№ 9(23)
2023
НАУКА і ТЕХНІКА СЬОГОДНІ
серії: право, економіка, педагогіка, техніка, фізико-математичні науки

УДК 004.42, 004.891, 519.83

https://doi.org/10.52058/2786-6025-2023-9(23)-624-633

Колос Надія Мирославівна кандидат фізико-математичних наук, доцент кафедри дискретного аналізу та інтелектуальних систем, Львівський національний університет ім. І. Франка, вул. Університетська, 1, м. Львів, 79000, тел.: (032) 239-42-11, https://orcid.org/0000-0001-9710-9667

Фешовець Олег Богданович студент факультету прикладної математики та інформатики, Львівський національний університет ім. І. Франка, вул. Університетська, 1, м. Львів, 79000, тел.: (032) 239-42-11

РОЗРОБКА ВЕБ-ДОДАТКУ ДЛЯ ГРИ В ШАШКИ

Анотація. У таку традиційну гру, як шашки, люди часто грали в минулому у свій вільний час. Ця гра стимулює розвиток логічного мислення та пізнання, щоб спланувати правильну стратегію перемоги в грі. Однак традиційні ігри, в які раніше часто грали всі верстви суспільства, все більше маргіналізуються молодим поколінням. Завдяки розвитку технологій з'являється молоде покоління більш схильне грати в цифрові ігри, ніж у традиційні. Пропоноване рішення цієї проблеми полягає у створенні традиційної гри в цифровій формі; і ця стаття спрямована на розробку ігрової програми в цифрові шашки з використанням машинного навчання. Очікується, що ця гра в цифрові шашки підвищить обізнаність і пристрасть молодого покоління до традиційних ігор.

В статті описано проектування та реалізацію веб-додатку для гри в шашки з використанням React. Для розробки інтелектуального комп'ютерного супротивника був використаний minimax алгоритм, який розглядає всі можливі ходи та їх наслідки на дереві гри. Цей алгоритм дозволяє комп'ютерному супротивнику приймати рішення про найкращий хід, оцінюючи стан гри та його можливі наслідки. Було досягнуто наступних результатів:

- Була створена архітектура додатку, яка базується на компонентній моделі React. Головні компоненти, такі як Play та Checkers, були розроблені для відображення гри та її стану.
- Для маршрутизації в додатку був використаний React Router, що дозволяє переходити між різними сторінками, й спростити розширення додатку в майбутньому.

- Логіка гри та правила перевірки ходів були реалізовані в компонентах, які обробляють ходи гравців, перевіряють їх правильність та оновлюють стан гри відповідно до правил англійських шашок.

У результаті було успішно реалізовано веб-додаток для гри в шашки. Користувачі можуть грати проти комп'ютерного супротивника, який використовує інтелектуальну стратегію прийняття рішень. Додаток надає користувачам зручний інтерфейс та можливість насолоджуватися грою в шашки, а компонентна архітектура дозволяє легко розширити додаток у майбутньому, без масштабного рефакторингу коду.

Ключові слова: гра в шашки, мінімакс алгоритм, експертна система, штучний інтелект, веб-додаток, компонентна архітектура, дерево.

Kolos Nadiia Myroslavivna Candidate of Physical and Mathematical Sciences, Associate Professor at the Department of Discrete Analysis and Intelligent Systems, Ivan Franko National University of Lviv, Universytetska St., 1, Lviv, 79000, tel.: (032) 239-42-11, https://orcid.org/0000-0001-9710-9667

Feshovets Oleh Bohdanovych Student of the Faculty of Applied Mathematics and Informatics, Ivan Franko National University of Lviv, Universytetska St., 1, Lviv, 79000, tel.: (032) 239-42-11

WEB APPLICATION DEVELOPMENT FOR CHECKERS GAME

Abstract. A traditional game like checkers is often played by people in the past during their free time. This game stimulates the development of logical thinking and cognition to plan the right strategy to win the game. However, traditional games that used to be often played by all sections of society are increasingly being marginalized by the younger generation. Due to the development of technology, the younger generation appears to be more inclined to play digital games than traditional ones. The proposed solution to this problem is to create a traditional game in digital form; and this paper aims to develop a digital checkers game application using machine learning. This digital checkers game is expected to increase awareness and passion among generation young towards the traditional games.

In the paper we describe the design and implementation of a web application for playing checkers using React. To develop an intelligent computer opponent, the minimax algorithm was used, which considers all possible moves and their consequences on the game tree. This algorithm allows the computer opponent to decide on the best move by evaluating the state of the game and its possible consequences. The following results were achieved:

• An application architecture based on the React component model was created. Main components such as Play and Checkers were designed to display the game and its state.

• React Router was used for routing in the application, which allows you to move between different pages and simplify the application extension in the future.

• Game logic and move checking rules have been implemented in components that process player moves, check their correctness, and update the game state according to the rules of English checkers.

As a result, a web application for playing checkers was successfully implemented. Users can play against a computer opponent that uses an intelligent decision-making strategy. The application provides users with a user-friendly interface and the ability to enjoy the game of checkers, and the component architecture makes it easy to extend the application in the future, without large-scale refactoring of the code.

Keywords: checkers game, minimax algorithm, recommendation system, artificial intelligence, web application, component architecture, tree, React, React Router.

Постановка проблеми. В сучасному світі дедалі більше буденних для людства речей активно проходять процес діджиталізації. Ще століття тому ніхто не міг собі уявити можливості сучасних комп’ютерів, їх вплив на наше життя й, зокрема, на сферу розваг.

Історія відеоігор бере свій початок у 40-х роках, коли було написано першу ігрову програму, й триває вже більше ніж вісім десятиліть. Ключовим моментом в розвитку сфери комп’ютерних ігор стала шахова програма “Turochamp”, розроблена Аланом Тюрінгом та Девідом Чамперноуном у 1947 році, що мала можливість зіграти повноцінну партію в шахи на рівні гравця-початківця. Комп'ютери тих часів не мали достатніх обчислювальних потужностей.

Першою машиною, яка досягла рівня шахового майстра, була “Belle”, розроблена в 1983 р. Джо Кондоном та Кеном Томпсоном. Офіційний рейтинг Ело був 2250, таким чином, “Belle” була найсильнішою шаховою машиною свого часу. Проте технології рухалися вперед і вже у лютому 1996 року шаховий суперкомп'ютер Deep Blue переміг Гаррі Каспарова з рахунком 4:2, тим самим ставши першим комп'ютером, що переміг чемпіона світу з шахів у турнірних умовах.

На даний момент шашки, на відміну від шахів, не мають доступних та зручних платформ, де користувачі в будь-який час та в будь-якому місці можуть насолоджуватися грою й покращувати свої навички. Розроблений веб-додаток сприятиме розвитку шашкової спільноти та допомагатиме ентузіастам-початківцям знайомитися з грою без додаткових витрат.

Аналіз останніх досліджень і публікацій. Першою програмою, розробленою людиною для гри в шашки проти комп'ютера, є "Chinook" [1]. Це комп'ютерна програма, яка грає в англійські шашки. Вона була розроблена між 1989 і 2007 роками в Університеті Альберти групою під керівництвом Джонатана Шеффера, до складу якої входили Роб Лейк, Пол Лу, Мартін Браянт і Норман Трелоар. У 2007 році він довів, що гра обов'язково закінчується нічиєю між двома ідеальними гравцями. Усі знання Chinook були розроблені його засновниками, а не за допомогою інструментів штучного інтелекту.

До топових програм Google Play та AppStore, які дають користувачу можливість зіграти повноцінну партію в шашки проти комп'ютера за правилами саме англійських шашок, належать відповідно програми [2] та [3]. Ці програми доступні як для телефонів, так і для планшетів, домашніх ПК, розумних годинників та Smart TV. Проте вони мають недоліки. Основним недоліком є те, що хоча ці програми й позиціонуються як безкоштовні, у них неможливо досягти професійного рівня гри без оплати спеціальних додатків.

Також варто відзначити роботу [4], в якій розроблено додаток з використанням мови ClojureScript, що поєднує в собі можливості платформи JavaScript та гнучкість й інтерактивну розробку Clojure. Проте програма має обмежений функціонал, користувачеві недоступний вибір сторони, складності гри чи результат партії.

Метою статті була побудова системи, яка б надавала користувачу можливість зіграти повноцінну партію в шашки проти комп'ютера за правилами англійських шашок.

Тобто гравець гратиме за наступними правилами:

- шашка може ходити тільки вперед;
- шашка не може бити назад;
- гравець обов'язково повинен побити шашку противника, якщо є така можливість;
- гравець за бажанням може продовжити биття, якщо є можливість побити дві і більше фігури суперника;
- шашка що досягла останнього ряду шахівниці стає дамкою;
- дамка може пересуватися на одну поле вперед чи назад.

Система повинна керуватися визначеними вище правилами й надавати користувачеві веб-додатку доступ до наступного функціоналу:

- вибір сторони;
- вибір складності з якою комп'ютер гратиме проти користувача;
- можливість переміщати фігури;
- можливість збивати фігури суперника;

- можливість переглянути результат партії.

Виклад основного матеріалу. ***Основною мовою програмування з допомогою якої реалізовувалася ігрова логіка й взаємодія користувача з нашим веб-додатком є JavaScript [5].***

В якості бібліотеки для розробки веб-додатку було обрано React [6]. Таке рішення було зумовлене можливостями бібліотеки, зокрема віртуальний DOM, який бібліотека React використовує для ефективного оновлення інтерфейсу та компонентна архітектура, що дозволяє розбити інтерфейс на незалежні компоненти, які спростять процес розширення додатка в майбутньому.

JSX (JavaScript XML) - це розширення синтаксису JavaScript, яке дозволяє створювати HTML-подібні структури в коді JavaScript. JSX використовується переважно у бібліотеці React для опису інтерфейсу користувача.

У React обробка подій відбувається за допомогою спеціальних властивостей, які викликаються при виникненні певної події. Ось кілька способів обробки подій в React:

- обробка подій за допомогою атрибутів JSX;
- обробка подій за допомогою методів класу компонента;
- передача параметрів у функцію-обробник.

Умовний рендеринг - це механізм, що дозволяє відображати різний вміст або компоненти в залежності від певних умов. Основні методи умовного рендерингу в React:

- умовний оператор if/else;
- тернарний оператор;
- логічний оператор "&&".

Хуки є функціональними компонентами, що дозволяють використовувати стан та інші функціональності React в компонентах без необхідності використовувати класові компоненти. Основні хуки, що використовувалися в процесі розробки веб-додатку:

- useState;
- useEffect;
- useContext.

React Router - це бібліотека для маршрутизації веб-додатків, що використовують React. Вона дозволяє організовувати навігацію між різними сторінками або компонентами додатку в залежності від URL. React Router надає компоненти і хуки, які допомагають встановити маршрути та керувати навігацією додатку. Основні компоненти React Router включають BrowserRouter, Route, Switch, Link, NavLink, Redirect та інші. Основна ідея

React Router полягає в тому, що ви визначаєте маршрути вашого додатку і пов'язані з ними компоненти, які будуть відображатися при зміні URL. При зміні URL, React Router відображає компонент відповідний до певного маршруту.

В якості додаткових інструментів, які використовувались для розробки дизайну було обрано графічний редактор Adobe Illustrator, який працює з векторною графікою. Тобто графічні елементи дизайну веб-додатку матимуть однаково хорошу роздільну здатність, незалежно від роздільної здатності екрану пристрою, з допомогою якого користувач взаємодіятиме з веб-додатком.

Архітектура веб-додатку. Для розробки веб-додатку було обрано компонентну архітектуру, яка є стала трендовою в сучасній веб-розробці. Компонентна архітектура фокусується на декомпозиції проекту на окремі функціональні або логічні компоненти, які представляють чітко визначені комунікаційні інтерфейси, що містять методи, події та властивості. Архітектура забезпечує вищий рівень абстракції та розділяє проблему на підпроблеми, кожна з яких пов’язана з розділами компонентів. Основною метою компонентної архітектури є забезпечення повторного використання компонентів. Компонент інкапсулює функціональність і поведінку елемента програмного забезпечення в бінарний блок, який можна повторно використовувати та самостійно розгортати.

Реалізація функціоналу. Даний веб-додаток містить у собі 3 сторінки (Home, Play та About), кожна сторінка являє собою вкладену компоненту головної компоненти “App”, що створюється за замовчуванням, проте лише компонента “Play” містить функціонал для взаємодії з користувачем. Для навігації між сторінками додатку використовується бібліотека React Router. Також в “App” міститься компонента “Navbar”, безпосередньо в якій знаходяться посилання для переходу між сторінками.

Ігровий функціонал додатку реалізується в компоненті “Play”. Тут міститься інформація про стан гри й реалізуються функції для старту гри, перегравання, зміни налаштувань гравця та зупинки гри (startGame, restartGame, changeGame та endGame відповідно). Також тут містяться дочірні компоненти для реалізації дошки й переміщення фігур та для реалізації панелі налаштувань гри. Окрім цього дерево компонентів огортає постачальник “AIProvider”, який використовується для передачі пропсів[1] в глибину дерева, для компонентів, що перебувають на різних рівнях ієрархії.

Реалізація ігрової дошки. Ігрова дошка та клітинки реалізуються в компоненті “Checkers”. Шашкова дошка представлена одновимірним масивом

[1] Пропси (props) - це механізм передачі даних в React компоненти.

чисел, де кожне число відповідає відповідній фігурі або пустій клітинці. Клітинки ж реалізовані в компоненті “Tile” та “DummyTile”. Перша використовується для активних (темних) клітинок на дошці, друга використовується для неактивних (світлих) клітинок, на які користувач не може перемістити фігури.

Реалізація логіки гри та правил переходів. Правила за якими ходять фігури обчислюються з допомогою функції validMoves, що є частиною компоненти “AIProvider”. Дана функція обчислює всі можливі ходи для заданої позиції (яка задається масивом фігур та позицією обраної для ходу фігури в масиві) й, в результаті, повертає масив можливих ходів, кожен елемент якого є масивом зі значеннями початкової позиції фігури, кінцевої позиції фігури, масиву з координатами всіх збитих фігур та інформацію, чи перетворилась шашка на дамку.

Далі функція getAllMoves, використовуючи функцію validMoves, обчислює всі допустимі ходи для гравця, й за наявності ходів з биттям позбувається всіх ходів, де биття неможливе.

Вищезгадана функція getAllMoves також використовується в компоненті “Checkers.jsx”, де кожного ходу обчислюються всі наступні можливі ходи. Для того щоб хід був доступний користувачеві, в компоненті “Checkers.jsx” реалізуються функції selectPiece для обрання фігури та movePiece для здійснення ходу, які передаються в якості параметрів для кожної клітини (компоненти “Tile.jsx”).

Черговість та зміна ходів реалізується у вищезгаданому компоненті “Checkers” з допомогою хука useState. Визначення переможця реалізовано з допомогою хука useEffect та вищезгаданої функції getAllMoves. Тобто при поверненні масиву розмірності нуль (відсутність ходів) призначає суперника переможцем. При наявності переможця здійснюється умовний рендеринг компоненти “EndPanel”, яка відображає переможця гри та дозволяє зіграти нову партію без змін налаштувань або відкрити доступ до компоненти “StartPanel” для зміни налаштувань з допомогою функції handleClick, яка передається в якості параметра.

Реалізація штучного інтелекту. Штучний інтелект [7, 8] веб-додатку – це експертна система на основі рекурсивного алгоритму minimax [9], який розглядає всі можливі ходи та їх наслідки на дереві гри. На кожному кроці алгоритм припускає, що гравець намагається максимізувати свої шанси на перемогу, а на наступному ході суперник намагається звести ці шанси до мінімуму, тобто максимізувати власні шанси. Кожен вузол дерева оцінюється за допомогою визначених евристичних функцій оцінки, що при значенні нескінченність означають перемогу гравця, а при значенні мінус нескінченність – суперника (Рис. 1).

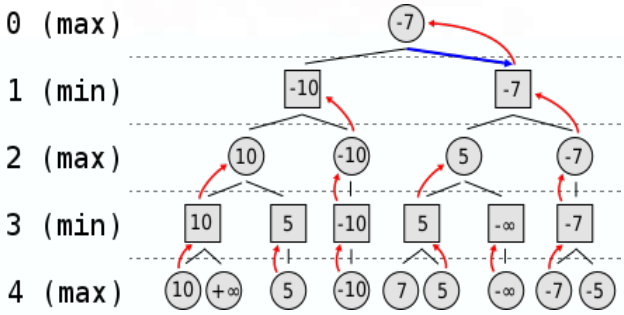

Рис. 1. *Приклад дерева алгоритму minimax*

Реалізація алгоритму. ***Алгоритм реалізується в компоненті "AIContext", як функція minimax (Рис. 2), яка в якості параметрів приймає масив-репрезентацію фігур, глибину дерева й максимізуючого гравця. Також minimax-алгоритм оптимізований з допомогою альфа-бета відсічення, яке відсікає всі ходи при знайденні доказу, що вони гірші за попередні.***

Евристична оцінка кожного вузла проводиться за допомогою функції evaluate (Рис. 3), яка в якості параметрів приймає масив-репрезентацію фігур й повертає різницю вартості фігур гравця та суперника, де вартість дамки вдвічі більша за вартість звичайної шашки.

```
function minimax(pieces, depth, maxPlayer) {
    if (depth === 0 || isWinner(pieces)) {
        const evaluation = evaluate(pieces);
        return { evaluation: evaluation, move: null };
    }

    if (maxPlayer) {
        let alpha = Number.NEGATIVE_INFINITY;
        let beta = Number.POSITIVE_INFINITY;
        let maxEval = Number.NEGATIVE_INFINITY;
        let bestMove = null;
        const moves = allPossibleBoards(pieces, maxPlayer);

        for (let i = 0; i < moves.length; i++) {
            const { evaluation } = minimax(moves[i], depth - 1, false);
            if (maxEval < evaluation) {
                maxEval = evaluation;
                bestMove = moves[i];
            }
            //alpha beta
            alpha = Math.max(alpha, evaluation);
            if (alpha >= beta) {
                break;
            }
        }

        return { evaluation: maxEval, move: bestMove };
    }
```

```
    else {
        let alpha = Number.NEGATIVE_INFINITY;
        let beta = Number.POSITIVE_INFINITY;
        let minEval = Number.POSITIVE_INFINITY;
        let bestMove = null;
        const moves = allPossibleBoards(pieces, maxPlayer);

        for (let i = 0; i < moves.length; i++) {
            const { evaluation } = minimax(moves[i], depth - 1, true);
            if (minEval > evaluation) {
                minEval = evaluation;
                bestMove = moves[i];
            }
            //alpha beta
            beta = Math.min(beta, evaluation);
            if (alpha >= beta) {
                break;
            }
        }

        return { evaluation: minEval, move: bestMove };
    }
}
```

Рис. 2. *Перша та друга частини коду функції minimax*

```
function evaluate(pieces) {
    let blue = 0;
    let red = 0;

    const pieceValues = {
        1: 5,
        2: 5,
        3: 10,
        4: 10
    }

    for (let i = 0; i < pieces.length; i++) {
        const piece = pieces[i];
        if (piece) {
            if (piece & 1) { blue += pieceValues[piece]; }
            else { red += pieceValues[piece] }
        }
    }

    return red - blue;
}
```

Рис. 3. *Функція evaluate*

Далі вищезгадані функції використовуються в компоненті “Checkers” для реалізації ходу – функції aiMovePiece, яка після кожного ходу гравця з використанням useEffect генерує наступний хід для суперника (комп’ютера).

Розробка інтерфейсу. Для розробки макетів та графічних елементів додатку (лого, фігури) використовувався графічний редактор. Розміщення елементів інтерфейсу (Рис. 4) та навігації є інтуїтивним й розроблене за подобою аналогічних шахових платформ. Також дизайн зроблено адаптивним, тобто користувачами додатку можуть бути власники персональних комп'ютерів з різною роздільною здатністю екрану.

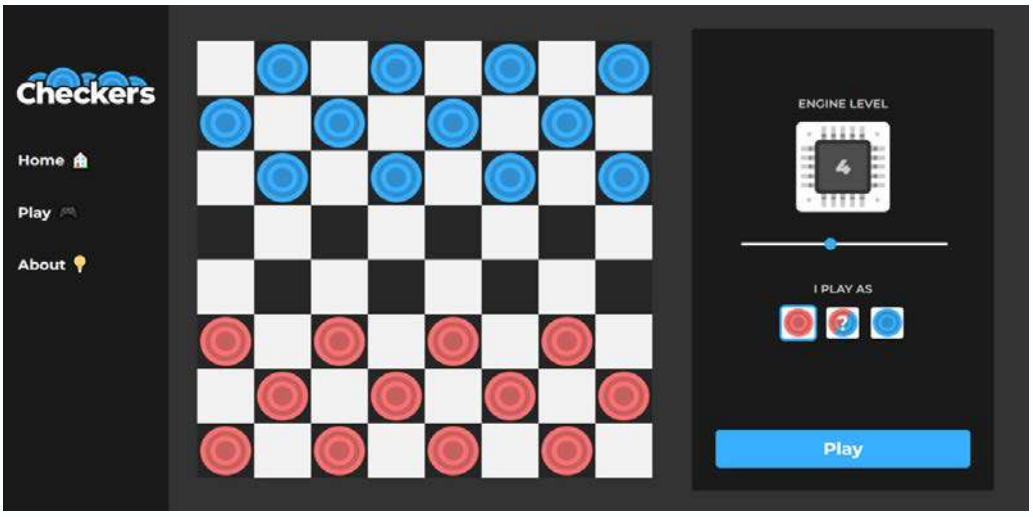

Рис. 4. *Вигляд інтерфейсу додатка*

Висновки. Було успішно реалізовано веб-додаток для гри в шашки з використанням React та minimax алгоритму. Користувачі можуть грати проти комп'ютерного супротивника, який використовує інтелектуальну стратегію

прийняття рішень. Додаток надає користувачам зручний інтерфейс та можливість насолоджуватися грою в шашки, а компонентна архітектура дозволяє легко розширити додаток у майбутньому, без масштабного рефакторингу коду.

Література:

1. Schaeffer, Jonathan. One Jump Ahead: Challenging Human Supremacy in Checkers // Springer, 1997 – 496 стор.
2. Програма "Checkers" на Google Play: https://play.google.com/store/apps/details?id=com.dimcoms.checkers&hl=en
3. Програма "Checkers - Clash of Kings" на AppStore: https://apps.apple.com/tr/app/checkers-clash-of-kings/id1505722728
4. Butenko M. English Checkers Web App made with ClojureScript // GitHub, Вересень 14, 2016, https://github.com/FI-Mihej/English-Checkers-Web-App
5. Документація MDN: https://developer.mozilla.org/en-US/
6. Документація React: https://react.dev/learn
7. Marissa Eppes, Game Theory — The Minimax Algorithm Explained // Towards Data Science, Серпень 7, 2019, https://towardsdatascience.com/how-a-chess-playing-computer-thinks-about-its-next-move-8f028bd0e7b1
8. Don Ross, Game Theory // The Stanford Encyclopedia of Philosophy (Fall 2021 Edition), Edward N. Zalta (ed.), https://plato.stanford.edu/archives/fall2021/entries/game-theory/
9. Ekbia H., Artificial Dreams: The Quest for Non-Biological Intelligence. // Cambridge: Cambridge University Press, 2008. doi:10.1017/CBO9780511802126

References:

1. Schaeffer, Jonathan. One Jump Ahead: Challenging Human Supremacy in Checkers // Springer, 1997 – 496 p.
2. Checkers app on Google Play: https://play.google.com/store/apps/details?id=com.dimcoms.checkers&hl=en
3. "Checkers - Clash of Kings" program on AppStore: https://apps.apple.com/tr/app/checkers-clash-of-kings/id1505722728
4. Butenko M. English Checkers Web App made with ClojureScript // GitHub, Sep 14, 2016, https://github.com/FI-Mihej/English-Checkers-Web-App
5. MDN Documentation: https://developer.mozilla.org/en-US/
6. React Documentation: https://react.dev/learn
7. Marissa Eppes, Game Theory — The Minimax Algorithm Explained // Towards Data Science, Aug 7, 2019, https://towardsdatascience.com/how-a-chess-playing-computer-thinks-about-its-next-move-8f028bd0e7b1
8. Don Ross, Game Theory // The Stanford Encyclopedia of Philosophy (Fall 2021 Edition), Edward N. Zalta (ed.), https://plato.stanford.edu/archives/fall2021/entries/game-theory/
9. Ekbia H., Artificial Dreams: The Quest for Non-Biological Intelligence. // Cambridge: Cambridge University Press, 2008. doi:10.1017/CBO9780511802126