Київський національний університет
імені Тараса Шевченка
Факультет комп'ютерних наук та кібернетики
Кафедра обчислювальної математики

ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА
на тему:

Методи поліпшення штучного інтелекту для гри в шахи

Виконав студента 4 курсу
Чередника Олега Сергійовича

Науковий керівний:
Оноцький В.В.

Засвідчую, що в цій роботі немає запозичень з праць інших авторів без відповідних посилань

Студент

Роботу розглянуто й допущено до захисту на засіданні кафедри обчислювальної математики
29.05.2023р
протокол №8

Завідувач кафедри
проф. Сергій Ляшко

Київ - 2023

Зміст

Вступ

В цій роботі досліджується алгоритми штучного інтелекту. Спочатку давайте згадаємо історію шахів та труднощі, з якими зіткнулися перші ентузіасти, які почали розробляти штучний інтелект для гри в шахи. Може здатися, що всі зусилля, вкладені у розробку ШІ для настільних та комп'ютерних ігор, є просто розвагою розробників, які не бажають навчити алгоритми розпізнавати мову та перешкоди на своєму шляху. Однак ігри виявилися найкращим середовищем для тестування алгоритмів щодо їх реакції, здатності планувати дії та коригувати їх відповідно до дій супротивника. Здатність штучного інтелекту перемагати людей в іграх вважається ознакою його ефективності.

Складність ігор полягає не стільки у кількості можливих ходів та позицій, скільки в особливостях стратегій. Ігри поділяються на детерміновані та недетерміновані, а також на ті, що мають або не мають елементи невизначеності. Наприклад, азартні ігри є недетермінованими, оскільки результат визначається випадковими факторами, такими як тасування карт або підкидання кубиків. З іншого боку, ігри, як от ґо або шахи, вважаються визначеними та повністю інформативними, оскільки кожен хід визначається діями гравців, і вони бачать всю ситуацію на дошці. Але навіть для таких ігор, як шахи або хрестики-нулики, алгоритмам складно знайти оптимальний варіант, оскільки кількість можливих варіантів занадто велика для повного перебору. Це призводить до необхідності розробки ефективних алгоритмів пошуку та оцінки позицій.

У розвитку шахів зі штучним інтелектом було кілька значних кроків. У 1997 році Deep Blue, комп'ютер розроблений IBM, переміг світового шахового чемпіона Гаррі Каспарова у шаховій партії. Цей успіх став проривом у галузі ШІ та показав можливості комп'ютерів у шахових іграх.

Після Deep Blue, інші проекти ШІ продовжили розвиватися. У 2011 році був створений шаховий двигун Stockfish, який вважається одним з найсильніших у світі. Він використовує методи ШІ, такі як альфа-бета відсічення та оцінка позицій за допомогою нейронних мереж, щоб приймати розумні рішення на дошці.

З'явилися й інші інновації у сфері шахових ігор. У 2017 році компанія DeepMind випустила AlphaZero, алгоритм з підсиленого навчання, який вивчав грати в шахи, го та шашки без попередніх знань про гру. Через чотири години самостійного тренування AlphaZero стала найсильнішою програмою у світі в усіх трьох іграх.

Наступна велика подія сталася в 2021 році, коли компанія OpenAI випустила GPT-3, яка виявила високу ефективність у різних задачах, включаючи шахи. Завдяки своїй здатності до генерації тексту та аналізу інформації, GPT-3 може бути використана для аналізу позицій, розробки стратегій та розгляду різних варіантів ходів.

Загалом, розробка штучного інтелекту у шахах продовжується, і дослідники постійно шукають нові способи покращення алгоритмів і стратегій.

Метод Мінімакс

Один із головних способів пошуку оптимального рішення — є метод Мінімакс, запропонований ще в 1945 р О. Моргенштер- ном (O. Morgenstern) і Дж. Фон Нейманом (J. von Neumann).

Джон фон Нейман (John von Neumann), 1903-1957 - угорсько-американський математик, відомий роботами в сферах квантової фізики, функціонального аналізу, теорії множин, інформатики, економіки та ін.

Найбільш відомий як праотець сучасної архітектури комп'ютерів (архітектура фон Неймана), а також як творець теорії ігор і концепції клітинних автоматів.

«Коли математична дисципліна відходить досить далеко від свого емпіричного джерела, вона все більше і більше перетворюється в безцільне вправу з естетики, в мистецтво заради мистецтва. При настанні цього етапу єдиний метод лікування, на мій погляд, полягає в тому, щоб повернутися до джерела і впорснути більш-менш прямо емпіричні ідеї. Я переконаний, що це завжди було необхідно для того, щоб зберегти свіжість і життєвість математичної теорії, і що це положення залишиться в силі і в майбутньому »

У теорії ігор мінімакс - це метод, який спрямований на мінімізацію очікуваних втрат. Для цього гравець припускає, що рішення, прийняте його опонентом, буде несприятливим. Тобто, найгірший сценарій очікується до руху суперника.

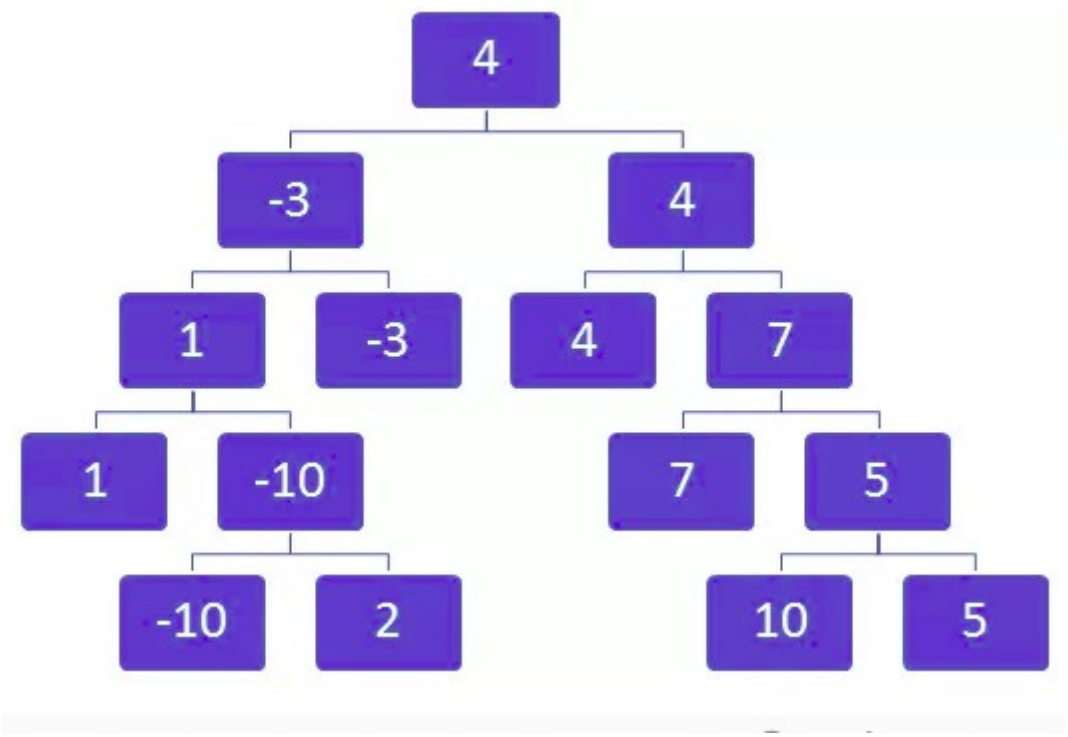
Іншими словами, метод мінімаксу складається з того, як прийняти найкраще рішення, припускаючи, що інший гравець вибере для вас найгірший сценарій.

Ми повинні взяти до уваги, що цей метод застосовний у грі для двох осіб (двоє гравців) і що це не кооператив, а гра з нульовою сумою. Це означає, що те, що виграє один гравець, втрачає інший і навпаки. Отже, кожен агент буде зацікавлений у максимізації власної корисності, навіть якщо це зашкодить іншому.

На цьому етапі ми також повинні пам’ятати, що теорія ігор - це розділ математики та економіки, який вивчає вибір, який оптимізує ситуацію людини, коли витрати та вигоди не фіксуються заздалегідь, а залежать від рішень інших.

Приклад алгоритму мінімаксу

У наступному дереві рішень ми показуємо результати, отримані гравцем x у кожен момент гри. Біля основи, на першому рівні, суперник приймає рішення. З цієї причини наведені сценарії, в яких гравець може програти -10 або виграти 5.



На другому рівні це залежить від гравця x, тому він максимізує свій прибуток. Якщо ви програєте 10 або виграєте 1, ви виграєте 1. Так само, між 5 або 7 ви виграєте 7.

Потім знову настає черга суперника, тому будуть наведені сценарії, в яких гравець x має найгірший результат, -3 та 4, залежно від випадку. Нарешті, між програшем 3 або перемогою 4 гравець x прийме рішення, яке дозволить останньому.

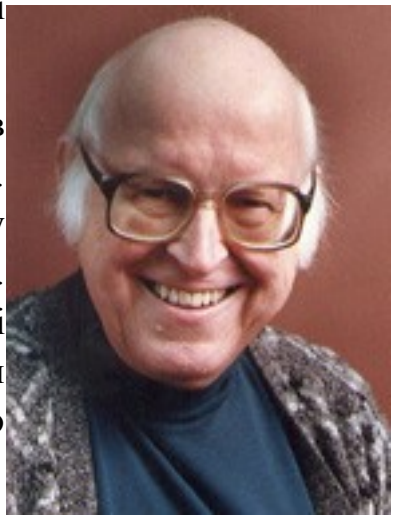
Ми повинні взяти до уваги, що значення кожного вузла будуть залежати від функції корисності.

Щоб краще зрозуміти дерево, припустимо, що в основі рішення лежить розподіл продукту. Конкурент (супротивник) може передавати розподіл на зовнішнє замовлення (див. Ліву сторону дерева). У цьому випадку він повинен вибрати, наприклад, між дилером А та Б. Таким чином, він вибирає першого, внаслідок чого гравець х втрачає 10 (якщо він вибрав В, гравець х виграє 12).

Однак, можливо, супротивник воліє поширювати свої товари сам, маючи можливість найняти моторизовані транспортні засоби або придбати вантажівку. З обох сценаріїв виберіть перший, який менш приємний для гравця х, оскільки він виграє 5, а не 10.

Альфа-Бета відсікання

Альфа-Бета відсікання один із алгоритмів оптимізації Мінімаксу.



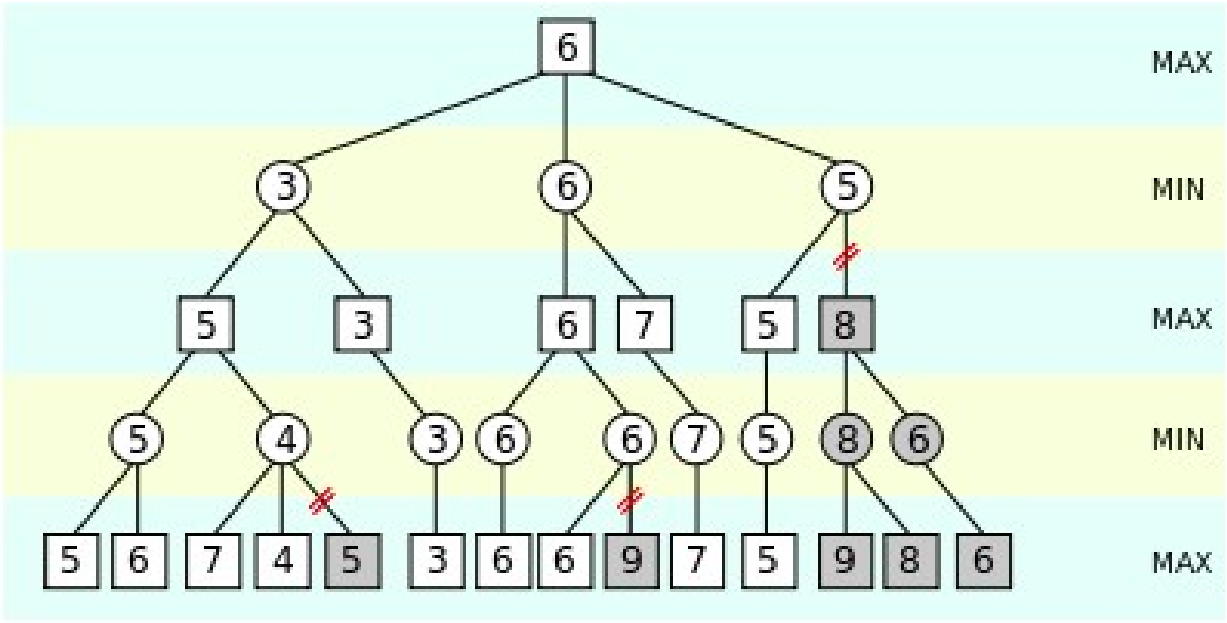
Аллен Ньюелл один з перших, що запропонував метод схожий на Альфа-Бета відсікання, ще у 1954 році. Аллен Ньюелл (1927 - 1992) — американський науковець у галузі когнітивної психології і штучного інтелекту. Працював у дослідному центрі RAND та Університеті Карнегі — Меллон. Він брав участь у розробці мови програмування IPL і двох самих ранніх програм штучного інтелекту — Logic Theory Machine і General Problem Solver.

Основна ідея

Алгоритм підтримує два значення, альфа та бета, які відповідно представляють мінімальний бал, якому забезпечений гравець, що максимізує, і максимальний бал, який гарантований гравець, що мінімізує. Спочатку альфа є негативною нескінченністю, а бета - позитивною нескінченністю, тобто обидва гравці починають з найгіршим можливим рахунком. Всякий раз, коли максимальний бал, який забезпечується гравцем, що мінімізує (тобто гравцем "бета"), стає меншим за мінімальний бал, який гарантує гравець, що максимізує

(тобто гравець "альфа") (тобто бета <альфа), максимізація гравцеві не потрібно розглядати подальших нащадків цього вузла, оскільки вони ніколи не будуть досягнуті в реальній грі.

Щоб проілюструвати це на прикладі з реального життя, припустимо, хтось грає в шахи, і настала їх черга. Переміщення "A" покращить положення гравця. Гравець продовжує шукати ходи, щоб переконатися, що кращий не пропущений. Переміщення "B" - теж хороший хід, але гравець тоді розуміє, що дозволить суперникові змусити мат в два ходи. Таким чином, інші результати ігрового ходу B більше не потрібно розглядати, оскільки суперник може змусити перемогу. Максимальна оцінка, яку суперник може змусити після ходу "B", є негативною нескінченністю: втрата для гравця. Це менше мінімального положення, яке було знайдено раніше; хід "A" не призводить до вимушених втрат за два ходи.



Ілюстрація альфа-бета обрізки. Сірі піддерева не потрібно досліджувати (коли ходи оцінюються зліва направо), оскільки відомо, що група піддерев в цілому дає значення еквівалентного піддерева або гірше, і як така не може впливати кінцевий результат. Максимальний та мінімальний рівні представляють хід гравця та противника відповідно.

Перевага альфа-бета-обрізки полягає в тому, що гілки дерева пошуку можна усунути. Таким чином, час пошуку може бути обмежений "більш перспективним" піддеревом, і одночасно можна виконати більш глибокий пошук. Як і його попередник, він належить до відділення клас алгоритмів. Оптимізація зменшує ефективну глибину трохи більше, ніж вдвічі менше, ніж у

простих мінімаксах, якщо вузли оцінюються в оптимальному або майже оптимальному порядку (найкращий вибір для сторони в русі, упорядкованої першою на кожному вузлі).

Зазвичай під час альфа-бета-версії піддерев тимчасово переважає або перевага першого гравця (коли багато ходів першого гравця хороші, і на кожній глибині пошуку перший хід, перевірений першим гравцем, є достатнім, але всі відповіді другого гравця спробувати знайти спростування), або навпаки. Ця перевага може багато разів змінювати сторону під час пошуку, якщо порядок переміщення є неправильним, щоразу призводить до неефективності. Оскільки кількість шуканих позицій експоненційно зменшується з кожним ходом ближче до поточної позиції, варто витратити значні зусилля на сортування ранніх ходів. Покращене сортування на будь-якій глибині експоненційно зменшить загальну кількість шуканих позицій, але сортування всіх позицій на глибинах біля кореневого вузла є відносно дешевим, оскільки їх так мало. На практиці порядок переміщення часто визначається результатами попередніх, менших пошуків, наприклад, через ітеративне поглиблення.

Евристичні вдосконалення

Подальше вдосконалення може бути досягнуто без шкоди точності за допомогою замовлення евристика для пошуку попередніх частин дерева, які, ймовірно, змусять альфа-бета-обрізання. Наприклад, у шахах ходи, що захоплюють фігури, можуть бути розглянуті перед ходами, які цього не роблять, і ходи, які отримали високі результати раніше призодить за допомогою аналізу ігрового дерева можна оцінити перед іншими. Ще однією поширеною і дуже дешевою евристикою є еверестичний вбивця, де останній хід, що спричинив бета-обрізання на тому ж рівні дерева в пошуку дерева, завжди розглядається спочатку. Цю ідею також можна узагальнити на набір таблиці спростування

Альфа-бета-пошук можна зробити ще швидшим, враховуючи лише вузьке вікно пошуку (як правило, це визначається здогадами на основі досвіду). Це відомо як *пошук прагнень*. У крайньому випадку пошук виконується з альфа та бета рівними; техніка, відома як пошук з нульовим вікном, *пошук у нульовому вікні*, або *скаутський пошук*. Це особливо корисно для пошуку перемог / програшів наприкінці гри, де додаткова глибина, отримана за вузьке вікно, і проста функція оцінки виграш / програш можуть призвести до остаточного результату. Якщо пошуковий запит не вдається, виявити, чи він не вдався,

просто високий(високий край вікна був занадто низький) або низький(нижній край вікна був занадто високий). Це дає інформацію про те, які значення вікна можуть бути корисними при повторному пошуку позиції.

З часом були запропоновані інші вдосконалення, і справді ідея Фальфабети (м'яка альфа-бета-версія) Джона Фішберна майже універсальна і вже включена вище у дещо зміненому вигляді. Фішберн також запропонував поєднання евристичного вбивства та пошуку за нульовим вікном під назвою Lalphabeta ("останній хід з мінімальним вікном альфа-бета-пошуку").

Оскільки мінімаксний алгоритм і його варіанти є по суті глибина першої, стратегія, така як ітеративне поглиблення зазвичай використовується спільно з альфа-бета, щоб можна було повернути досить хороший хід, навіть якщо алгоритм перервано до завершення його виконання. Ще однією перевагою використання ітеративного поглиблення є те, що пошуки на меншій глибині дають підказки щодо впорядкування переміщення, а також невеликі альфа- та бета-оцінки, що обидва можуть допомогти зробити відсічення для пошуку більшої глибини набагато раніше, ніж це було б можливо.

Приклад створення ШІ для шахів

Перший етап: «Візуалізація дошки з генерацією ходів»

На цьому етапі ми будемо використовувати бібліотеки chess.js для створення ходів і chessboard.js для візуалізації дошки. Бібліотека, яка відповідає за генерацію ходів, дозволяє застосовувати всі шахові правила, тому ми можемо розраховувати кожну дію для конкретного розташування фігур. Робота з цими бібліотеками дозволяє сконцентруватися на головному завданні - пошуку та створенні алгоритму, що дозволяє знайти оптимальний хід. Роботу починаємо з написання функції, яка повертає випадковий перебіг зі списку всіх можливих.

```
var calculateBestMove =function(game) {
    //generate all the moves for a given position
    var newGameMoves = game.ugly_moves();
    return newGameMoves[Math.floor(Math.random() * newGameMoves.length)];
};
```

Другий етап: «Оцінка позиції»

Тепер давайте розберемося, у якої сторони є перевага в тому чи іншому положенні. Найпростіший шлях - підрахувати відносну силу фігур на дошці, це можна зробити за допомогою таблиці.

Використовуючи функцію оцінки, ми маємо можливість створити алгоритм, який вибирає хід із максимальною оцінкою.

```
var calculateBestMove = function (game) {

    var newGameMoves = game.ugly_moves();
    var bestMove = null;
    //use any negative large number
    var bestValue = -9999;

    for (var i = 0; i < newGameMoves.length; i++) {
        var newGameMove = newGameMoves[i];
        game.ugly_move(newGameMove);

        //take the negative as AI plays as black
        var boardValue = -evaluateBoard(game.board())
        game.undo();
        if (boardValue > bestValue) {
            bestValue = boardValue;
            bestMove = newGameMove
        }
    }

    return bestMove;

};
```

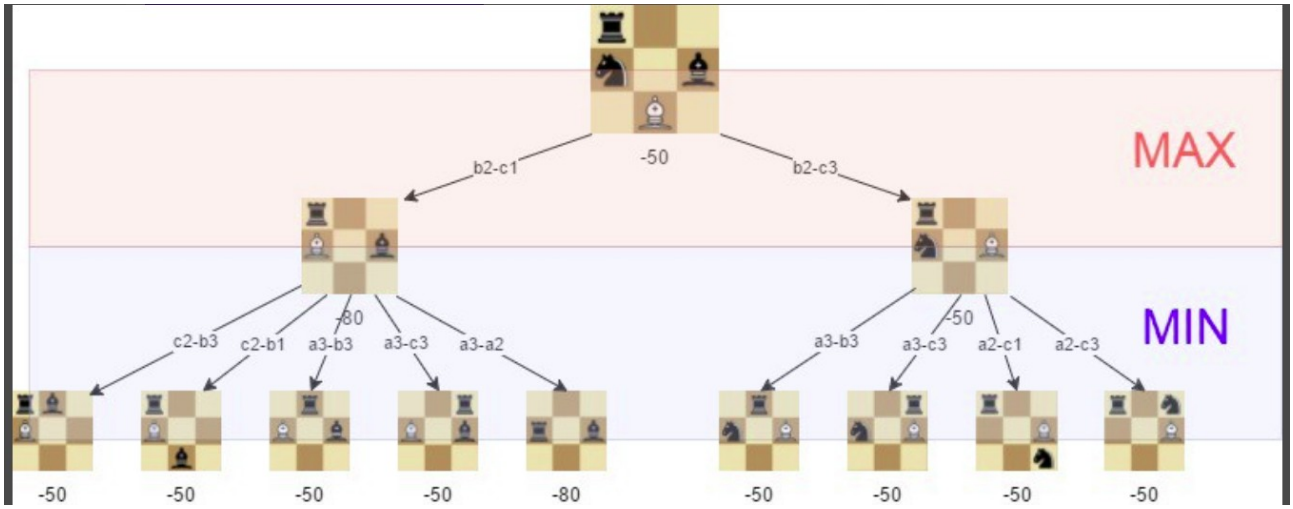
В принципі, рівень колишній, але алгоритм вже може взяти чужу фігуру, коли така можливість є.

Третій етап: «Дерево пошуку з мінімакс»

Після цього ми створюємо дерево пошуку. Тепер програма може вибрати із нього найкращий хід. Це робиться за допомогою мінімакс-алгоритму.

Тут рекурсивне дерево з відображенням усіх можливих ходів аналізується до заданої глибини. Позиція ж оцінюється на листі нашого дерева.

Далі ми повертаємо мінімальне чи максимальне значення нащадка у батьківський вузол. Все залежить від того, хід якої сторони зараз прораховується. Іншими словами, результат максимізується або мінімізується на кожному рівні.



Тут найкращим ходом для білих є b2-c3, оскільки він гарантує, що гравець дістанеться позиції з оцінкою -50.

```
var minimax = function (depth, game, isMaximisingPlayer) {
    if (depth === 0) {
        return -evaluateBoard(game.board());
    }
    var newGameMoves = game.ugly_moves();
    if (isMaximisingPlayer) {
        var bestMove = -9999;
        for (var i = 0; i < newGameMoves.length; i++) {
            game.ugly_move(newGameMoves[i]);
            bestMove = Math.max(bestMove, minimax(depth - 1, game, !isMaximisingPla
            game.undo();
        }
        return bestMove;
    } else {
        var bestMove = 9999;
        for (var i = 0; i < newGameMoves.length; i++) {
            game.ugly_move(newGameMoves[i]);
            bestMove = Math.min(bestMove, minimax(depth - 1, game, !isMaximisingPla
            game.undo();
        }
        return bestMove;
    }
};
```

З мінімакс-алгоритмом наш ШІ вже став розуміти базову тактику шахів. Варто зазначити, що ефективність мінімакс-алгоритму збільшується з глибиною пошуку.

За це відповідає наступний етап.

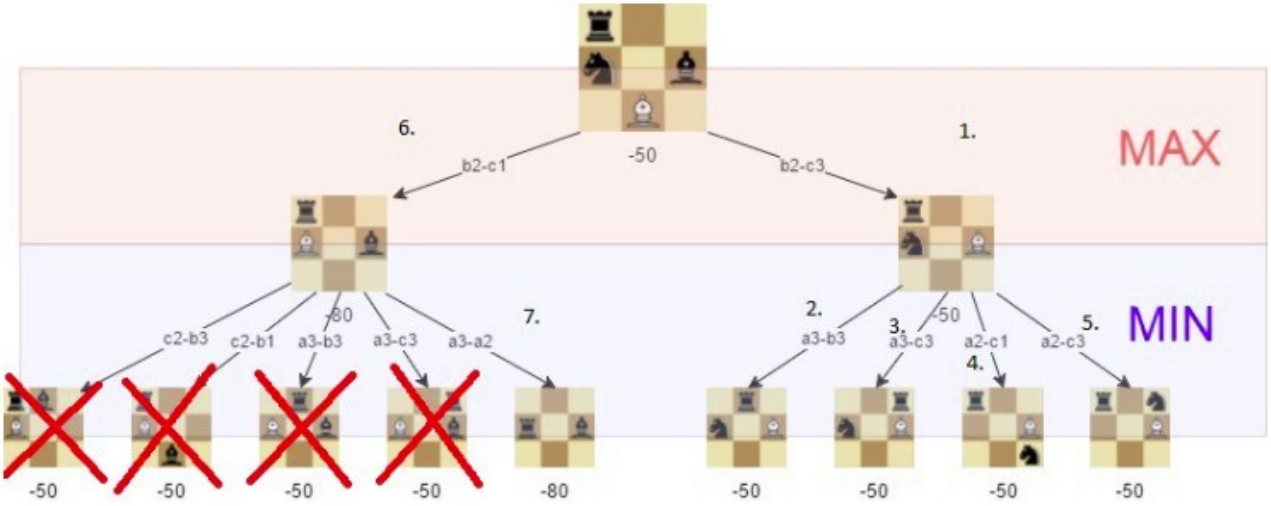
Четвертий етап: «Альфа-бета-відсікання»

Це метод оптимізації мінімакс-алгоритму, що дозволяє ігнорувати деякі гілки в дереві пошуку. І це дозволяє збільшити глибину пошуку, витрачаючи колишній обсяг ресурсів.

Альфа-бета-відсікання ґрунтується на ситуації, коли ми можемо зупинити оцінку певної гілки, якщо виявляється, що новий хід призведе до гіршої ситуації, ніж та, яку ми бачили при оцінці попереднього.

На результат мінімаксу оптимізація не впливає, але все починає працювати швидше.

Цей алгоритм набагато ефективніший у тому випадку, якщо спочатку перевірити шляхи, що ведуть до хороших ходів.

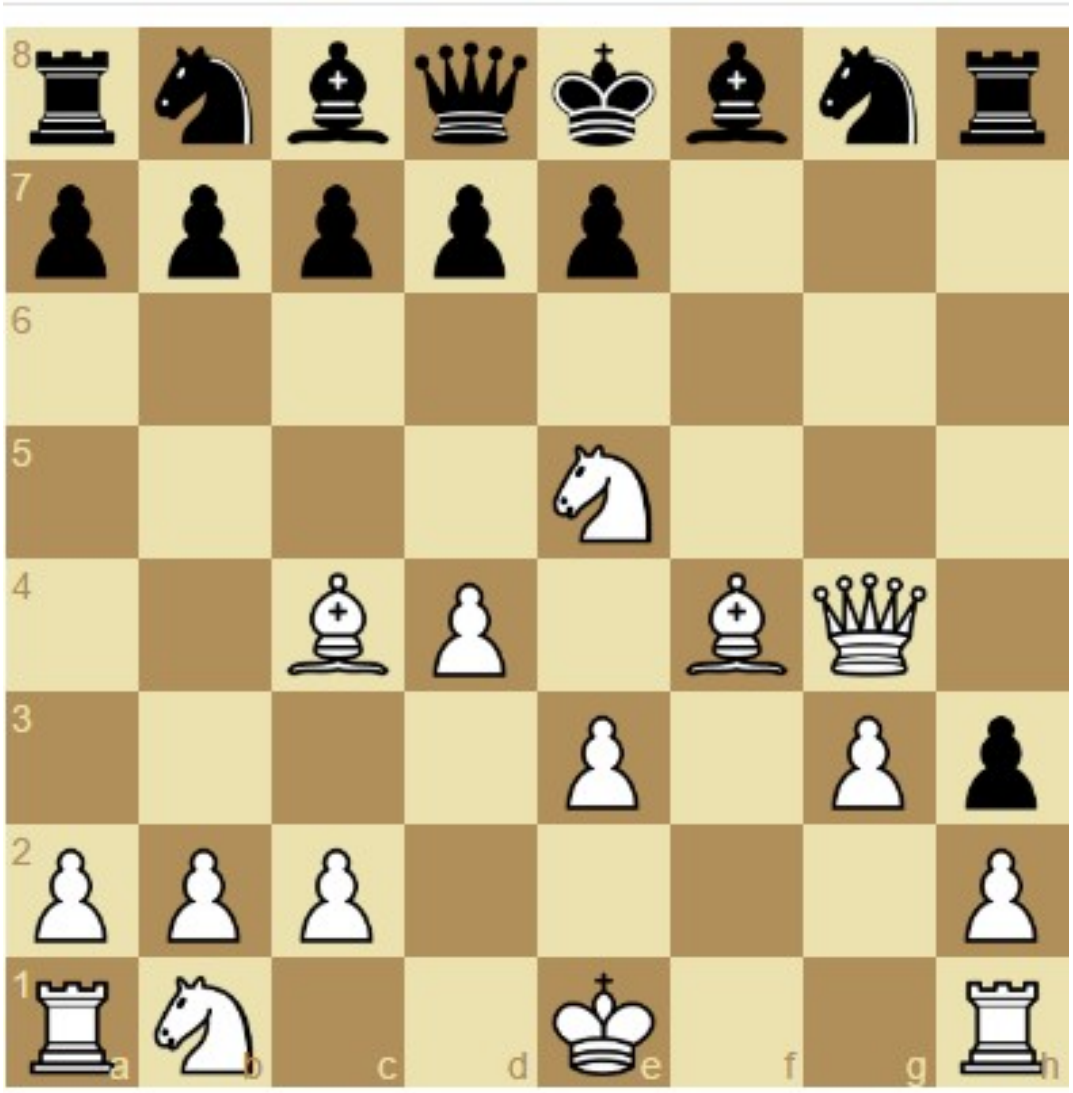


Нова функція minimax() з використанням альфа-бета відсікання

```
function minimax(depth, game, alpha, beta, isMaximisingPlayer) {
    positionCount++;
    if (depth === 0) {
        return -evaluateBoard(game.board());
    }

    var newGameMoves = game.ugly_moves();

    if (isMaximisingPlayer) {
        var bestMove = -9999;
        for (newGameMove of newGameMoves) {
            game.ugly_move(newGameMove);
            bestMove =  Math.max(bestMove, minimax(depth - 1, game, alpha, beta, !isMaximisingPlayer));
            game.undo();
            alpha = Math.max(alpha, bestMove);
            if (beta <= alpha) {
                return bestMove;
            }
        }
        return bestMove;
    } else {
        var bestMove = 9999;
        for (var i = 0; i < newGameMoves.length; i++) {
            game.ugly_move(newGameMoves[i]);
            bestMove = Math.min(bestMove, minimax(depth - 1, game, alpha, beta, !isMaximisingPlayer));;
            game.undo();
            beta = Math.min(beta, bestMove);
            if (beta <= alpha) {
                return bestMove;
            }
        }
        return bestMove;
    }
};
```



	Minimax	Minimax with alpha-beta
Positions	879750	61721

Як бачите, з альфа-бета-відсіканням мінімакс оптимізується, і дуже значно.

Positions — кількість позицій в хвилину, яку рахує ШІ від данної позиції

Кількість позицій, які потрібно оцінити у разі пошуку з глибиною 4 та початковою позицією, яка зображена вище

П`ятий етап: «Поліпшена функція оцінки»

Початкова функція оцінки досить проста, оскільки вона просто рахує ціну фігур, що знаходяться на дошці. Для її оптимізації можна враховувати позицію фігур. Наприклад, якщо розмістити коня у центрі дошки, він стає дорожче — спектр доступних ходів цієї фігури розшириться.

```
[ -3.0, -4.0, -4.0, -5.0, -5.0, -4.0, -4.0, -3.0],
[ -3.0, -4.0, -4.0, -5.0, -5.0, -4.0, -4.0, -3.0],
[ -3.0, -4.0, -4.0, -5.0, -5.0, -4.0, -4.0, -3.0],
[ -3.0, -4.0, -4.0, -5.0, -5.0, -4.0, -4.0, -3.0],
[ -2.0, -3.0, -3.0, -4.0, -4.0, -3.0, -3.0, -2.0],
[ -1.0, -2.0, -2.0, -2.0, -2.0, -2.0, -2.0, -1.0],
[  2.0,  2.0,  0.0,  0.0,  0.0,  0.0,  2.0,  2.0 ],
[  2.0,  3.0,  1.0,  0.0,  0.0,  1.0,  3.0,  2.0 ]
```

```
[ -2.0, -1.0, -1.0, -0.5, -0.5, -1.0, -1.0, -2.0],
[ -1.0,  0.0,  0.0,  0.0,  0.0,  0.0,  0.0, -1.0],
[ -1.0,  0.0,  0.5,  0.5,  0.5,  0.5,  0.0, -1.0],
[ -0.5,  0.0,  0.5,  0.5,  0.5,  0.5,  0.0, -0.5],
[  0.0,  0.0,  0.5,  0.5,  0.5,  0.5,  0.0, -0.5],
[ -1.0,  0.5,  0.5,  0.5,  0.5,  0.5,  0.0, -1.0],
[ -1.0,  0.0,  0.5,  0.0,  0.0,  0.0,  0.0, -1.0],
[ -2.0, -1.0, -1.0, -0.5, -0.5, -1.0, -1.0, -2.0]
```

```
[  0.0,  0.0,  0.0,  0.0,  0.0,  0.0,  0.0,  0.0],
[  0.5,  1.0,  1.0,  1.0,  1.0,  1.0,  1.0,  0.5],
[ -0.5,  0.0,  0.0,  0.0,  0.0,  0.0,  0.0, -0.5],
[ -0.5,  0.0,  0.0,  0.0,  0.0,  0.0,  0.0, -0.5],
[ -0.5,  0.0,  0.0,  0.0,  0.0,  0.0,  0.0, -0.5],
[ -0.5,  0.0,  0.0,  0.0,  0.0,  0.0,  0.0, -0.5],
[ -0.5,  0.0,  0.0,  0.0,  0.0,  0.0,  0.0, -0.5],
[  0.0,   0.0, 0.0,  0.5,  0.5,  0.0,  0.0,  0.0]
```

```
[ -2.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -2.0],
[ -1.0,  0.0,  0.0,  0.0,  0.0,  0.0,  0.0, -1.0],
[ -1.0,  0.0,  0.5,  1.0,  1.0,  0.5,  0.0, -1.0],
[ -1.0,  0.5,  0.5,  1.0,  1.0,  0.5,  0.5, -1.0],
[ -1.0,  0.0,  1.0,  1.0,  1.0,  1.0,  0.0, -1.0],
[ -1.0,  1.0,  1.0,  1.0,  1.0,  1.0,  1.0, -1.0],
[ -1.0,  0.5,  0.0,  0.0,  0.0,  0.0,  0.5, -1.0],
[ -2.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -2.0]
```

```
[-5.0, -4.0, -3.0, -3.0, -3.0, -3.0, -4.0, -5.0],
[-4.0, -2.0,  0.0,  0.0,  0.0,  0.0, -2.0, -4.0],
[-3.0,  0.0,  1.0,  1.5,  1.5,  1.0,  0.0, -3.0],
[-3.0,  0.5,  1.5,  2.0,  2.0,  1.5,  0.5, -3.0],
[-3.0,  0.0,  1.5,  2.0,  2.0,  1.5,  0.0, -3.0],
[-3.0,  0.5,  1.0,  1.5,  1.5,  1.0,  0.5, -3.0],
[-4.0, -2.0,  0.0,  0.5,  0.5,  0.0, -2.0, -4.0],
[-5.0, -4.0, -3.0, -3.0, -3.0, -3.0, -4.0, -5.0]
```

```
[0.0,  0.0,  0.0,  0.0,  0.0,  0.0,  0.0,  0.0],
[5.0,  5.0,  5.0,  5.0,  5.0,  5.0,  5.0,  5.0],
[1.0,  1.0,  2.0,  3.0,  3.0,  2.0,  1.0,  1.0],
[0.5,  0.5,  1.0,  2.5,  2.5,  1.0,  0.5,  0.5],
[0.0,  0.0,  0.0,  2.0,  2.0,  0.0,  0.0,  0.0],
[0.5, -0.5, -1.0,  0.0,  0.0, -1.0, -0.5,  0.5],
[0.5,  1.0, 1.0,  -2.0, -2.0,  1.0,  1.0,  0.5],
[0.0,  0.0,  0.0,  0.0,  0.0,  0.0,  0.0,  0.0]
```

І тепер наш алгоритм грає вже дуже непогано, звісно, порівняно із середнім гравцем.

Висновок

У цій роботі ми розубрали одні із найпростіших методів і алгоритмів для створення ШІ. Зараз штучний інтелект став невід`ємною частиною життя кожної люди і рано чи піздно у кожній галузі буде займати перше місце у списку необхідних елементів підприємств.

Не так давно комп'ютер обіграв і Гаррі Каспарова. Що ж це — безсумнівний успіх?
Про те, як грають комп'ютерні шахісти, написано дуже багато. Розповім зовсім коротко. Вони просто перебирають безліч варіантів.

Комп'ютер нічого не винаходить сам. Всі можливі варіанти підказані справжніми володарями інтелекту - талановитими програмістами і шахістами-консультантами ... Це далеко від створення повноцінного штучного інтелекту.

Література

- https://ua-referat.com/%D0%9F%D1%80%D0%BE%D0%B1%D0%BB%D0%B5%D0%BC%D0%B8_%D1%88%D1%82%D1%83%D1%87%D0%BD%D0%BE%D0%B3%D0%BE_%D1%96%D0%BD%D1%82%D0%B5%D0%BB%D0%B5%D0%BA%D1%82%D1%83
- https://nauka.ua/article/igri-u-yaki-grayut-roboti-navishcho-shi-vchat-gri-u-shahi-j-dota-2-ta-recept-peremog-vid-algoritmiv
- https://uk.upwiki.one/wiki/Transposition_table
- https://www.chessprogramming.org/Main_Page
- http://libfor.com/index.php?newsid=2418
- https://uk.economy-pedia.com/11040689-minimax#menu-1
- https://uk.wikipedia.org/wiki/%D0%9C%D0%B5%D1%82%D0%BE%D0%B4_%D0%9C%D0%BE%D0%BD%D1%82%D0%B5-%D0%9A%D0%B0%D1%80%D0%BB%D0%BE
- https://habr.com/ru/company/pixonic/blog/428892/

Відгук на кваліфікаційну роботу бакалавра на тему:
«Методи поліпшення штучного інтелекту для гри и шахи»
студентки 4-го курсу факультету комп'ютерних наук та кібернетики
Київського національного університету імені Тараса Шевченка
Чередника Олега Сергійовича

Рецензована робота присвячена різним методам теорії ігор. Був представлений метод Мінімакс, та метод Альфа-Бета відсікання. Також було продемонстровано ефективність цих методів. Для поліпшення точності і якості прийнятих рішень штучним інтелектом, було додано додаткові вагові таблиці для кожної з фігур, які були обрані з урахуванням теорії дебюту і мідлшпілю. Була додана можливість регуляції глибини пошуку оптимального ходу, використовуючи Альфа-Бета відсікання час на хід було вагомо зменшено. Чередник Олег створив примітивний інтерфейс для гри в шахи і демонстрації роботи усіх вище зазначених поліпшень і методів для ШІ.

В процесі роботи О.С Чередник проявила гідні знання в галузях приладної математики та комп'ютерних наук і довела професійність. Вважаю, що кваліфікаційна робота студента відповідає вимогам, які висуваються до бакалаврських робіт, і заслуговує на оцінку «добре», а її автор заслуговує на присвоєння кваліфікації бакалавра.

Асистент кафедри обчислювальної математики
факультету комп'ютерних наук та кібернетики
Київського національного університету
імені Тараса Шевченка,
кандидат фізико-математичних наук, асистент В.В. Оноцький

РЕЦЕНЗІЯ

на випускну кваліфікаційну роботу бакалавра Чередника Олега Сергійовича «Методи поліпшення штучного інтелекту для гри в шахи»

У випускній роботі Чередника Олега піднімається питання застосування ші для гри в шахи, і різноматні підходи до поліпшення якості гри і часу витраченого на хід. Детально розглядається методи Мінімакс та Альфа-бета відсікання. Також написана і продемострованна програма, що реалізує ці методи на практиці, мовою програмування було обрано Javascript.

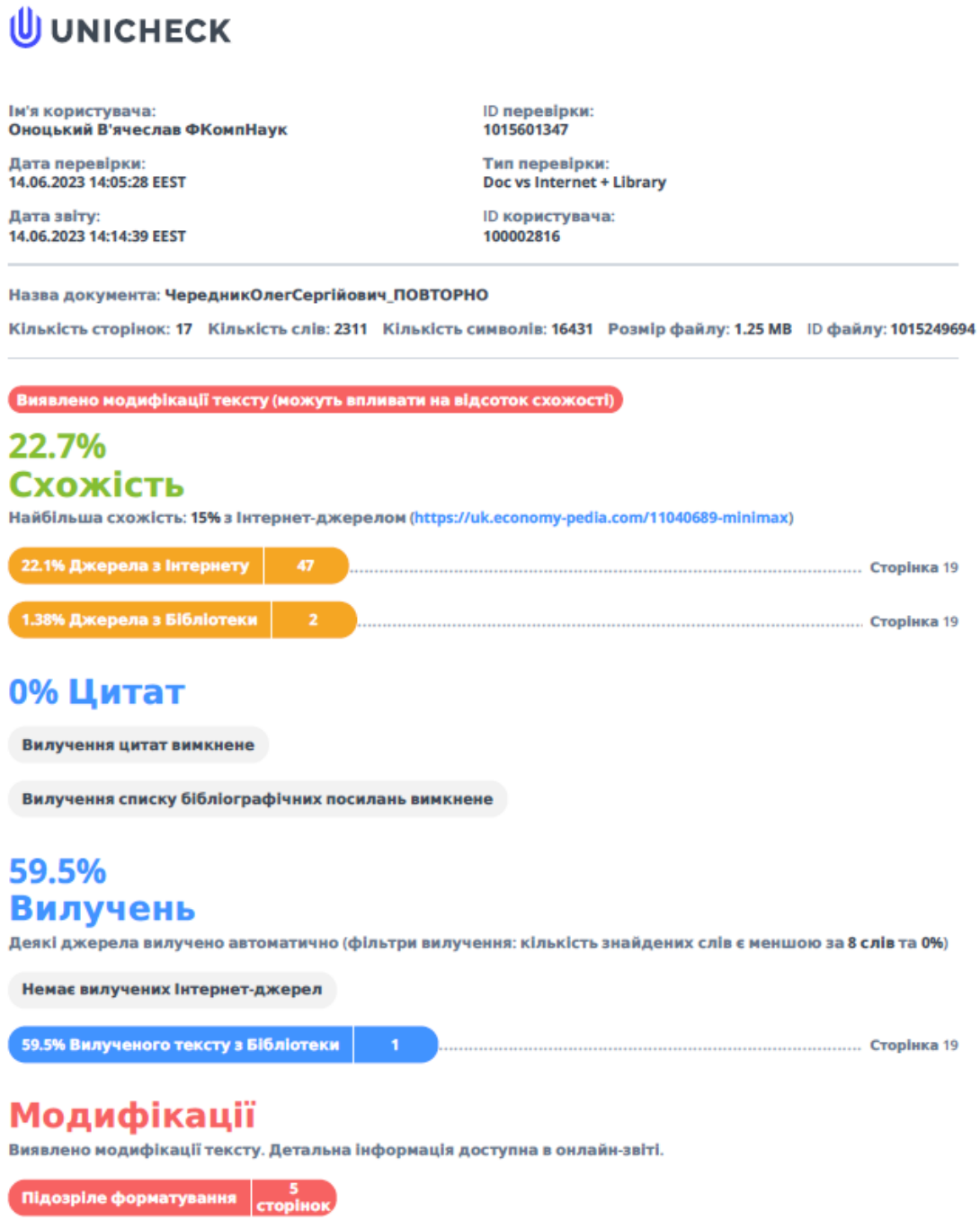
Проблеми, що розв’язано у випускній роботі, є цікавими проте матиматичний апарт, що використовувась є достатньо тривіальним. Робота була виконана вчасно, оформлена згідно вимог, які ставляться до кваліфікаційних робіт бакалавра, й тому її автор заслуговує на присвоєння кваліфікаційного рівня бакалавра, а дипломну роботу пропоную оцінити на “добре”.

Доцент кафедри дослідження операцій
факультету комп’ютерних наук та кібернетики
Київського національного університету імені Тараса Шевченка,
кандидат фізико-математичних наук, доцент Роман ЯКИМІВ

КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ТАРАСА ШЕВЧЕНКА

СИСТЕМА ЗАПОБІГАННЯ ТА ВИЯВЛЕННЯ АКАДЕМІЧНОГО ПЛАГІАТУ

Довідка про оригінальність кваліфікаційної роботи за освітнім рівнем бакалавр



Експертна оцінка роботи науковим керівником :

Робота виконана студентом Чередником самостійно і не містить суттєвих запозичень без відповідних посилань

Науковий керівник: _____ Оноцький В.В.

(підпис) (ПІБ)

Оператор: _____ Оноцький В.В.

(підпис) (ПІБ)