

**КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ  
ІМЕНІ ТАРАСА ШЕВЧЕНКА**

Факультет комп'ютерних наук та кібернетики  
Кафедра теоретичної кібернетики

**Кваліфікаційна робота  
на здобуття ступеня бакалавра**

за спеціальністю 122 Комп'ютерні науки

на тему:

**НАСТІЛЬНІ КОМП'ЮТЕРНІ ІГРИ З ЕЛЕМЕНТАМИ ШТУЧНОГО  
ІНТЕЛЕКТУ**

Виконав студент 4-го курсу  
Руслан РИБАЧОК

\_\_\_\_\_  
(підпис)

Науковий керівник:  
професор кафедри теоретичної кібернетики  
доктор фіз.-мат. наук, професор  
Анатолій ПАШКО

\_\_\_\_\_  
(підпис)

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

Студент

\_\_\_\_\_  
(підпис)

Роботу розглянуто й допущено до захисту на засіданні кафедри теоретичної кібернетики

« \_\_\_\_ » \_\_\_\_\_ 2023 р., протокол № \_\_\_\_

Завідувач кафедри

Юрій КРАК

\_\_\_\_\_  
(підпис)

Київ - 2023

## РЕФЕРАТ

Обсяг роботи 44 сторінки, 22 ілюстрацій, 1 таблиця, 20 джерел посилань.  
ШТУЧНИЙ ІНТЕЛЕКТ, МІНІМАКС, АЛЬФА-БЕТА ОБРІЗКА, ДЕРЕВО  
ПОШУКУ МОНТЕ-КАРЛО, ШАХИ, ПОШУК НАЙКРАЩОГО ХОДУ.

Об'єктом роботи є процес пошуку найкращого ходу в шахах за допомогою штучного інтелекту. Предметом роботи є програмний засіб для пошуку найкращого ходу в настільній комп'ютерній грі шахи.

Метою роботи є аналіз алгоритмів пошуку найкращого ходу та створення програмного засобу для їх порівняння.

Інструменти розробки: мова програмування Python, інтегроване середовище розробки PyCharm.

Результат роботи: було проведено теоретичний аналіз вирішення задачі з пошуку найкращого ходу у настільних комп'ютерних іграх за допомогою штучного інтелекту. На основі цих даних було розроблено програмний засіб для розв'язання задачі різними методами. Окрім того було проведено порівняння швидкості роботи різних алгоритмів.

Створений застосунок може використовувати для навчання та тренування гри в шахи, а також пошуку нових стратегій гри.

## ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАЧЕННЯ	5
ВСТУП	6
РОЗДІЛ 1. ПОНЯТТЯ ШТУЧНОГО ІНТЕЛЕКТУ ТА ОБЛАСТІ ЙОГО ЗАСТОСУВАННЯ	8
1.1 Медицина	9
1.2 Комп'ютерний зір	10
1.3 Робототехніка	11
1.4 Настільні комп'ютерні ігри	12
1.5 ChatGPT	13
РОЗДІЛ 2. МАТЕМАТИЧНЕ ТА ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ НАСТІЛЬНИХ ІГОР	15
2.1 Правила гри в шахи	15
2.1.1 Початкова позиція	15
2.1.2 Пересування фігур	16
2.1.3 Умови закінчення партії	18
2.1.4 Просунуті шахові правила	19
2.1.5 FEN формат запису позиції	20
2.2 Алгоритми для знаходження найкращого ходу в настільних комп'ютерних іграх	21
2.2.1 Алгоритм Minimax	22
2.2.2 Альфа-бета обрізка	24
2.2.3 Дерево пошуку Монте-Карло	25
2.3 Огляд існуючих програмних рішень	28
2.3.1 Stockfish	29
2.3.2 AlphaZero	29
2.3.3 Leela Chess Zero	30
РОЗДІЛ 3. ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ	31

3.1 Підготовка програмного середовища	32
3.2 Встановлення необхідних бібліотек	33
3.3 Реалізація алгоритмів пошуку найкращого ходу	35
3.3.1 Minimax	35
3.3.2 Альфа-Бета обрізка	36
3.3.3 Дерево пошуку Монте-Карло	37
3.3.4 Приклад використання програми	39
3.4 Порівняння алгоритмів	40
ВИСНОВКИ	42
ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ	43

## СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАЧЕННЯ

ШІ – штучний інтелект;

IDE – Integrated Design Environment, інтегроване середовище розробки;

UCB – Upper Confidence Bound, верхня межа впевненості;

MCTS – Monte-Carlo tree search, дерево пошуку Монте-Карло;

## ВСТУП

**Оцінка сучасного стану об'єкта розробки.** Штучний інтелект наразі є надважливим компонентом сучасного життя, що ґрунтується на технологіях та даних. Він широко використовується в таких галузях, як робототехніка, медицина, розпізнавання об'єктів на фото або відеотрансляції в реальному часі, та, звісно ж, в настільних комп'ютерних іграх.

На даний момент існує велика кількість алгоритмів штучного інтелекту, на основі яких уже створено потужні програми для знаходження ходів у таких настільних комп'ютерних іграх, як шахи, го та інші.

**Актуальність роботи та підстави її виконання.** Настільні комп'ютерні ігри — це зручний спосіб дослідження різних алгоритмів, що використовуються для штучного інтелекту. Хоча, існують також і інші способи застосування штучного інтелекту в цій галузі, наприклад він використовується для навчання настільній грі та подальшому процесу тренування людини, пошуку та дослідження нових ігрових стратегій, що можуть допомогти навіть наймайстернішим гравцям.

**Мета й завдання роботи.** Метою кваліфікаційної роботи є створення програмного засобу для порівняння різних алгоритмів штучного інтелекту. Для досягнення цієї мети поставлено такі завдання.

- Дослідити поняття штучного інтелекту.
- Дослідити існуючі алгоритми для пошуку найкращого ходу у настільних комп'ютерних іграх.
- Дослідити існуючі програмні рішення для вирішення задачі.
- Реалізувати алгоритми для пошуку найкращого ходу у настільних комп'ютерних іграх.

**Об'єкт, методи й засоби розроблення.** У роботі будуть досліджені різні алгоритми пошуку найкращого ходу у такій настільній комп'ютерній грі, як шахи. Інструментом для розробки програми було обрано інтегроване середовище

розробки PyCharm на основі мови програмування Python. Додатково були використані наступні бібліотеки та модулі Python:

- python-chess
- pygame
- random
- time

**Можливі сфери застосування.** Розроблений додаток можна буде використовувати для навчання та тренування гри в шахи, а також для аналізу алгоритмів пошуку найкращого ходу в комп'ютерних іграх.

## РОЗДІЛ 1. ПОНЯТТЯ ШТУЧНОГО ІНТЕЛЕКТУ ТА ОБЛАСТІ ЙОГО ЗАСТОСУВАННЯ

Штучний інтелект – ключовий компонент сучасних комп'ютерних наук, який розробляє теоретичні та практичні засади використання систем обробки інформації для вирішення завдань, які раніше були під силу лише людині. Мета штучного інтелекту – дати можливість автоматичним системам імітувати складні психічні функції, притаманні людині, включаючи мислення, судження, мову, розпізнавання образів, навчання, емоції та творчість. Він також має на меті дати можливість цим системам аналізувати зовнішні фактори і враховувати попередні знання для визначення найбільш розумних і ефективних рішень.

Штучний інтелект має глибоку історію розвитку. Значний вплив на саму можливість обговорення ідеї «штучного інтелекту» мав Рене Декарт, який розвинув механістичну гіпотезу, стверджуючи, що тварина є різновидом складної машини. Саме це стало поштовхом до створення перших цифрових механічних обчислювальних машин. Перший механічний цифровий комп'ютер був створений Вільгельмом Шикардом у 1623 році. За ним послідували пристрої, створені Блезом Паскалем і Готфрідом Вільгельмом Лейбніцем[1].

Розвиток механіки у ХІХ столітті дав винаходам новий поштовх до сучасного розуміння штучного інтелекту. У 1830-х роках англійський математик Чарльз Беббідж розробив ідею потужного цифрового калькулятора - аналітичного пристрою, який міг би обчислювати шахові ходи. С. М. Корсаков у 1832 році представив серію «розумних машин» і вперше в історії обчислювальної техніки використав перфоровані картки. Він також запропонував ідею використання наукових інструментів і способів поліпшення розумових здібностей.

У фундаментальній статті, написаній у 1950 році, Алан Тюрінг зробив прогнози щодо розвитку штучного інтелекту. Він розробив знаменитий тест Тюрінга після того, як помітив, що термін «мислення» не має чіткого визначення.

Було б правдоподібно стверджувати, що комп'ютер «мислить», якщо б він міг вести розмову (через телепринтер), яку не можна було б відрізнити від розмови з людиною. Використовуючи цю стислу версію питання, Тьюрінг зміг навести переконливі аргументи на користь правдоподібності «мислячої машини», а його стаття відповіла на всі найпоширеніші критичні зауваження щодо цієї ідеї. Тест Тьюрінга став першим значним внеском у філософію штучного інтелекту.

«Першою програмою штучного інтелекту» вважають Logic Theorist розроблену Гербертом А. Саймоном, Кліффом Шоу і Алленом Ньюеллом. Це була перша програма, спеціально розроблена для здійснення автоматичних міркувань. Вперше її представили на коференції у Дартмундському коледжі, на якому і сформувався поняття «штучного інтелекту».

Зараз ШІ використовується практично у всіх сферах нашого життя, наприклад у таких як:

- Медицина
- Комп'ютерний зір
- Робототехніка
- Настільних комп'ютерних іграх

Окремої згадки заслуговує ChatGPT. Розглянемо використання штучного інтелекту у кожному із цих напрямків конкретніше.

## **1.1 Медицина**

Моделі машинного навчання використовуються в медицині для сканування медичних даних і пошуку інформації, яка допоможе покращити досвід пацієнтів і результати лікування. Штучний інтелект нещодавно досяг значних успіхів у галузі комп'ютерних наук та інформатики, і зараз він стає найважливішим компонентом сучасної охорони здоров'я. Алгоритми ШІ та інші додатки, що працюють на основі ШІ, допомагають лікарям у клінічних умовах і поточних дослідженнях[2].

Допомога в прийнятті клінічних рішень та аналіз зображень –

найпоширеніші сфери застосування штучного інтелекту в медичних установах. Надаючи лікарям миттєвий доступ до інформації або досліджень, які стосуються їхнього пацієнта, системи підтримки прийняття клінічних рішень допомагають їм приймати рішення про лікування, ліки, психічне здоров'я та інші потреби пацієнтів. Технології штучного інтелекту використовуються для аналізу комп'ютерної томографії, рентгенівських знімків, магніто-резонансної томографії та інших зображень у сфері медичної візуалізації з метою пошуку пошкоджень або інших знахідок, які людина-рентгенолог не помітила б.

Загальні стандарти використання ШІ в медицині все ще встановлюються, так само як і результати досліджень і випробувань. Однак з'являється все більше шансів, що ШІ допоможе лікарям, науковцям та пацієнтам, яких вони лікують. Наразі мало хто сумнівається, що ШІ відіграватиме значну роль у цифрових платформах охорони здоров'я, які впливатимуть на сучасну медицину та допомагатимуть їй.

## **1.2 Комп'ютерний зір**

Використовуючи штучний інтелект, комп'ютерний зір дозволяє комп'ютерам видобувати цінні дані з візуальних даних, таких як зображення і відео. Після цього автоматизовані дії виконуються з використанням знань, отриманих завдяки комп'ютерному зору. Комп'ютерний зір дозволяє комп'ютерам «бачити», так само як ШІ дозволяє комп'ютерам «думати». На рис. 1.1 ми можемо чітко побачити, що процеси обробки зображення у людини та комп'ютера дуже схожі[3].

Для того, щоб комп'ютерний зір був дійсно ефективним, потрібна велика база даних. Це пов'язано з тим, що ці рішення багаторазово досліджують дані, поки не отримають всі знання, необхідні для виконання поставленого завдання. Наприклад, для того, щоб навчити комп'ютер розпізнавати здорові посіви, йому потрібно «побачити» тисячі візуальних зображень полів, тварин, посівів та інших відповідних даних. Тільки тоді він зможе точно ідентифікувати різні сорти

здорових культур, відрізняти їх від пошкоджених, оцінювати якість сільськогосподарських угідь, знаходити шкідників та інших тварин серед посівів тощо.

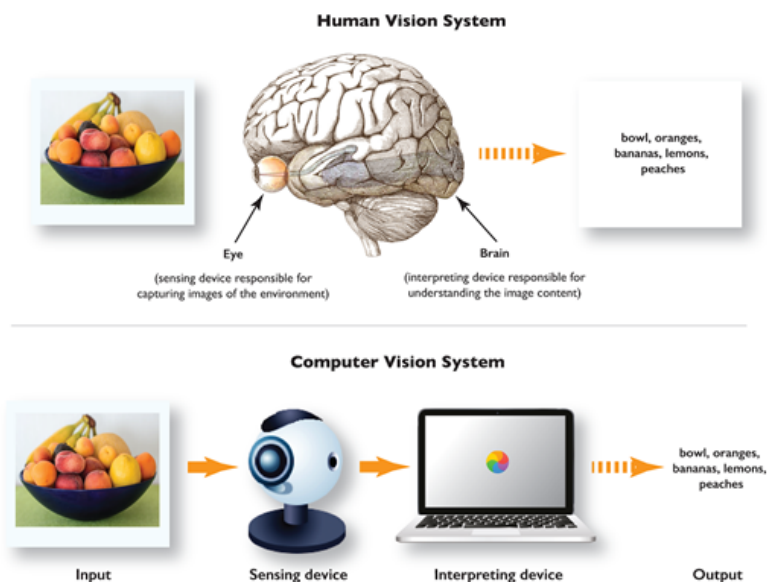


Рисунок 1.1 – Обробка зображення людиною та системою комп'ютерного зору

Комп'ютерний зір в основному базується на двох фундаментальних технологіях: згорткових нейронних мережах і глибокому навчанні.

### 1.3 Робототехніка

Насправді роботизовані системи поєднують у собі низку принципів штучного інтелекту. Ці системи мають широкий спектр функціональних можливостей. Роботи можуть бути оснащені датчиками, рушійними елементами та іншими компонентами для виконання різноманітних завдань, залежно від обставин. Датчики здатні ідентифікувати об'єкти, що перебувають у полі їхнього зору, вимірювати температуру, реагувати на дії користувача тощо. Основні розрахунки здійснюються в режимі реального часу процесорами, вбудованими в

електронні плати. Роботи також можуть змінювати свою поведінку у відповідь на зміни умов навколишнього середовища[4].

Комерційні та промислові роботи широко використовуються сьогодні і виконують багато завдань більш ефективно, точно і надійно, ніж люди. Крім того, їх використовують для виконання деяких завдань, які є надто брудними, монотонними, небезпечними або стомлюючими для людей.

#### **1.4 Настільні комп'ютерні ігри**

За останні кілька років популярність настільних ігор значно зросла. Зростаюче поширення штучного інтелекту в цих іграх є одним з факторів, що сприяють цьому. У минулому настільні ігри часто покладалися на людей-гравців, які вирішували, що робити і як реагувати. Але оскільки ШІ використовується все частіше, з'являється все більше ігор, в які можна грати без будь-якої взаємодії з людиною. Це забезпечує більш захопливий досвід для користувача.

Більшість програм зі штучним інтелектом прагнуть обіграти суперників-людей, використовуючи "розумні" рухи так само, як це робила би людина. Шахи - один з найстаріших прикладів використання настільних ігор зі штучним інтелектом. Перші дослідники шахових комп'ютерів просто хотіли створити пристрій, який міг би перемогти людину-суперника. Деер Влче, шаховий комп'ютер IBM, зробив це у 1997 році проти діючого на той час чемпіону світу із шахмат – Гаррі Каспарова. Сам цей процес ми можемо побачити на рисунку 1.2. Людина справа на рисунку лише переставляє фігури за комп'ютера[5].

Ще одним досягненням ШІ у настільних іграх можна назвати перемогу шахового рушія AlphaGo над Лі Сідолом (другим найкращим гравцем в го) з рахунком 4-1 у 2016 році.



Рисунок 1.2 – Гаррі Каспаров проти Deep Blue

## 1.5 ChatGPT

Випущений у листопаді 2022 року, ChatGPT - це чат-бот зі штучним інтелектом, створений компанією OpenAI. Термін "ChatGPT" поєднує в собі слова "Chat", що вказує на чат-ботовий характер системи, і "GPT", що розшифровується як Generative Pre-trained Transformer і є різновидом великої мовної моделі (LLM). Фундаментальні моделі GPT від OpenAI, а саме GPT-3.5 і GPT-4, слугують основою для ChatGPT, яка була вдосконалена (методом навчання з перенесенням) для розмовних додатків, використовуючи комбінацію технік навчання з учителем і та навчання з підкріпленням[6].

ChatGPT зручний у використанні та може стати у нагоді при написанні, аналізі та виправленні коду, написанні різних текстів типу есе, емейлів та інших. Він також може написати бізнес модель для стартапу. У квітні 2023 року він налічував понад 173 мільйони активних користувача за місяць. На рисунку 1.3 можна побачити приклад розмови з ChatGPT.

 Напиши чому у 2020 році спостерігалось різке збільшення кількості гравців у шахи





 У 2020 році спостерігалось різке збільшення кількості гравців у шахи з кількох причин.     
Пандемія COVID-19 та карантинні обмеження змусили багатьох людей залишатися вдома і шукати способи розваги та розвитку. Шахи стали популярним вибором, оскільки ця гра пропонує інтелектуальне виклик та можливість покращити когнітивні навички.  
Популярність серіалу "Королева Гамбіту", який пропагував шахи, також вплинула на збільшення інтересу до гри у багатьох людей. Крім того, зростання доступності онлайн-платформ для гри в шахи дало можливість людям грати та змагатися з іншими гравцями з усього світу. Всі ці фактори разом сприяли різкому збільшенню кількості гравців у шахи у 2020 році.

Рисунок 1.3 – Приклад розмови з ChatGPT

## **РОЗДІЛ 2. МАТЕМАТИЧНЕ ТА ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ НАСТІЛЬНИХ ІГОР**

### **2.1 Правила гри в шахи**

Для дослідження ШІ в настільних комп'ютерних іграх я обрав шахи. Тому, перш ніж перейти до алгоритмів ШІ, що використовуються для пошуку найкращого ходу, спочатку потрібно розповісти про правила гри в шахи.

#### **2.1.1 Початкова позиція**

Двоє гравців змагаються у грі в шахи на дошці вісім на вісім клітинок. Чорний і білий - класичні світлі і темні кольори, які чергуються на 64 клітинах. При правильній розстановці крайня права клітинка по краю, найближча до кожного гравця, повинна бути білою[7].

У двох горизонтальних рядах (іноді їх називають рангами), найближчих до кожного гравця, розміщуються його фігури. Другий ряд, або другий ряд з точки зору гравця, складається з ряду з восьми пішаків, кожен з яких розміщений на окремій клітині.

Ближній ряд майже симетричний: на двох крайніх клітинах ліворуч і праворуч розміщуються тури, на сусідній з ними внутрішній клітині - коні, і, нарешті, слони.

Король і ферзь розташовуються на двох середніх клітинах ряду. Король займає останню клітину протилежного кольору, а ферзь розташовується на клітині, яка відповідає його кольору (наприклад, чорний ферзь на чорній клітині). В результаті король і ферзь кожного кольору опиняються обличчям один до одного, створюючи ідеальну симетрію для двох гравців.

Повну початкову позицію можна побачити на рисунку 2.1.

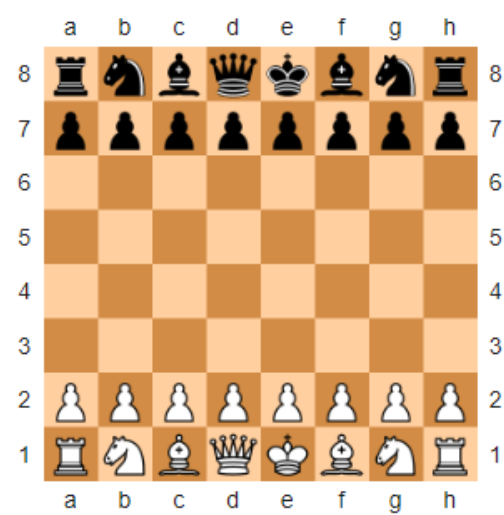


Рисунок 2.1 – Початкова шахова позиція

### 2.1.2 Пересування фігур

Тепер перейдемо до правил гри. Кожен шахіст робить по черзі один хід. Гравці повинні переміщати фігуру під час кожного ходу; вони не можуть пропустити хід. Кожна шахова фігура має певний спосіб пересування, і їй дозволено рухатися тільки цим способом. Розберемо ходи для кожного типу фігур.

Пішаки ходять по прямій лінії на одну клітинку. Вони не можуть рухатися вбік, по діагоналі або по горизонталі. Якщо пішак ще не був переміщений у грі, існує виняток з цього правила. Пішак може просунути на дві клітинки за один хід, якщо він цього ще не зробив. Ці дві клітинки повинні бути порожніми. Гравець також має можливість перемістити фігуру на одну клітинку. Пішак може рухатися по діагоналі тільки тоді, коли він захоплює фігуру супротивника. На кожній з діагональних клітинок ліворуч або праворуч від фігури пішак має можливість захопити фігуру цього гравця. Пішак рухається по діагоналі, щоб замінити захоплену фігуру як частину захоплення.

Тура має можливість переміщатися на будь-яку кількість клітинок вниз по поточному рядку (рангу) або стовпчику. Коли вона переміщується на вже зайняту область, вона може захоплювати фігури протилежного кольору, але не може

проходити крізь фігури свого кольору. Тура в ніякому випадку не може рухатися по діагоналі.

Єдина шахова фігура, яка може рухатися «крізь» інші фігури, "перестрибуючи" через них - це кінь. Під час руху він пересувається через фігури будь-якого кольору і бере фігури звичайним способом, приземляючись на клітину, зайняту фігурою протилежного кольору. Не дозволяється пересуватися на клітину, зайняту фігурою того ж кольору. Коні завжди рухаються у формі літери «Г»: дві клітинки вперед, назад, вліво або вправо, потім одна клітинка по горизонталі або вертикалі, або навпаки: одна клітинка вперед, назад, вліво або вправо, потім дві клітинки по горизонталі або вертикалі, щоб завершити форму літери «Г». Це означає, що кінь завжди може перейти на клітину, яка знаходиться поруч по діагоналі, але не в його поточному ряду (ранзі) або стовпчику. Лицар повинен пройти всю відстань; наприклад, він не може пройти дві клітини прямо, не пройшовши при цьому одну клітину вбік.

На рисунку 2.2 зображено як ходить пішак, тура та кінь, відповідно.

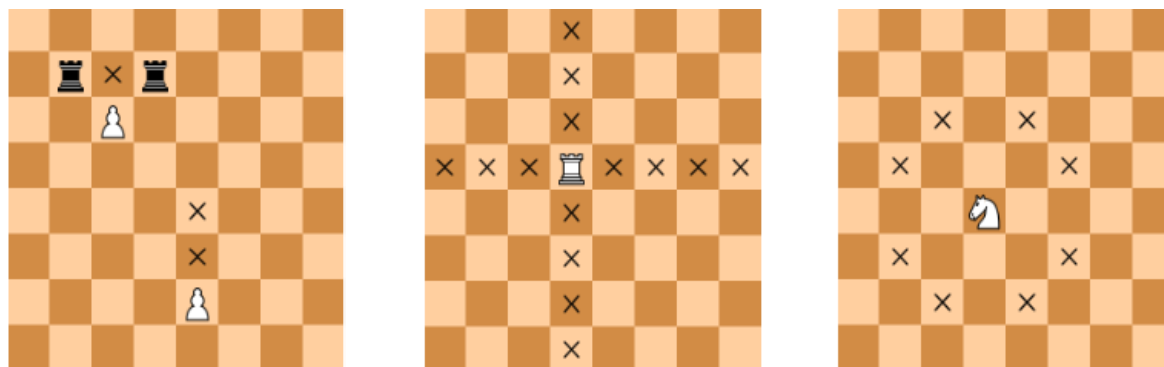


Рисунок 2.2

Слон може переміщатися на будь-яку кількість клітинок по діагоналі, а це означає, що він завжди ходить по діагоналі клітинок, колір яких збігається з кольором клітинки, яку він переміщує. В результаті, на початку гри у кожного гравця є по одному слону, який може ходити на клітинах кожного кольору. Слон не може з якихось причин рухатися ні по горизонталі, ні по вертикалі. Він забирає фігуру протилежного кольору, потрапляючи на свою клітинку. Слон не може рухатися через фігури того ж кольору.

Ферзь може переміщатися на будь-яку кількість клітинок у будь-якому напрямку. Ці дії повинні бути виконані за один хід, рухаючись по одній прямій лінії. (Іншими словами, ви не можете перемістити три клітинки по вертикалі, а потім три клітинки по діагоналі). Ферзь бере фігуру протилежного кольору, переміщуючись на її клітинку, оскільки він не може проходити через фігури одного кольору.

Король, незалежно від того, чи рухається він по вертикалі, горизонталі або діагоналі, переміщується лише на одну клітинку. Король не може зайняти позицію, в якій гравцеві суперника буде поставлено шах або мат.

На рисунку 2.3 зображені можливі ходи слона, ферзі та короля, відповідно

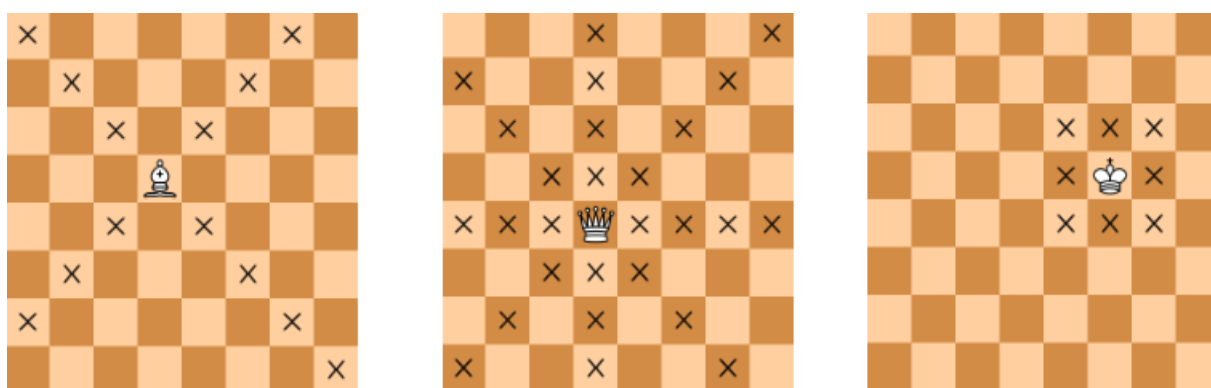


Рисунок 2.3

### 2.1.3 Умови закінчення партії

У шахах виникають три основні позиції, що зв'язані із закінченням гри: шах, мат та пат.

Шах виникає, коли король гравця, чия черга ходити, піддається атаці з боку фігур суперника. Гравець якому поставили шах має зробити хід, щоб уникнути його. Наприклад перемістити короля на безпечне від шахів місце, або забрати фігуру, що атакує короля. Якщо ж виникає ситуація, в якій від шаху не має захисту, то виникає мат і гравець, що поставив мат королю суперника – перемагає. Існують також і інші способи закінчення партії, наприклад при оголошенні одного

із гравців переможеним або через пат (тоді визнається нічия). Пат виникає у ситуації, коли правила гри не дозволяють зробити хід гравцю, а фігури суперника не атакують короля[7].

#### 2.1.4 Просунуті шахові правила

Тепер, коли ми розглянули як виглядає початкова позиція, як рухаються фігури та умови закінчення партії, можемо перейти до правил, що можуть бути невідомими усім гравцям: рокірування, перетворення та взяття на проході.

Король і тура виконують унікальний маневр рокіровки. Для того, щоб цей хід був законним, необхідно виконати три вимоги: король і тура не повинні були рухатися раніше; король не повинен був бути під атакою раніше, а поля між королем та турою не повинні бути під атакою під час рокіровки; поля між королем та турою повинні бути пустими. Король повертається, переміщуючись на дві клітинки в горизонтальному напрямку. Після рокіровки короля ставлять через одне поле по горизонталі в ту сторону, яку відбувається рокіровка, а туру – на те місце, що пропустив король.

Один з найвідоміших шахових ходів - взяття на проході, що з французької перекладається як «мимохідь». Взяття на проході виникає після того, як пішак рухається на дві клітинки вперед після необов'язкового початкового ходу. У цьому випадку на наступному ході суперник може перемістити свого пішака по діагоналі на клітину, через яку пройшов пішак, що рухається, фактично забравши пішака так, ніби він перемістився лише на одну клітину. Гравець, який має шанс захопити пішака, втрачає цей шанс, якщо взяття на проході не оголошується і не робиться наступним ходом суперника.

Пішак стає турою, конем, слоном або ферзем, якщо він переміщується на протилежний край дошки - ряд (ранг), який є найвіддаленішим від гравця, що ним керує. Нова фігура замінює пішака на клітині, яку він займає в даний момент, і рухається відповідно до вказівок, характерних для цієї фігури.

## 2.1.5 FEN формат запису позиції

FEN (Forsyth–Edwards Notation) формат – один із найпопулярніших способів запису шахової позиції. FEN формат не відображає попередні ходи. Він складається із 6 полів. Приклад запису початкової позиції:

*rnbqkbnr/pppppppp/8/8/8/8/PPPPPPP/RNBQKBNR w KQkq - 0 1.*

Перше поле FEN формату відображає розташування фігур на шахівниці. Починаючи з восьмого рангу і закінчуючи першим, він починається з пояснення вмісту кожного поля. Поля розташовані від першої вертикалі до восьмої для кожного рангу. Літери нижнього регістру відображають розташування чорних фігур, а верхнього – білих. Цифра відповідає за кількість пустих полів у горизонталі. Розглянемо це поле на прикладі

*r1bk1nr/p2p1pNp/n2B4/1p1NP2P/6P1/3P4/P1P1K3/q5b1* (рис. 2.4).

Починаємо з 8 горизонталі. Першою розташована чорна тура, потім йде пропущене поле, чорний слон, знову пропущене поле й так далі поки ми не дійдемо до скісої риски. Дійшовши до неї, опускаємось на ранг нижче і знову повторяємо процес заповнення.



Рисунок 2.4 – Пояснення запису першого поля в FEN форматі

Друге поле відповідає за те, кому належить наступний хід: w – хід білих, b – хід чорних.

Третє поле відповідає за рокіровку. Якщо жодна зі сторін не може рокіруватися, ставиться «-». Якщо якась зі сторін може рокіруватися то ставиться одна чи більше літера: «K», якщо білий король може рокіруватися в праву сторону і «Q», якщо в ліву; аналогічно для літер «k» та «q» для рокіровки чорного короля.

Четверте поле свідчить про можливість взяття на проході. Якщо стоїть «-», то взяття на проході немає, якщо записано хід пішака, то це просунуте правило можна застосувати.

У п'ятому та шостому полі записують півходи з останнього взяття фігури та число повних ходів, відповідно.

## **2.2 Алгоритми для знаходження найкращого ходу в настільних комп'ютерних іграх**

Комп'ютерні шахові програми, незалежно від того, чи реалізовані вони в апаратному чи програмному забезпеченні, використовують інші стратегії, ніж люди, для вибору своїх ходів. Вони використовують евристичні методи для побудови, пошуку та оцінки дерев, що представляють послідовності ходів з поточної позиції, і намагаються здійснити найкращу з можливих послідовностей під час гри. Ці дерева зазвичай мають від тисяч до мільйонів вузлів і є досить великими. Такий підхід добре працює, оскільки сучасні комп'ютери можуть обробляти від десятків тисяч до сотень тисяч вузлів за секунду і більше, а евристики розширення та скорочення скорочують дерево до найбільш релевантних вузлів.

Існує велика кількість алгоритмів ШІ, що застосовуються для пошуку найкращого ходу. Розглянемо декілька з них.

### 2.2.1 Алгоритм Minimax

Minimax - це правило прийняття рішень, що використовується в штучному інтелекті, теорії прийняття рішень, теорії ігор, статистиці та філософії для зменшення потенційних втрат у найгіршій ситуації.

Minimax використовує оцінку позиції, тому для початку зрозуміємо як можна отримати це значення у шахах. Для пошуку з обмеженою глибиною оцінка передбачає зважування та підрахунок фігур на кожній стороні. Позиція незбалансована, і білі в небезпеці, якщо чорні мають ферзя, а білі - ні. Білі, швидше за все, мають збалансовану позицію, якщо вони втратили три пішаки, але також мають додаткового слона. Залежно від того, скільком пішкам еквівалентна кожна фігура, ця функція оцінки присвоює їм певну кількість балів. Тому, можемо присвоїти пішаку 10 балів, коню та слону по 30 балів, турі – 50 балів та 90 балів ферзі. Для білих фігур ці значення будуть додатні, а для чорних – від’ємні (для наглядності можна переглянути рисунок 3.1). Оцінка позиції – це сума всіх балів фігур, наявних на шахівниці. Алгоритм, який використовує цю функцію оцінки, буде мати встановлені загальні принципи. Загальне правило полягає в тому, що брати фігури, особливо тури і ферзі, вигідно, а віддавати важливі фігури для знищення пішаків - негативно.










	10		-10
	30		-30
	30		-30
	50		-50
	90		-90

Рисунок 3.1 – оцінка фігур

Тепер можемо перейти до розгляду роботи алгоритму Minimax. В алгоритмі один учасник називається максимізатором, а інший – мінімізатором. На ігровому полі виставляється оцінка, і при цьому один гравець намагатиметься вибрати стан з найбільшою кількістю балів, а інший - з найменшою. Іншими словами, максимізатор намагається набрати якомога більше очок, тоді як мінімізатор намагається отримати якомога менше очок, роблячи зустрічні ходи. Для алгоритму ми обираємо глибину – кількість ходів після заданої позиції. З точки зору технології, ми починаємо з кореневого вузла і обираємо найкращий дочірній вузол. Для роботи алгоритму ми оцінюємо зовнішні вузли (листки) і виставляємо їм оцінку. В результаті ми рекурсивно поширюємо оцінки назад, аж до кореневого вузла[8].

Наведемо приклад використання алгоритму (рисунок 3.2). Для початку побудуємо дерево із глибиною 2. Потім, за допомогою оцінки позиції присвоюємо значення кожному зовнішньому вузлу. Рухаємося вгору по дереву з листків. Так як у нас гравець, що намагається обрати стан із мінімальною кількістю очок, присвоюємо йому найменше зі значень його дочірніх вузлів (для лівого вузла у нас це 3, для правого – 4). Аналогічно рухаємося вгору, лише з тою відмінністю, що гравець тепер намагається максимізувати результат (для нас це значення рівне 4). Ми досягли кореневого вузла та можемо обрати хід на результаті вибору в останньому кроці.

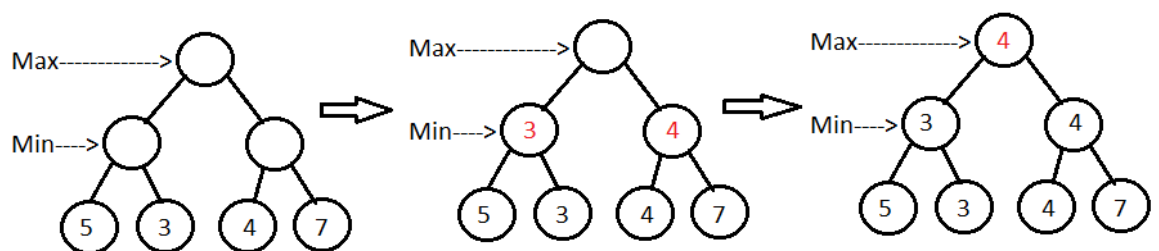


Рисунок 3.2 – приклад застосування алгоритму Minimax

Найголовніша перевага алгоритму Minimax в тому, що при правильному підході до оцінки позиції, він з великою точністю знайде найкращий хід для заданої глибини.

Основним недоліком Minimax алгоритму є те, що він стає дуже повільним у

складних іграх, таких як шахи, го тощо. Цей тип ігор має величезний фактор розгалуження, і гравець має багато варіантів для прийняття рішення. Це обмеження алгоритму Minimax можна покращити за допомогою альфа-бета обрізки, яку ми обговоримо наступною.

### 2.2.2 Альфа-бета обрізка

Альфа-бета обрізка - це модифікована версія алгоритму Minimax, яка націлена на оптимізацію цього алгоритму.

Кількість станів гри, які має розглянути алгоритм Minimax є дуже великою. Тому існує метод, за допомогою якого без перевірки кожного вузла дерева гри ми можемо обчислити правильне мінімаксне рішення, і цей метод називається обрізанням. Вона включає в себе два порогових параметри альфа і бета для майбутнього розширення, тому вона називається альфа-бета обрізанням[9].

Альфа-бета обрізка може бути застосована на будь-якій глибині дерева, а іноді вона обрізає не тільки листя дерева, але й цілі піддерева.

Двопараметричний алгоритм можна визначити наступним чином:

- Альфа: Найкращий (з найбільшим значенням) вибір, який ми знайшли на даний момент у будь-якій точці на шляху Максимізатора. Початкове значення альфа дорівнює  $-\infty$ .
- Бета: Найкращий (з найменшим значенням) вибір, який ми знайшли в будь-якій точці на шляху Мінімізатора. Початкове значення бета дорівнює  $+\infty$ .

Альфа-бета обрізка до стандартного алгоритму Minimax повертає той самий хід, що і стандартний алгоритм, але видаляє всі вузли, які насправді не впливають на остаточне рішення, але роблять алгоритм повільним. Таким чином, відсікаючи ці вузли, ми робимо алгоритм швидшим.

Розглянемо приклад зображений на рисунку 3.3. Говорячи простими словами, В – мінімайзер. Саме тому для нього обирається найменше зі значень

вузлів D та E. Попередньо обчисливши значення для вузла D (максимальне з 2 та 3), ми отримали 3. Це означає, що для оптимального ходу значення вузла B повинно бути 3. Оцінюємо першого з дочірніх вузлів E. Отримали значення 5, що є більшим ніж 3. Так як ми прагнемо максимізувати оцінку з дочірніх вузлів для E, значення буде точно більшим чи рівним 5, що вже є неприйнятним для нас. Наступні ходи для вузла E можна навіть не розглядати. Аналогічно відкидаємо піддерево G. Так як нам потрібно максимізувати значення вузлів B та C, але мінімізувати F та G. Обчисливши значення для вузла F, можемо дійти до висновку, що мінімальним значенням для F та G буде число, що 1, що є неприйнятним для нас, так як на потрібно число, що буде більше 3. Саме тому ми і відкинули піддерево G.

Основна перевага залишається все тою ж, що й для Minimax алгоритму, але тепер для пошуку ходу витрачається менше часу.

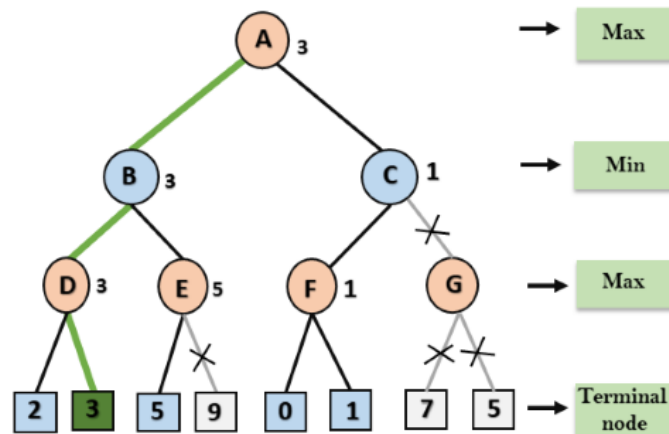


Рисунок 3.3 – Приклад використання Альфа-Бета обрізки

### 2.2.3 Дерево пошуку Монте-Карло

Під час використання альфа-бета-обрізки ми застосували функцію

оцінювання позиції для скорочення кількості аналізованих ігрових станів. Однак оцінити позицію в грі шахи, го, та інших – дуже і дуже важко. Пошук по дереву методом Монте-Карло дає змогу оцінити ігровий стан без будь-яких знань про стратегію гри. Замість застосування специфічного евристичного методу алгоритм MCTS симулює випадкові ігри для оцінки конкретного ігрового стану.

Перейдемо до самого алгоритму MCTS. У MCTS вузли є будівельними блоками дерева пошуку. Ці вузли формуються на основі результатів ряду симуляцій. Процес дерева пошуку Монте-Карло можна розбити на чотири окремі кроки, а саме: відбір, розширення, симуляція та зворотне розширення. Кожен з цих кроків детально пояснюється нижче.

При відборі ми використовуємо політику дерева для побудови шляху від кореня до найперспективнішого зовнішнього вузла (листка). Листок – це вузол, який має недосліджений дочірній вузол. Деревоподібна політика – це інформована політика, яка використовується для вибору дій (вузлів) у сніговій шапці (дослідженій частині ігрового дерева на противагу величезній недослідженій нижній частині). Одним з важливих аспектів такої політики є баланс між розвідкою та експлуатацією, що є постійною темою в ШІ, особливо в навчанні з підкріпленням. В нашому алгоритмі ми будемо використовувати політику засновану на UCB (Upper Confidence Bound). Точніше, кожен вузол має пов'язане з ним значення UCB, і під час вибору ми завжди обираємо дочірній вузол з найбільшим значенням UCB. Формула для обчислення цього значення:

$$S_i = \frac{w_i}{n_i} + c \cdot \sqrt{\frac{\ln N_i}{n_i}}$$

- $S_i$  – значення нашого вузла;
- $w_i$  – кількість виграшних симуляцій після  $i$ -го ходу;
- $n_i$  – кількість відвідувань вузла  $i$ ;
- $N_i$  – загальна кількість відвідувань батьківського вузла;
- $c$  – константа (іноді температура);

Для константи зазвичай обирається значення 1,41. Вона відіграє важливу

роль у балансі між експлуатацією та розвідкою. За більшого значення  $c$  ми витратимо більше часу на відвідування найменш досліджених вузлів (розвідка). За меншого значення  $c$  ми зосередимося на отриманні точної оцінки найбільш перспективного вузла (експлуатація)[10].

На кроці розширення ми просто випадковим чином вибираємо недосліджений вузол листкового вузла.

В наступному кроці проводиться симуляція гри, найчастіше з вибором випадкових ходів, аж до моменту, поки не буде досягнуто закінчення гри. В результат може бути записано 1 при виграші, -1 при програші та 0 при нічії.

При зворотньому розширенні ми оновлюємо інформації, отриману від симуляції для всіх вузлів від якої була проведена симуляція і аж до кореневого вузла.

Для наглядного прикладу роботи алгоритму, можна подивитись на рисунок 3.4.

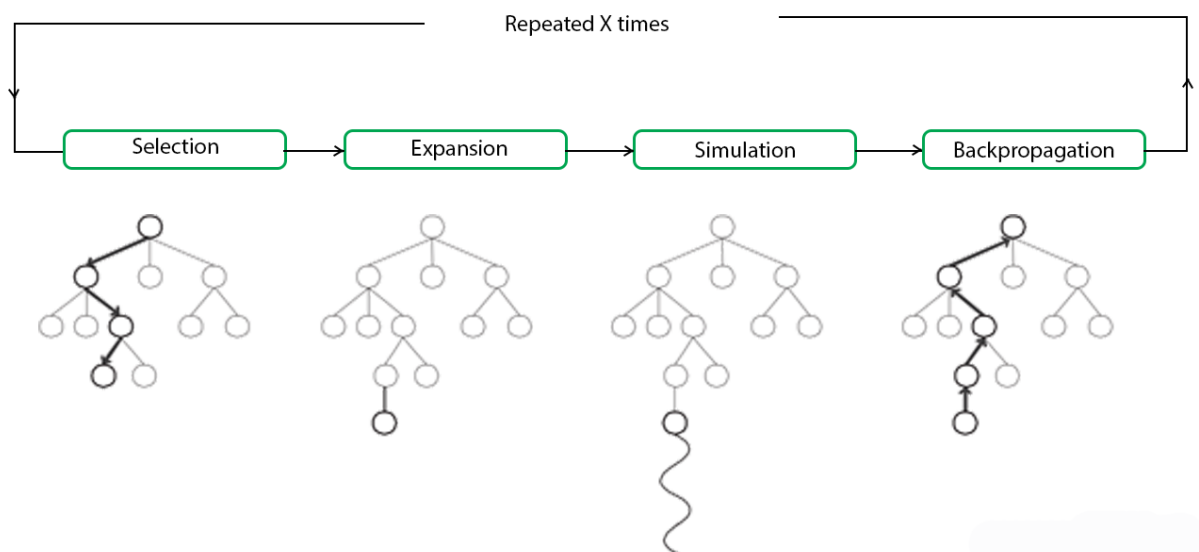


Рисунок 3.4 – Приклад роботи алгоритму пошуку дерева Монте-Карло

Одною із найголовніших переваг дерева пошуку Монте-Карло є те, що йому зовсім не потрібна функція оцінки позиції. Алгоритму потрібні лише правила гри для симуляції та визначення закінчення партії.

Основним недоліком є необхідність у великій кількості ітерацій для

визначення правильного ходу, що призводить до великої кількості витраченого часу. Ще одною особливістю цього алгоритму є те, що у деяких випадках виграш може досягатися лише одною специфічною послідовністю ходів, яка є важливою, проте може бути не поміченою MCTS через випадковість ходів в симуляції. Така лінія знаходиться «поза радаром пошуку».

### 2.3 Огляд існуючих програмних рішень

Зазвичай шахові програми аналізують тисячі результатів, перш ніж зробити ефективний хід. Оскільки апаратне забезпечення та методи програмування вдосконалюються з кожним роком, шахові движки стають все більш інтелектуальними. Сучасні програми більш вибіркові та краще розуміють позицію. Якщо побудувати повне дерево всіх можливих ходів на шаховій дошці, то вийде 10120 ходів. Це надзвичайно велика кількість.

Першим комп'ютером, якому вдалося перемогти діючого чемпіона світу з шахів, став Deep Blue. У першій зустрічі, що відбулася у Філадельфії в 1996 році, діючий на той час чемпіон Гаррі Каспаров переміг з рахунком 4-2. У матчі-реванші 1997 року, що відбувся в Нью-Йорку, перемогу здобув Deep Blue з рахунком 3.5-2.5. Це стало початком розробки ще потужніших шахових двигунів[11].

Станом на зараз, у найкращого гравця в шахи – попереднього чемпіона світу Магнуса Карлсена, шаховий рейтинг становить 2859 очок. В той же час, найкращі шахові двигуни мають рейтинг у більш ніж 3500 очок.

Існують буквально сотні "рейтингових списків", які вимірюють відносну силу шахових рушіїв на основі того, скільки ходів вони роблять за хвилину. На додаток до ранжування шахових рушіїв від найкращого до найгіршого, ці списки також надають допустимі похибки наведених рейтингів.

Розглянемо три найкращих шахових рушіїв: Stockfish, Alphazero та Leela Chess Zero.

### 2.3.1 Stockfish

Безкоштовний шаховий рушій з відкритим вихідним кодом під назвою Stockfish доступний на багатьох настільних і мобільних пристроях. Завдяки універсальному шаховому інтерфейсу його можна використовувати у шаховому програмному забезпеченні.

Станом на квітень 2023 року Stockfish є найсильнішим шаховим рушієм у світі, який регулярно посідає перше або близьке до першого місце у більшості рейтингових списків шахових рушіїв. Прогнозований рейтинг Elo перевищує 3500. 19 разів вигравав чемпіонат з комп'ютерних шахів на Chess.com та 14 разів - чемпіонат найкращих шахових рушіїв.

До команди розробників Stockfish з відкритим вихідним кодом входять Торд Ромстад, Марко Костальба, Юна Кііскі, Гері Лінскотт, Стефан Ніколе, Стефан Гешвентнер і Юст ВандеВонделе. Він походить від рушія Торда Ромстада з відкритим вихідним кодом Glaurung, який був випущений у 2004 році.

У багатопроцесорних системах Stockfish може використовувати до 1024 потоків процесора. Його таблиця транспонування може мати розмір до 32 ТБ. Stockfish використовує бітові дошки і розширений альфа-бета пошук. Його вища глибина пошуку порівняно з іншими рушіями є результатом, частково, більш агресивного обрізання та зменшення кількості пізніх ходів[12].

### 2.3.2 AlphaZero

AlphaZero складається з CNN (згорткової нейронної мережі) на основі ResNet50, яка має дві гілки і обчислює політику (дає ймовірності ходів; придумує можливі ходи) та значення (для оцінки позиції), а також пошуку дерева Монте-Карло для оцінки стану та оновлення правила вибору дій[13].

Не маючи доступу до відкритих книг або таблиць ендшпільів, AlphaZero навчався виключно за допомогою самостійної гри, використовуючи 5 000 TPU

(Tensor Processing Unit) першого покоління для створення ігор і 64 TPU другого покоління для навчання нейронних мереж. За даними DeepMind, рейтинг AlphaZero був вищим, ніж у Stockfish 8 після чотирьох годин навчання, а після дев'яти годин навчання алгоритм виграв обмежений у часі турнір на 100 партій проти Stockfish 8 (28 перемог, 0 поразок і 72 нічий). Саме тому AlphaZero можна по праву вважати найкращими шаховим рушієм сьогодення, хоча він і не має відкритого коду.

### 2.3.3 Leela Chess Zero

Leela Chess Zero – це шаховий рушій з відкритим вихідним кодом на основі нейронних мереж. Глибоке навчання з підкріпленням, яке створює нейронну мережу, що використовує метод пошуку за деревом Монте-Карло для визначення найкращих можливостей, - це метод навчання, який використовується у програмі Leela Chess Zero. Програма AlphaZero, яку вона замінює, була заснована на тому ж принципі. Перед початком навчання програмі надаються лише фундаментальні шахові правила. Починаючи з нульового рівня, LCZero грає мільйони партій проти самого себе, будуючи на їх основі нові мережі з поступово вищими рівнями гри[14].

Станом на грудень 2022 року Leela Chess Zero зіграла понад 1,5 мільярда партій проти самої себе, граючи близько 1 мільйона партій щодня, і здатна грати на рівні, який можна порівняти з Stockfish, провідною програмою для звичайних шахів.

Leela Chess Zero є безкоштовною програмою і може використовуватись будь-яким користувачем на відміну від шахового рушія AlphaZero.

### РОЗДІЛ 3. ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ

Після того як було обговорено правила гри в шахи, проаналізовано алгоритми пошуку найкращих ходів та розглянуто існуючі програмні рішення, можемо переходити до створення власного проєкту з пошуку найкращих ходів у шахах.

Для розробки я обрав мову програмування Python. Python – це друга по популярності мова програмування у світі (рисунок 3.1). Визначення популярності проводилося на основі пошукових запитів в браузері Google у 2023 році[15].

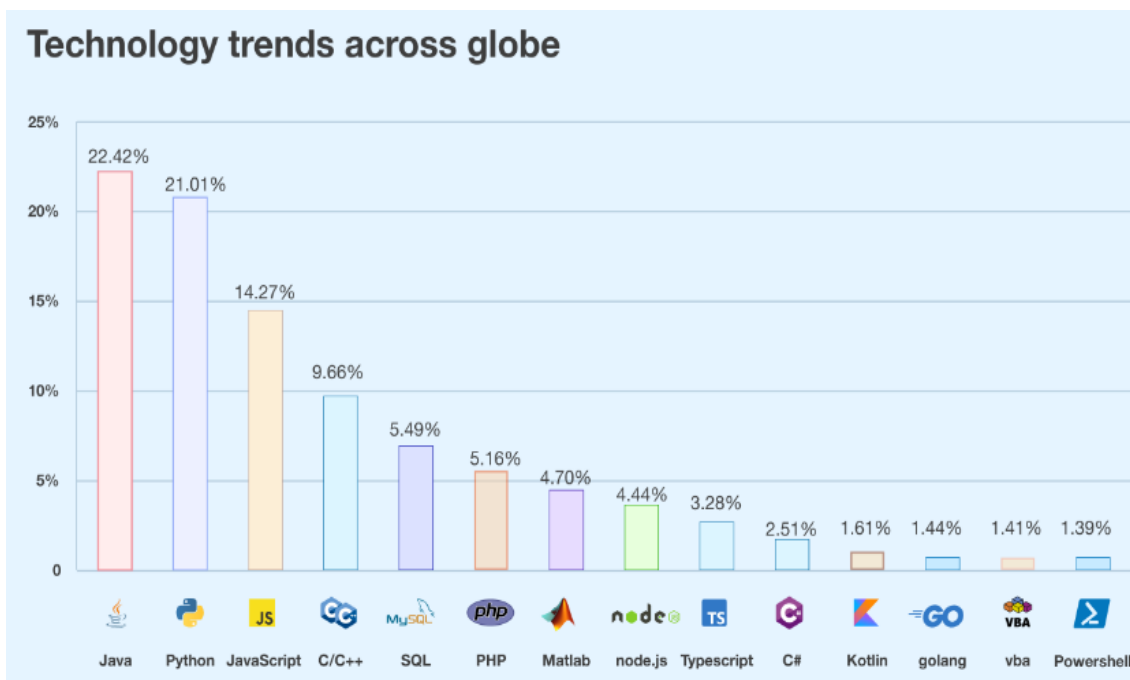


Рисунок 3.1 – Графік популярності мов програмування

Python було також обрано через велику кількість різноманітних бібліотек, що значно спрощують роботу розробникам. Також ця мова є простою у використанні, що дозволяє з легкістю створювати нові проєкти. Що ще хочеться зазначити, це те, що Python – це кросплатформна мова програмування, що дозволяє працювати проєктам, розробленим на цій мові, більш ніж на одній операційній системі, таких як Windows, macOS та Linux[16].

### 3.1 Підготовка програми середовища

Для створення свого проєкту я обрав інтегроване середовище PyCharm. За даними опитування проведеного компанією JetBrains, PyCharm – друга за популярністю Python IDE [17](рисунок 3.2).

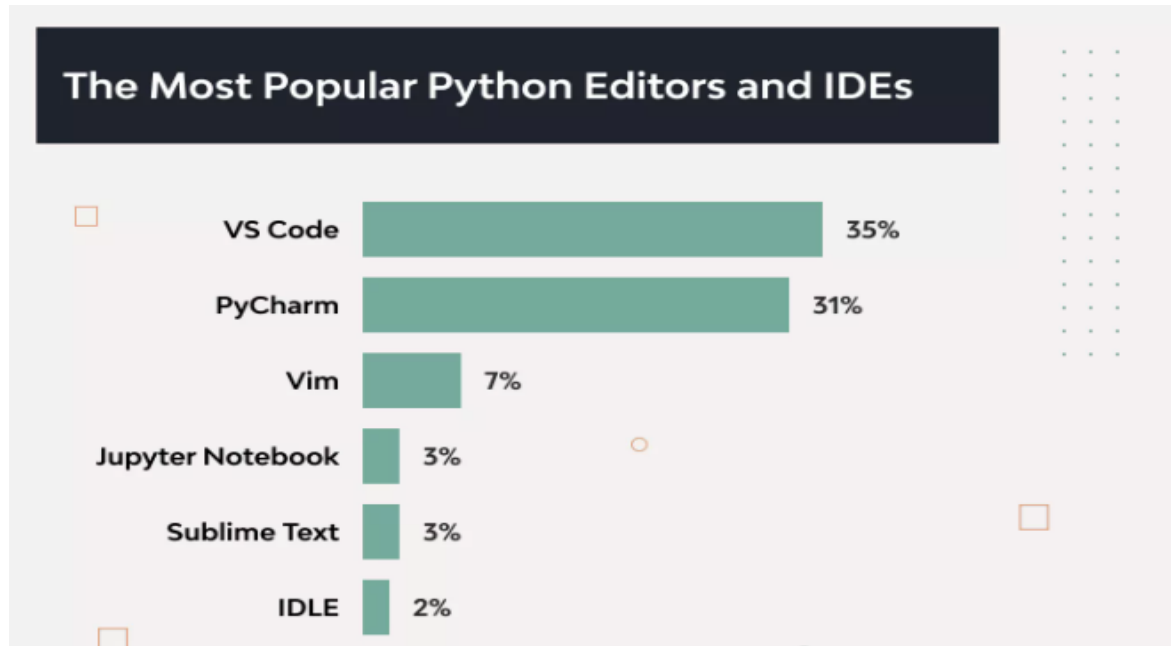


Рисунок 3.2 – Порівняння популярності Python IDE

PyCharm дуже зручний у використанні, адже в ньому наявні такі функції, як: автоматичне форматування коду (вирівнювання відступів та інше), підсвітка синтаксису та рефакторинг. Завантажити PyCharm можна на офіційному сайті[18].

Перед початком розробки потрібно встановити мову програмування. В нашому випадку це Python. Завантажити його можна на офіційному сайті.

Перевірити чи правильно працює Python можна вписавши в командному рядку «print(“Hello, World!”)» так, як показано на рисунку 3.3. Все працює так як і повинно, можемо переходити до наступних кроків.

```

Командний рядок - python
Microsoft Windows [Version 10.0.19045.2965]
(c) Корпорація Майкрософт. Усі права захищені.

C:\Users\rwand>python
Python 3.10.11 (tags/v3.10.11:7d4cc5a, Apr 5 2023, 00:38:17) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> print("Hello, World!")
Hello, World!

```

Рисунок 3.3 – Перевірка Python

Для проєкту нам потрібно ще встановити `pip` – інструмент керування пакунками на основі Python, який використовується для встановлення та супроводу програмних пакунків. При версії Python 2.7.9, 3.4 або новішої, встановлювати `pip` не потрібно – в цих версіях він йде за замовчуванням.

### 3.2 Встановлення необхідних бібліотек

Як уже зазначалось раніше, Python має велику кількість різноманітних бібліотек, що можуть полегшити роботу розробникам.

У проєкті я буду використовувати бібліотеки `python-chess` та `pygame`. Детальніше про них та як їх встановити я розповім нижче.

`Python-chess` – це зручна бібліотека для генерації та валідації ходів. В цій бібліотеці уже розроблена вся основна логіка гри в шахи, включно з просунутими шаховими правилами, про які розповідалось у розділі 2. `Python-chess` також підтримує різні формати шахової позиції, включно з FEN форматом, який ми будемо використовувати в проєкті[19].

Для встановлення цієї бібліотеки потрібно ввести команду, яку можна знайти в офіційній документації (рис. 3.4).

Requires Python 3.7+. Download and install the latest release:

```
pip install chess
```

Рисунок 3.4 – Встановлення бібліотеки `python-chess`

`Pygame` – набір модулів для написання комп'ютерних ігор на мові програмування Python. `Pygame` має комплексну документацію та через те, що вона вважається «стандартною» для написання комп'ютерних ігор – багато інструкцій з використання. Встановити цю бібліотеку також можна спеціальною командою з офіційної документації [20](рис. 3.5).

## Installation

```
pip install pygame
```

Рисунок 3.5 – Встановлення бібліотеки pygame

Існує і інший шлях під'єднання необхідних бібліотек до проекту. Зробити це також можна скориставшись менеджером бібліотек PyCharm. Для цього потрібно ввести назву бібліотеки у пошуковому рядку та натиснути кнопку «Install» біля потрібної бібліотеки (рисунок 3.6).

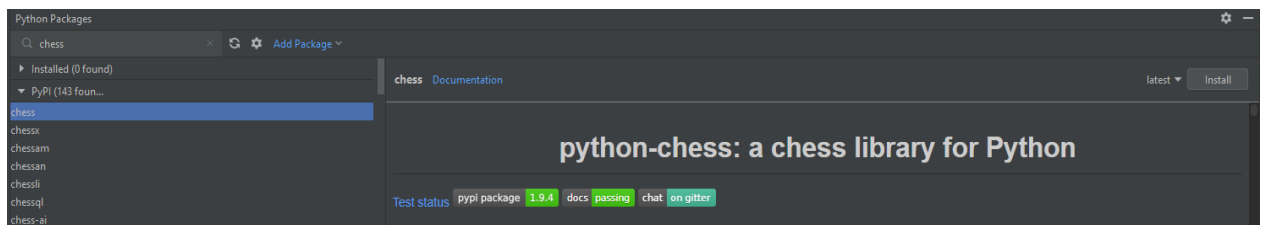


Рисунок 3.6 – Менеджер бібліотек Pycharm

В розробці також будуть використовуватись наступні модулі Python:

- Random
- Math
- Time

Модуль `random` буде використовуватись для симуляції гри при реалізації алгоритму дерева пошуку Монте-Карло.

Модуль `math` для обчислення значення Upper Confidence Bound.

Модуль `time` буде використовуватись для визначення часу на хід зроблений шаховим двигуном.

Всі вище наведені модулі не потребують інсталяції, оскільки вони вже вбудовані в Python і можуть бути використані безпосередньо шляхом імпорту необхідних функцій або цілого модуля в робочий файл.

### 3.3 Реалізація алгоритмів пошуку найкращого ходу

Для реалізації я обрав три основних алгоритми пошуку найкращого ходу в настільних комп'ютерних іграх: Minimax, Minimax з Альфа-Бета обрізкою та дерево пошуку Монте-Карло. Алгоритми були детально розглянуті у розділі 2.

#### 3.3.1 Minimax

Для реалізації алгоритму Minimax потрібно спочатку ввести функцію обрахунку позиції. У нас це *eval(board)*. На вхід ця функція приймає ігорвий стан та додає оцінку кожної фігури на дошці. Для білих фігур – значення додатні, для чорних – від'ємні. Оцінка фігури отримується за допомогою функції *get\_piece\_val(piece)* (рисунок 3.7)

```
def get_piece_val(piece):
    if (piece == None):
        return 0
    value = 0
    if piece == "P" or piece == "p":
        value = 10
    if piece == "N" or piece == "n":
        value = 30
    if piece == "B" or piece == "b":
        value = 30
    if piece == "R" or piece == "r":
        value = 50
    if piece == "Q" or piece == "q":
        value = 90
    if piece == "K" or piece == "k":
        value = 900
    return value
```

Рисунок 3.7

Тепер переходимо до реалізації самого алгоритму (рис. 3.8). Алгоритм Minimax працює на основі принципу альтернування між максимальним і мінімальним гравцями. Якщо поточний гравець є Максимайзером, то для кожного законного ходу він рекурсивно викликає функцію *minimax* для наступного стану гри після зробленого ходу. Значення *is\_max\_player* змінюється на *False*, щоб показати, що наступним кроком буде хід мінімального гравця. З результатів рекурсивного виклику обирається найвище значення (максимум) із отриманих

оцінок і повертається як оцінка для поточного стану гри. Якщо поточний гравець є Мінімайзером, алгоритм працює аналогічно, але замість максимуму вибирається мінімум.

```

if current_depth == 0:
    leaf_node_score = eval(board)
    return (leaf_node_score, nodes_per_depth)

if is_max_player:
    best_score = -100000
    for legal_move in board.legal_moves:
        move = chess.Move.from_uci(str(legal_move))
        board.push(move)
        node_score, nodes_per_depth = minimax(board, max_depth, current_depth - 1, False, nodes_per_depth)
        best_score = max(best_score, node_score)
        board.pop()
    return best_score, nodes_per_depth
else:
    best_score = 100000
    for legal_move in board.legal_moves:
        move = chess.Move.from_uci(str(legal_move))
        board.push(move)
        node_score, nodes_per_depth = minimax(board, max_depth, current_depth - 1, True, nodes_per_depth)
        best_score = min(best_score, node_score)
        board.pop()
    return best_score, nodes_per_depth

```

Рисунок 3.8 Реалізація алгоритму

Функція *minimax* отримує на вхід поточний стан *board* гри, глибину пошуку *max\_depth*, поточну глибину *current\_depth*, прапорець *is\_max\_player*, який вказує, чи зараз розглядається хід максимального гравця, і словник *nodes\_per\_depth*, який використовується для аналізу алгоритму. Функція повертає кортеж, що складається з оцінки найкращого ходу для поточного стану і оновленого словника *nodes\_per\_depth*.

### 3.3.2 Альфа-Бета обрізка

Реалізація алгоритму Альфа-Бета обрізки є майже ідентичною з алгоритмом Мінімах. Основна відмінність в тому, алгоритм Альфа-Бета обрізки має додаткові параметри *alpha* та *beta*, які представляють верхню і нижню межі оцінок для Максимайзера і Мінімайзера відповідно.

Під час рекурсивного виклику функції *alpha\_beta*, якщо поточний гравець є Максимайзером, алгоритм оновлює значення *alpha* з кожним знайденим кращим (більшим) значенням оцінки. При цьому, якщо значення *beta* стає меншим або рівним *alpha*, то можна виконати відсічення і повернути поточне найкраще значення.

Якщо поточний гравець є Мінімайзером, алгоритм оновлює значення *beta* з кожним знайденим кращим (меншим) значенням оцінки. При цьому, якщо значення *beta* стає меншим або рівним *alpha*, відсікається гілка і повертається поточне найкраще значення.

### 3.3.3 Дерево пошуку Монте-Карло

Для початку необхідно розробити структуру даних для представлення дерева.

```
class MCTSNode(object):
    def __init__(self, board, node_parent=None, last_move=None):
        self.board = board
        self.node_parent = node_parent
        self.last_move = last_move
        self.black_win_counts = 0
        self.white_win_counts = 0
        self.rollouts_num = 0
        self.children = []
        all_moves = [board.san(i) for i in list(board.legal_moves)]
        self.unvisited_moves = all_moves
```

В цьому `board` коді описує теперішній ігровий стан, `node_parent` – батьківський вузол, `last_move` – хід, що привів до даної позиції, `black_win_counts` та `white_win_counts` для статистики симуляцій, `rollouts_num` зберігає число симуляцій та `unvisited_moves` для списку всіх не відвіданих ходів з цієї позиції.

Тепер, коли ми описали структуру, можемо переходити до самого алгоритму дерева пошуку Монте-Карло. Для нас це клас `MCTS`:

```
class MCTS:
    def __init__(self, rounds_num, const):
        self.num_rounds = rounds_num
        self.const = const

    def select_move(self, board):
        root_node = MCTSNode(board)
```

```

for i in range(self.num_rounds):
    node = root_node
    while (not node.can_add_child()) and (not node.terminal()):
        node = self.choose_child(node)
    if node.can_add_child():
        node = node.add_random_child()
    winner = self.random_game_simulation(node.board)
    while node is not None:
        node.note_win(winner)
        node = node.node_parent
player = ""
if board.turn:
    player = "White"
else:
    player = "Black"
best_move = None
best_percent = -1.0
for child in root_node.children:
    if child.board.turn:
        player = "White"
    else:
        player = "Black"
    child_percent = child.winning_frac(player)
    if child_percent > best_percent:
        best_percent = child_percent
        best_move = child.last_move
return board.parse_san(best_move).uci()

```

Для алгоритму можна задати кількість ітерацій (*rounds\_num*) та константу для обчислення UCSB.

Ми починаємо зі створення нового дерева. Кореневий вузол - це поточний ігровий стан. Потім ми генеруємо кілька симуляцій. У цій реалізації обробляємо кожен хід фіксовану кількість разів за допомогою циклу. В інших реалізаціях замість цього можна обмежити час, виділений на обробку.

Кожен раунд починається з обходу дерева аж до знаходження вузла, до якого можна додати дочірній елемент (це будь-яка позиція на дошці, що передбачає допустимий хід, якого ще немає в дереві). Функція *choose\_child* відповідає за вибір найкращої гілки для дослідження. У нас це відбувається за порівнянням UCSB значення. Детальніше про те, як це відбувається, розповідалося в другому розділі.

Після знаходження відповідного вузла ми викликаємо *add\_random\_child* для вибору і включення в дерево будь-якого з наступних ходів. На цьому етапі *node* являє собою новостворений *MCTSNode* з нульовою кількістю симуляцій.

Тепер ми здійснюємо симуляцію з цього вузла шляхом виклику *random\_game\_simulation*. Реалізація *random\_game\_simulation* полягає в простому

розіграші партії з випадковими ходами і визначенням переможця.

Нарешті, ми оновлюємо значення лічильника вигравів новоствореного вузла і всіх його предків.

Після завершення заданої кількості раундів (ітерацій) ми маємо вибрати хід. Для цього ми просто перебираємо всі вузли верхнього рівня та вибираємо той, який має максимальний відсоток вигравів.

### 3.3.4 Приклад використання програми

Для того, щоб використовувати програму, потрібно для початку в коді ввести стартову позицію у FEN форматі. Про нього розповідалося у розділі 2.

```
fen = "r6k/pp4pp/1b1P4/8/1n4Q1/2N1RP2/PPq3p1/1RB1K3 b - - 0 1"
board = chess.Board(fen)
```

Рисунок 3.9 – Ввід шахової позиції

Далі нам потрібно в коді змінити яким з алгоритмів ми хочемо користуватися (рисунок 3.10)

```
# 1 for minimax, 2 for alpha-beta pruning, 3 for mcts
choose_engine = 2
```

Рисунок 3.10 – Вибір алгоритма

За потреби можна також змінити глибину для алгоритмів Minimax та Альфа-Бета відсікання та кількість ітерацій та константу для дерева пошуку Монте-Карло.

Для демонстрації роботи програми оберемо алгоритм Minimax з глибиною пошуку 3 (рисунок 3.11). Початкова шахова позиція у FEN форматі – *r6k/pp4pp/1b1P4/8/1n4Q1/2N1RP2/PPq3p1/1RB1K3 b - - 0 1*. Це позиція з матом у два ходи, який ставлять чорні. Шаховий двигун грає за чорних. За білих грає користувач. Щоб зробити хід, користувачу потрібно натиснути на фігуру та перетягнути її на потрібне поле. Зліва у нас знаходиться сама шахова дошка,

справа – початкова позиція, зроблені ходи (зліва – білих, справа – чорних) та час витрачений на хід шаховим двигуном.



Рисунок 3.10 – Приклад роботи програми

### 3.4 Порівняння алгоритмів

Порівняння алгоритмів проводилося за часом витрачений на розв'язання шахової головоломки (Таблиця 1). Нижче наведені стартові позиції записані у FEN форматі.

Головоломка 1: *r6k/pp4pp/1b1P4/8/1n4Q1/2N1RP2/PPq3p1/1RB1K3 b - - 0 1.*

Головоломка 2: *7r/p3ppk1/3p4/2p1P1Kp/2Pb4/3P1QPq/PP5P/R6R b - - 0 1.*

Головоломка 3: *rn2kb1r/ppp1pppp/8/8/4q3/3P1N1b/PPP1BPnP/RNBQ1K1R b kq - 0 1.*

Номер ГОЛОВОЛОМКИ	Тривалість розв'язку, с		
	Minimax	Альфа-Бета обрізка	MCTS
1	8.3	0.38	1042.81

2	132.27	8.03	987.34
3	220.74	25.41	1035.4

Таблиця 1 – Порівняння алгоритмів

Отже, найкращі результати показав алгоритм Альфа-Бета обрізки, чого і слід було очікувати, адже порівняно з алгоритмом Minimax, він не обробляє всі ходи. В нашому випадку алгоритм Альфа-Бета обрізки мав хороший результат через доволі раннє знаходження мату, в результаті чого і були обрізані цілі піддерева.

Найгірші результати показав алгоритм дерева пошуку Монте-Карло. Це також не було несподіванкою, адже для вибору правильного ходу йому потрібно приблизно 10000 ітерацій, що займає приблизно 8 хвилин. При оптимізації коду, цей результат можна покращити. Хоча, при виборі меншого значення константи вдалося дещо зменшити кількість ітерацій для пошуку правильного ходу. Це відбулося завдяки зосередженню алгоритма на точній оцінці підібраного ходу. Варто також зазначити, що ймовірність правильності ходу при меншій кількості ітерацій та значення константи падає, тому такий результат не є стабільним.

## ВИСНОВКИ

Під час виконання цієї кваліфікаційної роботи було проаналізовано основні відомості про штучний інтелект та сфери, в яких він використовується, зокрема й настільні комп'ютерні ігри. Також був проведений аналіз алгоритмів пошуку найкращого ходу в настільних комп'ютерних іграх та огляд існуючих програмних рішень.

Для перевірки алгоритмів знаходження правильного ходу було розроблено програмний додаток на мові програмування Python, за допомогою інтегрованого середовища PyCharm. Додаток простий у використанні та дозволяє користувачу грати в настільну комп'ютерну гру шахи проти рушія зі штучним інтелектом на основі таких алгоритмів як Minimax, Альфа-Бета обрізка та дерево пошуку Монте-Карло. За допомогою програми вдалося порівняти ці алгоритми і було визначено, що найшвидшим алгоритмом для пошуку правильного ходу є Альфа-Бета обрізка. Найгіршим, у свою чергу – дерево пошуку Монте-Карло.

Розроблений додаток може використовуватися як для тренування гри в шахи, так і для створення та порівняння нових алгоритмів пошуку найкращого ходу у настільних комп'ютерних іграх.

Доцільним є продовження розробки додатку для впровадження та аналізу нових алгоритмів, а також оптимізації уже розроблених.

## ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Штучний інтелект: історія виникнення та перспективи розвитку [Електронний ресурс]. – Режим доступу до ресурсу: <https://futurum.today/shtuchnyi-intelekt-istoriia-vynyknennia-ta-perspektyvy-rozvytku/>
2. Deep Medicine: How Artificial Intelligence Can Make Healthcare Human Again, 2019. – 400 с. – (Basic Books).
3. What is computer vision? [Електронний ресурс]. – 2023. – Режим доступу до ресурсу: <https://www.ibm.com/topics/computer-vision>.
4. Robotics and artificial intelligence: The role of AI in robots [Електронний ресурс]. – 2022. – Режим доступу до ресурсу: <https://aibusiness.com/verticals/robotics-and-artificial-intelligence-the-role-of-ai-in-robots>.
5. The Evolution of Chess AI [Електронний ресурс]. – 2022. – Режим доступу до ресурсу: <https://builtin.com/artificial-intelligence/chess-ai>.
6. Introducing ChatGPT [Електронний ресурс]. – 2022. – Режим доступу до ресурсу: <https://openai.com/blog/chatgpt>.
7. Правила гри в шахи ФІДЕ [Електронний ресурс] – Режим доступу до ресурсу: [https://ukrchess.org.ua/arbiter/fide/pravila\\_gry\\_fide\\_ukr.pdf](https://ukrchess.org.ua/arbiter/fide/pravila_gry_fide_ukr.pdf).
8. Minimax Algorithm in Chess, Checkers & Tic-Tac-Toe [Електронний ресурс]. – 2022. – Режим доступу до ресурсу: <https://www.idtech.com/blog/minimax-algorithm-in-chess-checkers-tic-tac-toe>
9. Alpha-Beta [Електронний ресурс]. – 2019. – Режим доступу до ресурсу: <https://www.chessprogramming.org/Alpha-Beta>.
10. Фергюсон К. Deep Learning and the Game of Go / К. Фергюсон, М. Памперла., 2019. – 384 с. – (Manning).
11. 25 years ago today: How Deep Blue vs. Kasparov changed AI forever

- [Электронный ресурс]. – 2022. – Режим доступа до ресурсу:  
<https://aibusiness.com/ml/25-years-ago-today-how-deep-blue-vs-kasparov-changed-ai-forever>.
12. Stockfish Chess Engine: The Ultimate Guide [Электронный ресурс] – Режим доступа до ресурсу: <https://www.chessjournal.com/stockfish/>.
  13. Седлер М. Game Changer: AlphaZero's Groundbreaking Chess Strategies and the Promise of AI / М. Седлер, Н. Реган., 2019. – 416 с. – (New In Chess).
  14. Leela Chess Zero [Электронный ресурс] – Режим доступа до ресурсу: [https://www.chessprogramming.org/Leela\\_Chess\\_Zero](https://www.chessprogramming.org/Leela_Chess_Zero).
  15. Top Programming Languages of 2023 [Электронный ресурс] – Режим доступа до ресурсу:  
<https://www.simform.com/blog/top-programming-languages/>.
  16. Python Developers Survey 2021 Results [Электронный ресурс] – Режим доступа до ресурсу:  
<https://lp.jetbrains.com/python-developers-survey-2021/>.
  17. Pycharm [Электронный ресурс] – Режим доступа до ресурсу:  
<https://www.jetbrains.com/pycharm/>.
  18. Python [Электронный ресурс] – Режим доступа до ресурсу:  
<https://www.python.org/about/>.
  19. python-chess: a chess library for Python [Электронный ресурс] – Режим доступа до ресурсу: <https://python-chess.readthedocs.io/en/latest/>.
  20. Pygame [Электронный ресурс] – Режим доступа до ресурсу:  
<https://www.pygame.org/wiki/about>.