РОЗДІЛ ІІ. ІНФОРМАЦІЙНО-КОМП'ЮТЕРНІ ТЕХНОЛОГІЇ

DOI: 10.25140/2411-5363-2024-3(37)-122-131
УДК 004.89:51-7:51-8

Галина Вікторівна Марчук[1], Ольга Володимирівна Коротун[2], Віталій Леонідович Левківський[3], Микола Олександрович Українець[4]

[1]старший викладач кафедри комп'ютерних наук
Державний університет «Житомирська політехніка» (Житомир, Україна)
E-mail: pzs_mgv@ztu.edu.ua. **ORCID** https://orcid.org/0000-0003-2954-1057. **ResearcherID:** AAD-7514-2022

[2]кандидат педагогічних наук, доцент, доцент кафедри комп'ютерних наук
Державний університет «Житомирська політехніка» (Житомир, Україна)
E-mail: korotun-o@ztu.edu.ua. **ORCID** https://orcid.org/0000-0003-2240-7891. **ResearcherID:** LPP-8040-2024

[3]доктор філософії з інженерії програмного забезпечення, доцент кафедри комп'ютерних наук
Державний університет «Житомирська політехніка»(Житомир, Україна)
E-mail: levkivskyy@ztu.edu.ua. **ORCID** https://orcid.org/0000-0002-1643-0895. **ResearcherID:** GYU-9377-2022

[4]асистент кафедри комп'ютерних наук
Державний університет «Житомирська політехніка»(Житомир, Україна)
E-mail: kkn_umo@ztu.edu.ua. **ORCID** https://orcid.org/0009-0002-1185-491X

ДОСЛІДЖЕННЯ МЕТОДІВ ШТУЧНОГО ІНТЕЛЕКТУ ДЛЯ СТВОРЕННЯ ІНТЕЛЕКТУАЛЬНИХ ІГРОВИХ АГЕНТІВ

Постійний розвиток технологій машинного навчання значно впливає на ігрову індустрію. Штучний інтелект дозволяє створювати все більш складних і розумних ботів, які здатні до навчання та самовдосконалення, що відкриває нові перспективи для розробників ігор, дозволяючи їм створювати більш інтерактивні та захоплюючі ігрові світи. Це дослідження спрямоване на створення інтелектуального шахового агента, який за допомогою методів машинного навчання та алгоритмів пошуку зможе самостійно вдосконалювати свої шахові навички та адаптуватися до різноманітних стилів гри супротивників. Представлена в статті інформація має науково-методичний характер. Очікується, що розглянуті алгоритми можуть бути використані не лише у шахах, а й у інших галузях, що вимагають прийняття рішень в умовах невизначеності. Було проведено дослідження різних алгоритмів машинного навчання, включаючи традиційні методи, і такі як глибинне навчання (зокрема, згорткові та рекурентні нейронні мережі). За результатами аналізу було визначено, що глибинне навчання є найперспективнішим методом для розпізнавання складних шахових патернів та автоматичного формування ефективних стратегій. З метою оптимізації процесу прийняття рішень ботом було обрано систему штучного інтелекту AlphaZero, розроблену компанією DeepMind, яка базується на засадах глибинного навчання. AlphaZero є потужним інструментом для гри в шахи завдяки своїй здатності до самонавчання, глибинним нейронним мережам та ефективному алгоритму пошуку в дереві ходів. Вона відкриває нові горизонти в галузі штучного інтелекту й демонструє, що машини можуть не тільки перевершувати людей у складних інтелектуальних задачах, а й робити це принципово новими способами. AlphaZero не спеціалізується тільки на шахах, вона може бути адаптована для будь-якої гри, в якій ходи гравців чергуються. Це робить її дуже універсальним інструментом для дослідження штучного інтелекту.

Ключові слова: *AlphaZero; гра; відеоігри; ігровий бот; нейронні мережі; глибинне навчання; штучний інтелект*.
Рис.: 2. Бібл.: 11.

Актуальність теми дослідження. Використання штучного інтелекту (ШІ) в розробці ігрових ботів стало ключовим фактором у підвищенні реалістичності та складності сучасних ігор. ШІ дозволяє створювати ботів, які не просто виконують заздалегідь запрограмовані дії, а здатні адаптуватися до стилю гри кожного гравця, ухвалювати динамічні рішення та створювати непередбачувані ситуації, при цьому, ігровий процес стає більш захоплюючим та вимагає від гравців більшої стратегічної гнучкості.

Постановка проблеми. Створення шахового бота, здатного не тільки оптимально прораховувати ходи, а й приймати стратегічні рішення в умовах неповної інформації та постійних змін позицій на дошці, залишається складним завданням. Проблема полягає у виборі та налаштуванні алгоритмів, які зможуть ефективно поєднувати методи пошуку, аналізу й оптимізації для досягнення результатів, які наближають ШІ до рівня гри професійного шахіста. У статті представлено результати науково-методичного дослідження для вирішення цієї проблеми.

Аналіз останніх досліджень і публікацій. Останніми роками ми стали свідками швидкого зростання впливу ШІ на різноманітні сфери, і індустрія ігор не стала винятком. Інтеграція різних моделей ШІ в ігри відкрила нові можливості для розробників,

дозволяючи оптимізувати ігрові процеси та створювати більш захопливий досвід для гравців. Зокрема, аспектам використання штучного інтелекту в іграх присвячено багато публікацій як зарубіжних, так і вітчизняних авторів [1-4].

У статті [5] проведено огляд найпопулярніших алгоритмів ШІ, які застосовуються в іграх, проаналізовано їх вплив на різні аспекти геймплею та спроби зазирнути в майбутнє, щоб зрозуміти, які перспективи відкриваються перед ШІ в ігровій індустрії. Особливу увагу було приділено тому, як ШІ використовується для оптимізації взаємодії гравців з ігровими елементами, що дозволяє створювати більш персоналізований та захоплюючий ігровий досвід.

Авторами статті [6] розглянуто як існуючі методи, так і потенційні, які можуть бути застосовані в майбутньому. Зокрема, авторами проведено аналіз можливості використання машинного навчання для оптимізації розробки, створення інтелектуальних ігрових елементів та автоматизації рутинних завдань. Незважаючи на відносну новизну, цей напрямок демонструє великі перспективи для розвитку індустрії ігор. Сьогодні від штучного інтелекту очікують майже людської досконалості. Нейронні мережі допомагають досягти цього рівня, роблячи штучний інтелект розумнішим. Наприклад, компанія OpenAI використовує їх для створення надзвичайно розумних ігрових ботів. Щоб навчити такі боти грати в складні ігри, часто використовують метод машинного навчання з підкріпленням. Цей метод дозволяє ефективно навчатися в іграх з величезною кількістю можливих ходів.

У статті [7] наведено вступний аналіз наукової літератури, що стосується застосування цих ключових дослідницьких напрямів у сфері відеоігор.

Авторами статті [8] проведено дослідження того, як технології штучного інтелекту ефективно використовуються для вирішення проблем, що виникають під час розробки ігор та інтеграції нових функцій у гру. У цьому дослідницькому документі розглядаються два алгоритми: пошук шляху ботів і прийняття рішень ботом.

Метою дослідження [9] є створення інноваційної 2D-гри, де персонажі, керуються штучним інтелектом, діють автономно в межах своїх можливостей. Для реалізації цього проекту автори обрали ігровий рушій Unity та мову програмування C#. Гра генерує випадкові рівні за допомогою алгоритмів, а гравці можуть відстежувати свої досягнення, такі як кількість переможених ворогів. Це дослідження спрямоване на розширення меж застосуванням штучного інтелекту в розробці відеоігор. Штучний інтелект революціонізує ігрову індустрію, надаючи розробникам потужні інструменти для створення більш реалістичних та інтерактивних ігор. Завдяки ШІ, персонажі стають більш живими та індивідуальними, здатними реагувати на емоції гравців та адаптуватися до їхнього стилю гри. Крім того, ШІ дозволяє створювати більш глибокі та захоплюючі сюжетні лінії, аналізуючи текстові повідомлення гравців та розуміючи їх наміри.

Стаття [10] містить вичерпний огляд штучного інтелекту для використання у складних іграх.

Мета статті. Мета цього дослідження полягає у аналізі алгоритмів для створення шахового бота, який використовує методи та стратегії штучного інтелекту для вирішення задач гри, яка вважається великим викликом для програмістів та вчених, оскільки вимагає не лише обчислювальних зусиль, а й аналізу та стратегічного мислення.

Виклад основного матеріалу. Створення ефективного шахового бота потребує потужних алгоритмів, здатних опрацювати величезну кількість можливих позицій на шахівниці. Хоча класичні методи машинного навчання мають свої переваги в багатьох завданнях, їх застосування у шахах може бути обмеженим через складність гри. Глибинне навчання, зі своєю здатністю виявляти складні патерни в даних, виявилося більш придатним для створення сильних шахових програм. Однак класичні методи все ще можуть використовуватися як допоміжні інструменти або для вирішення конкретних

підзадач. Класичні методи, такі як лінійна регресія, метод опорних векторів, дерева рішень можуть бути використані для створення початкових моделей.

Лінійна регресія, незважаючи на свою простоту, може бути потужним інструментом для створення першої версії шахового бота. Вона дозволить встановити базові залежності між конфігураціями дошки та ймовірністю певного ходу. Хоча шахи – це гра з великою кількістю складних взаємодій, можна спростити її до набору числових характеристик, які можна проаналізувати за допомогою лінійної регресії. Концепція лінійної регресії досить інтуїтивна та легко інтерпретується і яку можна представити рівнянням:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_p x_p + \varepsilon, \quad (1)$$

де y – залежна змінна; β_0 – вільний член, який відповідає значенню y, коли всі незалежні змінні дорівнюють нулю; $\beta_1, \beta_2, \ldots, \beta_p$ – коефіцієнти регресії, які показують, наскільки змінитись Y при зміні відповідної незалежної змінної на одиницю, за умови, що інші незалежні змінні залишаються незмінними; $X_1, X_2, \ldots, X_p$ – незалежні змінні (предиктори), ε – помилка моделі, яка враховує вплив усіх інших факторів, що не включені до моделі. Одним із найпоширеніших підходів до навчання моделей регресії є оптимізація коефіцієнтів за допомогою методу найменших квадратів, який полягає в мінімізації суми квадратів відхилень передбачених значень від фактичних(Mean Squared Error, MSE):

$$MSE = \frac{1}{n}\sum_{i=1}^{n}\left(y_i - \hat{y}_i\right)^2, \quad (2)$$

де y_i – фактичне значення; $\hat{y}_i$ – прогнозоване значення; n - кількість спостережень.

Представимо один із варіантів реалізації лінійної регресії для ігрового боту. Кожну позицію на дошці представлено як вектор чисел. Наприклад, для кожної клітинки можна використовувати три біти: один для визначення кольору фігури, один для типу фігури й один для того, чи клітинка порожня. Хід також кодується як вектор. Можна вказати початкову та кінцеву позицію фігури. Збирається великий набір даних пар «позиція – хід», який може бути даними з партій професійних гравців. Далі будується лінійна модель, яка на основі вектора, що описує позицію, прогнозує вектор, що описує найкращий хід.

Хоча лінійна регресія є потужним інструментом статистичного аналізу, вона не підходить для створення повноцінного шахового бота. Лінійна регресія немає вбудованого механізму для пошуку оптимального ходу, вона може лише прогнозувати числові значення, а не вибирати конкретні дії. Лінійна регресія не враховує довгострокові стратегічні цілі, вона фокусується лише на локальних взаємодіях між фігурами. Для цього потрібні більш складні алгоритми, здатні впоратися зі складністю шахової гри.

Метод опорних векторів (Support Vector Machines, SVM) є потужним алгоритмом машинного навчання, що використовується для класифікації даних. Його основна ідея полягає у знаходженні оптимальної гіперплощини, яка максимально відокремлює точки різних класів. У контексті шахів, SVM може бути застосований для класифікації позицій як виграшних, програшних чи нічийних.

Продемонструємо даний алгоритм для навчальної вибірки, де x – матриця, кожен рядок якої представляє один об’єкт (вектор ознак), y – вектор міток класів (для бінарної класифікації: -1 або 1). Потрібно знайти вектор ваг w і зсув b такі, що представлена лінійна функція максимально відокремлює точки різних класів:

$$f\left(x\right) = w^T \cdot x + b \quad (3)$$

Функціональна відстань від точки x до гіперплощини визначається як:

$$dist = \frac{\|w\|^2}{2}. \quad (4)$$

Оптимальною гіперплощиною є така гіперплощина, де функціональна відстань від найближчих точок (опорних векторів) різних класів до неї є максимальною. Задача SVM зводиться до такого формулювання задачі оптимізації з наступною цільовою функцією та умовою:

$$\min = \frac{\|w\|^2}{2} \tag{5}$$

$$y_i \cdot \left(w^T \cdot x_i + b \right) \geq 1, \quad i = 1, n \tag{6}$$

Щоб навчити шахового бота грати за допомогою алгоритму SVM, потрібно використати велику кількість партій професійних шахістів. Кожна позиція описується вектором ознак, а результат партії виступає як мітка класу. SVM будує гіперплощину, яка оптимально розділяє позиції різних класів у багатовимірному просторі ознак. SVM може бути ефективним для класифікації простих позицій, але для складних позицій можуть знадобитися більш потужні моделі. Крім того, навчання SVM на великих наборах даних може вимагати значних обчислювальних ресурсів. Шахи є динамічною грою, і ситуація на шахівниці постійно змінюється і статична модель SVM може не враховувати всіх нюансів гри.

Дерева рішень та випадковий ліс – це потужні алгоритми машинного навчання, які чудово підходять для класифікації та регресії. Їх можна ефективно застосувати для створення моделі шахового бота, яка прогнозуватиме найкращі ходи. При використанні дерева рішень відбувається побудова моделі у вигляді дерева, де кожен вузол представляє атрибут (ознаку), кожна гілка – значення атрибута, а листя – класи або числові значення. Для цього припустимо, що X – множина об'єктів, A – множина атрибутів, C – множина класів та T – дерево рішень, що можна представити у вигляді наступної функції, яка ставить у відповідність кожному об'єкту x клас c:

$$T\left(x\right) \to c, \quad c \in C \tag{7}$$

Випадковий ліс – це алгоритм для побудови множини дерев рішень та використання їх для прийняття рішення. Він дозволяє підвищити точність побудованої моделі. Представимо B як множину дерев рішень, T_i – і-те дерево рішень, тоді прогноз випадкового лісу для об'єкта x обчислюється за такими формулами:

$$f(x) = \arg\arg\max_c \sum_i I(T_i(x)) = c \tag{8}$$

де $I(T_i(x)) = c$ дорівнює 1, якщо дерево T_i класифікує об'єкт x як клас c та 0 в іншому випадку.

Ключовими аспектами, що характеризують шахову позицію є геометричне розташування фігур на шахівниці, ступінь контролю над центральними полями, наявність безпосередньої загрози королю суперника (шах) або можливості поставити мат. Модель навчається на наборі даних, що містить велику кількість шахових позицій з відповідними ходами, вказавши метрику якості, яку потрібно оптимізувати (наприклад, Precision, F1-score) [11]. Точність (Precision) показує, яка частка з усіх позитивних прикладів, які модель класифікувала як позитивні, насправді є позитивними. У контексті шахового бота це можна інтерпретувати як частину ходів, які бот окреслив як «найкращі», які насправді є «найкращими» з погляду ідеальної гри.

$$\Pr ecision = \frac{TP}{TP + FP} \tag{9}$$

де TP (True Positive) – кількість правильно класифікованих позитивних прикладів; FP (False Positive) – кількість помилково класифікованих негативних прикладів як позитивні (помилкові спрацювання).

F1-score є гармонійним середнім між точністю та повнотою (recall), враховує як точність, так і повноту, і є більш збалансованим показником, ніж кожна з цих метрик окремо.

$$F1-score = 2 \cdot \frac{\Pr ecision \cdot \mathrm{Re} call}{\Pr ecision + \mathrm{Re} call}, \quad (10)$$

де $Повнота(\mathrm{Re} call) = TP/(TP+FN)$; FN (False Negative) – кількість помилково класифікованих позитивних прикладів як негативні (пропущені приклади).

Оцінка якості шахового бота є досить складним завданням, оскільки вона залежить від багатьох факторів, включаючи складність позицій, глибину пошуку, та навіть суб'єктивні оцінки гравців. Проте, точність та F1-score є двома з найпоширеніших та найкорисніших метрик, які можна використовувати для оцінки роботи шахового бота.

Структура дерева рішень схожа на логічні правила, що полегшує інтерпретацію моделі та розуміння, чому було обрано той чи інший хід. Можна використовувати як числові, так і категоріальні дані, що характерно для шахових позицій. Випадковий ліс – це ансамбль дерев рішень, що дозволяє уникнути перенавчання та підвищити точність прогнозів. Дерева рішень та випадковий ліс відносно швидко навчаються, що важливо для створення прототипу бота. Створення сильного шахового бота - це складний процес, який вимагає знань з машинного навчання, шахів та програмування. Однак навіть проста модель на основі дерев рішень та випадкового лісу може грати в шахи на досить високому рівні.

Глибинне навчання, зокрема, згорткові нейронні мережі (Convolutional Neural Network, CNN) та рекурентні нейронні мережі (Recurrent Neural Networks, RNN), виявилися надзвичайно ефективними для складних ігор, таких як шахи. CNN дозволяють моделі вивчати візуальні патерни на шахівниці, тоді як RNN можуть враховувати послідовність ходів. Завдяки своїй здатності до автоматичного виявлення складних ознак, глибинні нейронні мережі досягають значно кращих результатів, ніж класичні методи.

AlphaZero – це комп’ютерна програма, розроблена DeepMind, яка продемонструвала видатні результати в таких складних іграх, як го, шахи та сьогі. В основі AlphaZero лежать глибинні нейронні мережі, які навчаються грати, аналізуючи величезну кількість партій, зіграних із собою. Архітектура мережі AlphaZero використовує глибинну нейронну мережу з кількома блоками згорткових нейронів для обробки ігрового стану. Ця мережа дає змогу моделі виявляти складні патерни та стратегії, які можуть бути недоступні для людини. Модель навчається шляхом гри самої з собою мільйони разів. За кожен хід модель отримує нагороду або покарання, що дає їй змогу поступово покращувати свою стратегію. Цей процес дає змогу моделі знаходити оптимальні рішення в складних ігрових ситуаціях.

AlphaZero використовує алгоритм Монте-Карло з деревом пошуку для вибору найкращого ходу. Модель генерує безліч можливих ходів і оцінює їх за допомогою нейронної мережі. Потім вона вибирає хід, який, за її оцінкою, призведе до найбільшого очікуваного результату. Алгоритм Monte-Carlo Tree Search (MCTS) поєднує в собі елементи пошуку в глибину та випадкового пошуку та дозволяє ефективно досліджувати великі простори станів. Нехай S — простір станів, A(s) — множина допустимих дій у стані s, T: S × A → S — функція переходу, що визначає наступний стан після виконання дії, та R: S → R — функція винагороди, що асоціює зі станом відповідне числове значення.

Тоді процес MCTS можна представити такими етапами:

– ініціалізація – створення кореневого вузла, який представляє початковий стан гри;

– вибір вузла s для розширення за допомогою функції вибору select(s);

– розширення вузла, якщо вузол s не має нерозгорнутих дочірніх вузлів, створити всі можливі дочірні вузли s' = T(s, a) для всіх a ∈ A(s);

– симуляція – вибір випадкової дії *a* зі стану *s′* і виконання її, отримавши новий стан s", цей процес повторюється до кінця гри, допоки отримаємо результат r;

– зворотне поширення – оновлення оцінки вузлів на шляху від s' до кореневого вузла, використовуючи функцію оновлення update(s, r);

– повторення кроків 2-5 до досягнення заданого ліміту часу або ітерацій.

Функція вибору select(s) у більшості випадків використовує один з популярних методів UCB1:

$$UCB1(s) = Q(s) + \sqrt{\frac{2 \cdot ln(N)}{N(s)}}, \quad (11)$$

де *Q(s)* – середнє значення винагороди для вузла s; *N(s)* – кількість разів, коли вузол s був відвіданий та N – загальне число відвідувань кореневого вузла.

Отже, ключовою особливістю AlphaZero є використання глибинної нейронної мережі для оцінки станів. Модель навчається на великій кількості даних, отриманих в результаті самонавчання, і дозволяє алгоритму швидко оцінювати позиції та вибирати оптимальні ходи. MCTS дозволяє ефективно досліджувати великі кількості станів, що особливо важливо для складних ігор, таких як го, шахи та сьогі. Алгоритм можна застосовувати до різних ігор, змінюючи лише функцію переходу та винагороди. UCB1 забезпечує баланс між експлуатацією відомих хороших ходів та дослідженням нових, потенційно кращих варіантів.

Результати дослідження. Для навчання моделі шахового бота потрібен набір даних, які можна зібрати за допомогою шахового рушія Stockfish. Рушій зіграє велику кількість партій сам із собою, обираючи оптимальні ходи. Кожна партія буде записана у файл, забезпечуючи якісний набір даних для подальшого навчання моделі.

Метод mineGames проводить шахові партії з використанням рушія Stockfish детально фіксуючи кожен хід у файл movesAndPositions[n], де n відповідає порядковому номеру зіграної партії.

Лістинг програмного коду методу mineGames:

```
def mineGames(numGames: int):
MAX_MOVES = 500
for i in range(numGames):
currentGameMoves = []
currentGamePositions = []
board = chess.Board()
stockfish.set_position([])
for i in range(MAX_MOVES):
moves = stockfish.get_top_moves(3)
exit
if len(moves) == 0:
print("game is over")
break
elif len(moves) == 1:
move = moves[0]["Move"]
elif len(moves) == 2:
move = random.choices(moves, weights=(80, 20), k=1)[0]["Move"]
else:
move = random.choices(moves, weights=(80, 15, 5), k=1)[0]["Move"]
currentGamePositions.append(stockfish.get_fen_position())
print(move)
print(board)
board.push_san(move)
currentGameMoves.append(move)
```

```
stockfish.set_position(currentGameMoves)
if checkEndCondition(board):
print("game is over")
break
saveData(currentGameMoves, currentGamePositions)
```

Для представлення шахових позицій і ходів можна використовувати масиви чисел замість традиційної нотації. Це спрощує обробку даних і дозволяє застосовувати алгоритми машинного навчання. Бібліотека gym-chess мови Python надає зручний інструментарій для роботи з шаховими даними, використовуючи підхід, заснований на AlphaZero. Для створення глибинної нейронної мережі скористаємось фрейморком PyTorch. Інкапсулюємо дані у контейнери PyTorch, для подальшого навчання моделі.

Проводимо налаштування параметрів для тренування моделі: EPOCHS = 60; LEARNING_RATE = 0.001; MOMENTUM = 0.9. Далі створюємо екземпляр моделі. Використовуємо функцію втрат torch.nn.CrossEntropyLoss(), оптимізатор torch.optim.SGD, який використовує метод градієнтного спуску з моментом для оновлення ваг моделі. Після цього запускається цикл тренування моделі.

Проводимо відстеження найкращого значення втрати на валідаційному наборі (best_vloss). Якщо поточна втрата краща за попередню найкращу втрату, то модель зберігається в файл. Кожні п'ять епох також виводимо середні втрати на тренувальному та валідаційному наборах. Нижче наведено лістинг методу для виконання однієї епохи навчання.

Лістинг програмного коду методу train_one_epoch:

```
def train_one_epoch(model, optimizer, loss_fn, epoch_index, tb_writer):
running_loss = 0.0
last_loss = 0.0
for i, data in enumerate(training_loader):
inputs, labels = data
optimizer.zero_grad()
outputs = model(inputs)
loss = loss_fn(outputs, labels)
loss.backward()
optimizer.step()
running_loss += loss.item()
if i % 1000 == 999:
last_loss = running_loss / 1000 # loss per batch
tb_x = epoch_index * len(training_loader) + i + 1
tb_writer.add_scalar("Loss/train", last_loss, tb_x)
running_loss = 0.0
return last_loss
```

За результатами навчання отримано оптимальну модель (рис. 1).

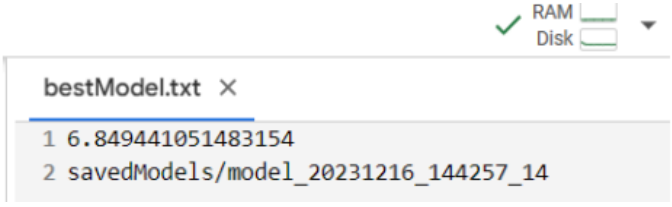

Рис. 1. Оптимальна модель за результатами навчання

Джерело: розроблено авторами.

Після завершення тренування, для оцінки моделі зіграємо тестову партію. Для цього завантажимо отриману модель:

```
saved_model = Model()
model = Model()
f = open("/content/drive/MyDrive/ChessPlayer/savedModels/bestModel.txt", "r")
bestLoss = float(f.readline())
model_path = f.readline()
f.close()
model.load_state_dict(torch.load('/content/drive/MyDrive/ChessPlayer/savedModels/model_20231216_144257_14'))
```

Далі створимо дошку і почнемо партію білими фігурами, зробивши перший хід:

```
board = chess.Board()
moveStr = "e2e4"
move = chess.Move.from_uci(moveStr)
board.push(move)
board
```

Провівши деяку кількість ходів, стає зрозуміло, що модель шахового бота дійсно розуміє як вона має грати і робить правильні ходи. На рис. 2 наведено стан дошки під час гри з моделлю шахового боту. За результатами зіграної партії можна зробити висновок, що модель засвоїла правила гри та діє відповідно них.

Рис. 2. Гра з моделлю шахового бота

Джерело: розроблено авторами.

Модель продемонструвала базове розуміння шахів та здатність робити правильні ходи. Однак, щоб розкрити весь її потенціал і досягти рівня досвідчених гравців, необхідне більш інтенсивне навчання на різноманітних шахових позиціях. Більша кількість тренувальних ітерацій дозволить моделі глибше вивчити шахові стратегії та тактики, що є ключовим для перемоги в складних партіях.

Висновки. Створення потужного шахового бота – це складний процес, який потребує глибокого розуміння як правил гри, так і сучасних методів штучного інтелекту. Було розглянуто різні алгоритми прийняття рішень, такі як лінійна регресія, метод опорних векторів, дерева рішень, методи глибинного навчання. Після ретельного аналізу різних алгоритмів машинного навчання було обрано глибинне навчання як найперспективніший підхід. Глибинні нейронні мережі здатні вивчати складні шахові патерни та автоматично виявляти ефективні стратегії. Щоб оптимізувати процес прийняття рішень ботом було обрано систему штучного інтелекту, розроблену DeepMind - AlphaZero.

Була успішно розроблена та навчена модель шахового бота. Результати тестової партії підтвердили здатність моделі до навчання, але для досягнення високої точності прогнозування необхідне розширення бази навчальних даних. Збільшення різноманітності шахових позицій в навчальній вибірці дозволить моделі ефективніше виявляти складні шахові патерни та покращити якість прийнятих рішень.

Список використаних джерел

1. Hu, Z. Deep learning applications in games: a survey from a data perspective / Z. Hu, Y. Ding, R. Wu, [et al.] // Appl Intell. – 2023. – Vol. 53. – Pp. 31129-31164. DOI: https://doi.org/10.1007/s10489-023-05094-2.

2. Kotkov, D. Gaming Bot Detection: A Systematic Literature Review / D. Kotkov, G. Pandey, A. Semenov // Computational Data and Social Networks. CSoNet 2018. Lecture Notes in Computer Science, Springer, Cham. – 2018. – Vol. 11280. DOI: https://doi.org/10.1007/978-3-030-04648-4_21.

3. Filipović, A. The Role of Artificial Intelligence in Video Game Development / A. Filipović // Kultura Polisa. – 2023. – Vol. 20.3. – Pp. 50-67.

4. Tang, C. Research on Artificial Intelligence Algorithm and Its Application in Games / C. Tang, Z. Wang, X. Sima, L. Zhang // 2nd International Conference on Artificial Intelligence and Advanced Manufacture (AIAM), Manchester, United Kingdom, 15-17 October 2020. – 2020. – Pp. 386-389. DOI: https://doi.org/10.1109/AIAM50918.2020.00085.

5. Xinhe, T. AI applications in video games and future expectations / T. Xinhe // Applied and Computational Engineering. – 2024. – Vol. 54. – Pp. 161-170. DOI: https://doi.org/10.54254/2755-2721/54/20241484.

6. Edwards, G. The Role of Machine Learning in Game Development Domain – A Review of Current Trends and Future Directions / G. Edwards, N. Subianto, D. Englund, [et al.] // 2021 Digital Image Computing: Techniques and Applications (DICTA), Gold Coast, Australia. – 2021. – Pp. 01-07. DOI: https://doi.org/10.1109/DICTA52665.2021.9647261.

7. Skinner, G. Artificial Intelligence and Deep Learning in Video Games A Brief Review / G. Skinner, T. Walmsley // 2019 IEEE 4th International Conference on Computer and Communication Systems (ICCCS), Singapore. – 2019. – Pp. 404-408. DOI: https://doi.org/10.1109/CCOMS.2019.8821783.

8. Singh, K. Artificial Intelligence Based Path Finding and Decision Making in First Person Shooting Game / K. Singh, A. V. Singh, S. K. Khatri, S. Som // 2019 Third International Conference on Inventive Systems and Control (ICISC), Coimbatore, India. – 2019. – Pp. 168-171. DOI: https://doi.org/10.1109/ICISC44355.2019.9036471.

9. Ayas, A. Y. Artificial Intelligence (AI)-Based Self-Deciding Character Development Application in Two-Dimensional Video Games / A. Y. Ayas, H. Aydın, A. Çetinkaya, Z. Güney // Bilgi Ve İletişim Teknolojileri Dergisi. – 2023. – Vol. 5(1). – Pp. 1-19. DOI: https://doi.org/10.53694/bited.1247338

10. Westera, W. Artificial intelligence moving serious gaming: Presenting reusable game AI components / W. Westera, R. Prada, S. Mascarenhas, [et al.] // Education and Information Technologies. – 2020. –Vol. 25. – Pp. 351-380. DOI: https://doi.org/10.1007/s10639-019-09968-2

11. Марчук, Д. К. Методи оцінки ефективності моделей виявлення об’єктів у комп'ютерному зорі / Д. К. Марчук, М. С. Граф // Вісник Херсонського національного технічного університету. – 2023. – № 2 (85). – С. 181-186.

References

1. Hu, Z., Ding, Y., Wu, R. et al. (2023). Deep learning applications in games: a survey from a data perspective. *Appl Intell* 53, 31129–31164. https://doi.org/10.1007/s10489-023-05094-2

2. Kotkov, D., Pandey, G., Semenov, A. (2018). Gaming Bot Detection: A Systematic Literature Review. *In: Chen, X., Sen, A., Li, W., Thai, M. (eds) Computational Data and Social Networks. CSoNet 2018. Lecture Notes in Computer Science*, *11280*. Springer, Cham. https://doi.org/10.1007/978-3-030-04648-4_21.

3. Filipović, A. (2023). The Role of Artificial Intelligence in Video Game Development. *Kultura polisa, 20*, 50-67. https://doi.org/10.51738/Kpolisa2023.20.3r.50f.

4. Tang, C., Wang, Z., Sima, X., Zhang, L. (2020). Research on Artificial Intelligence Algorithm and Its Application in Games. *AIAM.* 386-389. https://doi.org/10.1109/AIAM50918.2020.00085.

5. Tian, X. (2024). AI applications in video games and future expectations. *Applied and Computational Engineering, 54*, 161-170. https://doi.org/10.54254/2755-2721/54/20241484.

6. Edwards, G., Subianto, N., Englund, D., *et al*. (2021). The Role of Machine Learning in Game Development Domain - A Review of Current Trends and Future Directions. *2021 Digital Image Computing: Techniques and Applications (DICTA)*, Gold Coast, Australia, 01-07. https://doi.org/10.1109/DICTA52665.2021.9647261.

7. Skinner, G., Walmsley, T. (2019). Artificial Intelligence and Deep Learning in Video Games A Brief Review. *ICCCS* 404-408. https://doi.org/10.1109/CCOMS.2019.8821783.

8. Singh, K., Singh, A., Khatri, S. K., Som, S. (2019). Artificial Intelligence Based Path Finding and Decision Making in First Person Shooting Game. *ICISC* 168-171. https://doi.org/10.1109/ ICISC44355.2019.9036471.

9. Ayas, A.Y., Aydın, H., Çetinkaya, A., & Güney, Z. (2023). Artificial Intelligence (AI)-Based Self-Deciding Character Development Application in Two-Dimensional Video Games. *Bilgi ve İletişim Teknolojileri Dergisi.* https://doi.org/10.53694/bited.1247338.

10. Westera, W., Prada, R., Mascarenhas, S. et al. (2020). Artificial intelligence moving serious gaming: Presenting reusable game AI components. *Educ Inf Technol 25*, 351–380. https://doi.org/ 10.1007/s10639-019-09968-2.

11. Marchuk D. K., Graf M. S. (2023). Metody otsinky efektyvnosti modelei vyiavlennia obiektiv u kompiuternomu zori [Methods for Evaluating the Effectiveness of Object Detection Models in Computer Vision]. *Visnyk Khersonskoho natsionalnoho tekhnichnoho universytetu - Visnyk of Kherson National Technical University*, *2*(85), 181-186.

Отримано 29.09.2024

UDC 004.89:51-7:51-8

Galyna Marchuk[1], Olha Korotun[2], Vitalii Levkivskyi[3], Mykola Ukrainets[4]

[1]Senior Lecturer of the Department of Computer Sciences
Zhytomyr Polytechnic State University (Zhytomyr, Ukraine)
E-mail: pzs_mgv@ztu.edu.ua. **ORCID** https://orcid.org/0000-0003-2954-1057. **ResearcherID:** AAD-7514-2022

[2] PhD in Pedagogical Sciences, Associate Professor, Associate Professor of the Department of Computer Sciences
Zhytomyr Polytechnic State University (Zhytomyr, Ukraine)
E-mail: korotun-o@ztu.edu.ua. **ORCID** https://orcid.org/0000-0003-2240-7891. **ResearcherID:** LPP-8040-2024

[3]PhD in Software Engineering, Associate Professor of the Department of Computer Sciences
Zhytomyr Polytechnic State University (Zhytomyr, Ukraine)
E-mail: levkivskyy@ztu.edu.ua. **ORCID** https://orcid.org/0000-0002-1643-0895. **ResearcherID:** GYU-9377-2022

[4]Assistant of the Department of Computer Sciences
Zhytomyr Polytechnic State University (Zhytomyr, Ukraine)
E-mail: kkn_umo@ztu.edu.ua. **ORCID** https://orcid.org/0009-0002-1185-491X

RESEARCH OF ARTIFICIAL INTELLIGENCE METHODS FOR CREATING INTELLIGENT GAME AGENTS

The constant development of machine learning technologies has had a great impact on the gaming industry. Artificial intelligence allows the creation of increasingly complex and intelligent bots that are capable of learning and self-improvement, opening up new perspectives for game developers that allow them to create more interactive and immersive game worlds. This research aimed at the creation of an intelligent chess agent that, with the help of machine learning methods and search algorithms, would be able to independently improve its chess skills and adapt to the various playing styles of its opponents. Presented in the article information is a scientific and methodical character. It is expected that considered algorithms can be used not only in chess, but also in other fields that require decision-making under conditions of uncertainty. Various machine learning algorithms have been investigated, including traditional methods and deep learning methods (in particular, convolutional and recurrent neural networks). According to the analysis results, it was determined that deep learning is the most promising method for recognizing complex chess patterns and automatically forming effective strategies. DeepMind's AlphaZero artificial intelligence system, which is based on the principles of deep learning, was chosen in order to optimise the bot's decision-making process. AlphaZero is a powerful chess tool due to its self-learning ability, deep neural networks, and efficient moves tree search algorithm. It opens new horizons in the field of artificial intelligence and shows that computers can outperform humans in complex intellectual tasks, but also do it in fundamentally new ways. AlphaZero is not designed particularly for playing chess, it can be adapted for any game where the moves of the players alternate. It makes this engine a truly universal tool for artificial intelligence research.

Keywords: *AlphaZero; game; video games; game bot; neural networks; deep learning; artificial intelligence.*
Figures: 2. References: 11.

Марчук Г. В., Коротун О. В., Левківський В. Л., Українець М. О. Дослідження методів штучного інтелекту для створення інтелектуальних ігрових агентів. *Технічні науки та технології.* 2024. № 3(37). С. 122-131.