделлю. Це включає написання коду, який дозволить додатку надсилати поточний стан гри до моделі та отримувати оптимальний хід у відповідь.

4. Реалізація логіки гри з моделлю:

- Вбудовуємо модель у логіку гри, забезпечуючи автоматичне виконання ходів, запропонованих моделлю.

- Оновлюємо стан шахової дошки після кожного ходу, враховуючи як ходи гравця, так і моделі.

4. 5. Технології

Мова програмування Kotlin була обрана як основна для розробки мобільного додатку. Kotlin – сучасна та ефективна мова програмування, яка стрімко розвивається та на якій все частіше пишуться нові додатки та переписуються з Java старі, що додає актуальності роботи.

Для реалізації інтерфейсу користувача застосовувався сучасний інструмент Jetpack Compose, який базується на мові Kotlin та надає декларативний спосіб створення інтерфейсів, що дозволяє розробникам швидко та легко будувати складні UI-компоненти.

4. 6. Висновки

Підсумовуючи, можна сказати, що інтеграція алгоритму підкріплювального навчання в мобільний додаток для гри в шашки принесла значні покращення в порівнянні з традиційними методами. Ефективність алгоритму, зручний інтерфейс користувача та технічна стабільність додатку свідчать про успішну реалізацію проекту. Ці результати відкривають перспективи для подальшого розвитку та застосування RL в інших мобільних іграх та додатках.

Таким чином, мобільний додаток є не лише демонстрацією успіхів сучасних технологій підкріплювального навчання, але й надійним інструментом для вдосконалення навичок гри в шашки користувачів різного рівня.

ВИСНОВКИ

У дипломній роботі було досліджено та розроблено мобільний додаток для гри в шашки, заснованого на методах машинного навчання, зокрема навчання з підкріпленням. Метою роботи було створення інтелектуального супротивника для гри в шашки, який здатний адаптуватися до різних стратегій гравців та демонструвати високий рівень гри, а також реалізувати можливість гри з ним та вдосконалення своїх навичок.

Основні результати:

1. Аналіз існуючих методів

Було розглянуто основні алгоритми для гри в шашки, включаючи класичні методи, такі як алгоритм Мінімакс із відсіченням альфа-бета, та сучасні підходи, засновані на нейронних мережах і навчанні з підкріпленням.

Також досліджено гібридні методи, що комбінують різні підходи для досягнення кращих результатів.

2. Основи Reinforcement Learning

Описано основні концепції та математичні основи НП, що дозволяють агенту навчатися через взаємодію з навколишнім середовищем. Детально розглянули архітектуру агент-навколишнє середовище та ключові поняття, такі як функції винагороди, значення дій та політики.

3. Розробка моделі RL на основі DQN:

Обрано алгоритм Deep Q-Network для реалізації моделі гри в шахи. Спроектовано архітектуру нейронної мережі, налаштовано параметри навчання та проведено процес тренування моделі. Модель навчалася шляхом багаторазових ітерацій та взаємодій з середовищем, демонструючи здатність покращувати свої рішення з часом.

4. Розробка мобільного додатку:

Обрано платформу Android та інструменти для розробки мобільного додатку. Спроектовано архітектуру додатку, включаючи всі необхідні компоненти, такі як користувацький інтерфейс та обробка логіки гри.

Інтегровано натреновану модель у додаток, забезпечивши коректну взаємодію між моделлю та ігровим процесом.

Розроблений мобільний додаток для гри в шахи демонструє високу ефективність та здатність до адаптації, завдяки використанню методів НП. Модель, побудована на основі DQN, показала здатність вчитися на власному досвіді та приймати оптимальні рішення в різних ігрових ситуаціях.

У подальшій роботі можливе вдосконалення моделі шляхом використання більш складних алгоритмів RL, а також інтеграція додаткових функцій у мобільний додаток, що підвищать його функціональність та привабливість для користувачів.

Таким чином, дана дипломна робота показує, що використання сучасних методів машинного навчання може значно покращити інтелектуальні можливості мобільних додатків для ігор, що відкриває нові перспективи для розвитку цього напрямку.

ДОДАТКИ

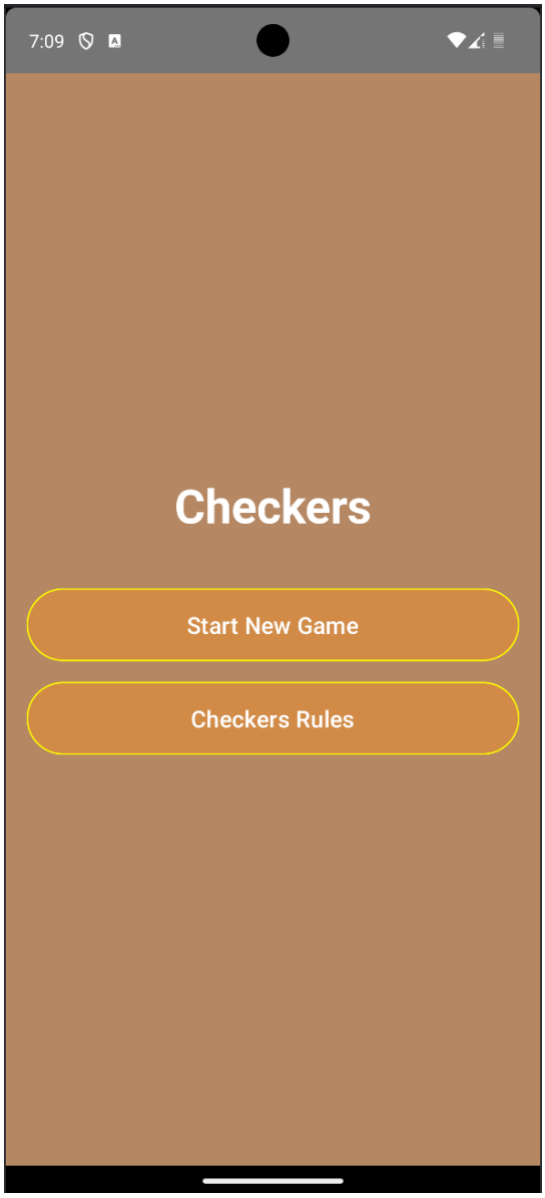

Рис. 1 – Головне меню

Рис. 2 – Початок гри

Рис. 3 – Просунута частина гри

Рис. 4 – Кінець гри

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Bishop, C. M. (2006). Pattern Recognition and Machine Learning. Springer.
2. Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., & Zaremba, W. (2016). OpenAI Gym. arXiv preprint arXiv:1606.01540.
3. Chollet, F. (2018). Deep Learning with Python. Manning Publications.
4. Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep Learning. MIT Press.
5. Kingma, D. P., & Ba, J. (2015). Adam: A method for stochastic optimization. In International Conference on Learning Representations (ICLR).
6. Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., ... & Hassabis, D. (2015). Human-level control through deep reinforcement learning. Nature.
7. Mobile OS market share worldwide 2009-2024 | Statista. Statista. URL: https://www.statista.com/statistics/272698/global-market-share-held-by-mobile-operating-systems-since-2009/ (дата звернення: 30.05.2024).
8. Sutton, R. S., & Barto, A. G. (2018). Reinforcement Learning: An Introduction (2nd ed.). MIT Press.
9. Van Hasselt, H., Guez, A., & Silver, D. (2016). Deep reinforcement learning with double Q-learning. In Proceedings of the AAAI Conference on Artificial Intelligence.