МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ПРИВАТНЕ АКЦІОНЕРНЕ ТОВАРИСТВО «ПРИВАТНИЙ ВИЩИЙ НАВЧАЛЬНИЙ ЗАКЛАД «ЗАПОРІЗЬКИЙ ІНСТИТУТ ЕКОНОМІКИ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ»

Кафедра інформаційних технологій

ДО ЗАХИСТУ ДОПУЩЕНА

Зав. кафедрою _____
(підпис)
д.е.н., доцент Левицький С.І.

КВАЛІФІКАЦІЙНА МАГІСТЕРСЬКА РОБОТА

ТЕМА РОЗРОБКА АЛГОРИТМУ ТА ПРОГРАМНОЇ РЕАЛІЗАЦІЇ ВИБОРУ СТРАТЕГІЇ ІГОР НА ШАХІВНИЦІ

Виконав
ст. гр. _____ В.В.Росін

Науковий керівник _____ Д.Г. Медведєв
к.т.н. (підпис)

Запоріжжя

2024

ПРАТ «ПВНЗ «ЗАПОРІЗЬКИЙ ІНСТИТУТ ЕКОНОМІКИ
ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ»

Кафедра інформаційних технологій

ЗАТВЕРДЖУЮ

Зав. кафедри _____
(підпис)
д.е.н., доцент Левицький С.І.

__.__.____ р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ МАГІСТЕРСЬКУ РОБОТУ

Студенту гр. КІ-112м , спеціальності *«Комп'ютерна інженерія*_____

_____

1.Тема: Розробка алгоритму та програмної реалізаці вибору стратегії ігор на шахівниці
затверджена наказом по інституту № 02-25 від 05.12.2022 р.

2. Термін здачі студентом закінченої роботи: 13.01.2024 р.

3. Перелік питань, що підлягають розробці:

- постановка завдання;
- інформаційний огляд;
- опис основних технологій реалізації;
- планування функціоналу додатку;
- розробка бази даних;
- створення алгоритмів виконання завдання;
- розробка серверної архітектури;
- створення клієнтського веб-додатку;
- оформити звіт за результатами роботи.

ЗАТВЕРДЖУЮ
Зав.кафедрою _____

К А Л Е Н Д А Р Н И Й Г Р А Ф І К

підготовки магістерської дипломної роботи

студентом інституту ЗІЕІТ *денної* **форми навчання**

гр. _ІПЗ-112м_ **Росіну Володимиру Володимировичу**

на 2023-2024 навчальний рік

№ етапу	Зміст	Терміни виконання	Готовність по графіку %, підпис керівника	Підпис керівника про повну готовність етапу, дата
1.	Корегування теми магістерської дипломної роботи, збір практичного матеріалу за темою магістерської дипломної роботи	**04.09.23-17.10.23**		
2.	**І атестація** І розділ магістерської дипломної роботи	**23.10.23-28.10.23**		
3.	**ІІ атестація** ІІ розділ магістерської дипломної роботи	**20.11.23-25.11.23**		
4.	**ІІІ атестація** ІІІ розділ магістерської дипломної роботи, висновки та рекомендації, додатки, реферат, перевірка програмою «Антиплагіат»	**18.12.23-23.12.23**		
5.	**Доопрацювання магістерської дипломної роботи, підготовка презентації, отримання відгуку керівника і рецензії**	**25.12.23-06.01.24**		
6.	Попередній захист магістерської дипломної роботи	**08.01.24-13.01.24**		
7.	Подача магістерської дипломної роботи на кафедру	**за 3 дні до захисту**		
8.	**Захист магістерської дипломної роботи**	**15.01.24-20.01.24**		

Дата видачі завдання: 04 .09.2023 р.

Керівник кваліфікаційної
магістерської роботи _____ Медведєв Д.Г.
(підпис) (прізвище та ініціали)

Завдання отримав до виконання _____ Росін В.В.
(підпис студента) (прізвище та ініціали)

РЕФЕРАТ

Кваліфікаційна робота містить 71 стор., 18 рис., 16 використаних джерел.

Об’єкт: машинне навчання в логічних іграх.

Предмет: вибір найкращого ходу в логічних іграх.

Мета: розробка еволюційної програми гри в «Реверсі».

Задачі: проаналізувати існуючі методи та алгоритми для розробки еволюційної програми гри в «Реверсі»; розробити алгоритм з можливістю самонавчання.

В роботі виконано аналіз існуючих методів машинного навчання, розроблено штучну нейронну мережу для здійснення рішень під час обирання можливих ходів та аналізу позиції в грі «Реверсі». Створено веб-додаток, який реалізовує розроблені алгоритми.

ЛОГІЧНІ ІГРИ, ШТУЧНА НЕЙРОНА МЕРЕЖА, РЕВЕРСІ, HTML5, JAVASCRIPT, JSON, CANVAS, NODEJS.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

ІЛП	Індуктивне логічне програмування
МПВ	Марковський процес вирішування
ОС	Операційна система
ПЗ	Програмне забезпечення
ШНМ	Штучна нейронна мережа
CSS	Каскадні таблиці стилів
HTML	Мова розмітки гіпертекстових документів

ВСТУП

Сьогодні комп'ютерні технології стрімко розвиваються, і це відображається на різноманітності комп'ютерних ігор. Однак, хоча комп'ютерні ігри часто вимагають від гравця неабияких навичок та вмінь, самі вони є результатом складної роботи розробників програмного забезпечення і демонструють їхні навички та вміння.

У більшості випадків такі ігри є ресурсоємними, але не демонструють власного інтелекту.

Існує також значна кількість логічних ігор, таких як шахи, шашки, го, санмеджі та інверсія. Програмування таких ігор вимагає від програми прояву власного інтелекту. Програма повинна приймати власні рішення і обирати продовження гри.

Це є частиною такої галузі комп’ютерних наук як штучний інтелект.

Прийняття рішень є невід’ємною частиною логічних ігор. Але можливість робити висновки на помилках й навчатись на них, є явною ознакою наявності інтелекту.

Більшість програм з самонавчанням базуються на штучних нейронних мережах. Це надає змогу програмі приймати не лише вірні рішення, але й зменшити кількість хибних.

Тому розробка алгоритму самонавчання в логічних іграх є актуальною.

Мета: розробка еволюційної програми гри в «Реверсі».

Об’єкт: машинне навчання в логічних іграх.

Предмет: вибір найкращого ходу в логічних іграх.

Задачі дослідження:

1. Проаналізувати існуючі методи та алгоритми для самонавчання в логічних іграх.

2. Розробити алгоритм самонавчання для гри «Реверсі».

3. Вдосконалити існуючий програмний продукт, який реалізовуватиме перебірні стратегії з новим способом оцінки позиції, алгоритмом самонавчання.

РОЗДІЛ 1 ОСОБЛИВОСТІ ПРОГРАМУВАННЯ ЛОГІЧНИХ ІГОР

1.1. Вибір логічних ігор. Правила ігор.

Перше питання яке виникає при розробці це вибір платформи на якій буде працювати програма. Тому в силу своєї універсальності було вирішено обрати WEB-інтерфейс.

Існує велика кількість ігор цього типу, але на даному етапі написання кваліфікаційної роботи, я зупинився на створенні комп’ютерного супротивника для Реверсі.

Реверсі представляє собою настільну гру на спеціальній дошці 8 на 8 клітин та 64 фішки чорних або білих кольорів.

Гра була винайдена в Великобританії в 1880 році і користувалася великою популярністю, але згодом була забута. Відродили її в Японії, де вона в 1971 році під назвою отелло знову стала популярна. З 1977 року регулярно проводяться чемпіонати світу з гри в реверсі.

Реверс - це стратегічна гра, схожа на шашки або шахи. Як і в шахах, гру прийнято ділити на три частини: дебют (рання гра), середня гра (середня партія) і закінчення (кінцева партія). Однак, на відміну від шахів, кількість можливих дебютів набагато менша, і всі вони легше запам'ятовуються. Всі серйозні гравці знають дебют на п'ять-шість ходів наперед, щоб уникнути явно програшних ходів на цьому етапі. Середня партія, мабуть, найбільш "вільна" і в той же час найскладніша частина гри. Тим не менш, багато партій, які, здавалося б, програні в мітельшпілі, набувають нових рис у завершальній фазі гри - ендшпілі. Золоте правило ендшпілю - правильно використовувати час. Для заданої тактики прийнято рахувати фішки, які впливають на кінцевий результат гри. Звичайно, кількість результатів залежить від того, з якого ходу ви почнете рахувати. Ось чому комп'ютери грають набагато краще, ніж люди. Комп'ютер може прорахувати всі можливі

варіанти (яких за комп'ютерними мірками дуже мало) і завжди обирає той, який мінімізує результат людини і максимізує шанси комп'ютера.

Правила гри «Реверсі»

На початку партії чорні розміщують чотири фігури у центрі дошки: d5 і e4 за чорних і d4 і e5 за білих. Чорні роблять перший хід. Потім вони ходять по черзі.

Роблячи хід, гравець повинен поставити свою фігуру на одну з клітинок дошки так, щоб фігури суперника були розміщені в горизонтальному, вертикальному або діагональному ряді між цією фігурою та фігурами його кольору, які вже є на дошці. Всі фігури суперника в цьому "закритому" ряду змінюють колір і належать гравцеві, який зробив цей хід.

Якщо в результаті одного ходу одночасно "закриваються" дві або більше фішок суперника, то всі фішки в усіх "закритих" рядах міняють свій колір на протилежний.

Гравець має право вибрати одну з можливих комбінацій. Якщо у гравця є допустима рука, вона не може бути відкинута. Якщо у гравця немає жодної можливої комбінації, роздача переходить до суперника.

Гра закінчується, коли всі фішки на дошці втрачені, або коли жоден гравець не може зробити хід, або коли один із суперників втрачає фішки. В кінці гри підраховуються фішки кожного кольору, і перемагає гравець, чиї фішки опинилися на дошці. Якщо у гравців однакова кількість фішок, гра закінчується внічию.

Стратегія «багато і швидко»

У класичній грі "Реверсивний Отелло" гравець намагається (при кожній нагоді) виграти якомога більше фішок свого опонента. У літературі, присвяченій реверсивному Отелло, ця стратегія називається "максимальною грою". На середніх стадіях гри така стратегія призводить до невигідної ситуації. Ефективні ходи будуть втрачені, і гравець буде змушений робити слабкі ходи, що призведе до втрати ігрових фішок. В результаті, якщо

гравець-початківець намагається атакувати "багато і швидко" на початку гри, він завжди буде програвати в кінці.

Стабільні диски

Фішка (диск), яку не можна ніяк повертати в наступних партіях, називається фіксованою. Наприклад, камінь у кутку дошки.

У звичайній реверсивній грі фіксований диск - це велика перевага. І чим раніше гравець отримає фіксований диск, тим більше користі від нього (хороший реверсер навряд чи упустить перемогу, якщо він виграв один з кутів в першій половині гри).

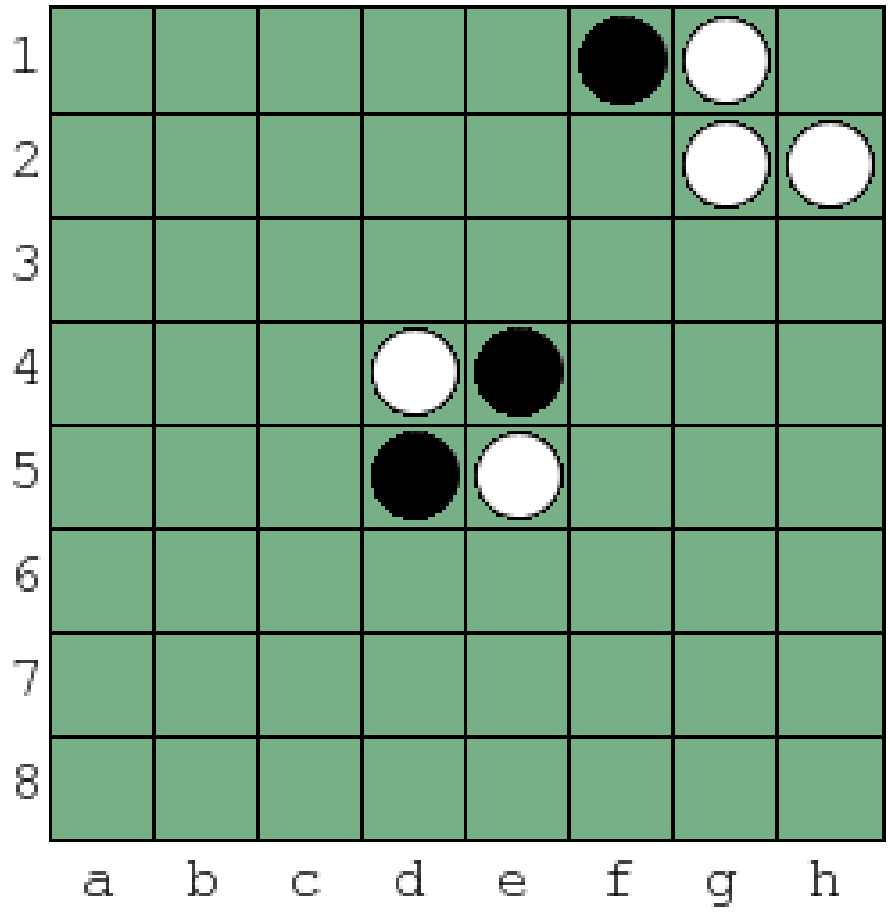

Рис. 1.1 Стабільні диски

Чорні, рано чи пізно, будуть змушені зайняти кут h1 і прилеглі до нього клітини f1, g1, g2, h2 і h3, створивши на цих полях стабільні диски (Рис. 1.1).

Подібні блоки можуть створюватися не тільки навколо кутів, але і навколо полів на краю дошки (Рис. 1.2).

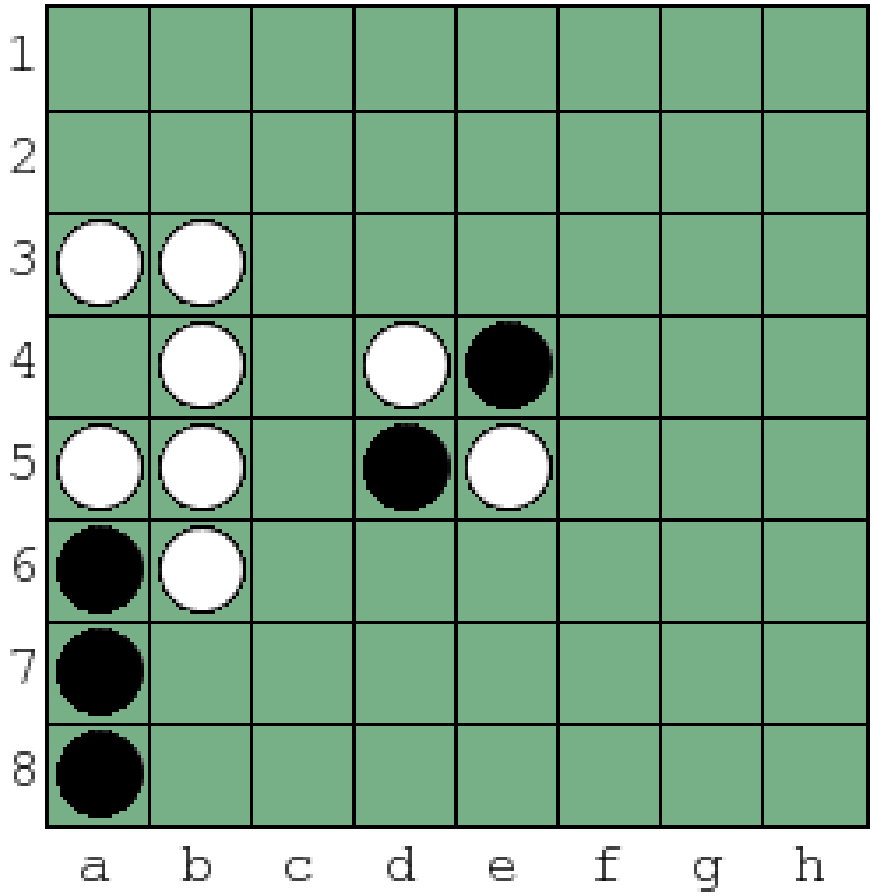

Рис. 1.2 Блок навколо полів

Але ефективність подібної тактики на краях може бути дуже низька і навіть збиткова (противник зможе використовувати хід в заблоковане поле для передачі вам ходу в невигідній позиції).

Цінності полів в реверсі

Звичайно ж, багато що залежить від конкретної позиції на дошці, але в цілому можна сказати наступне: для реверсі явно сильними полями є кути a1, a8, h1, h8, а явно слабкими – X-поля b2, b7, g2, g7 (Рис. 1.3).

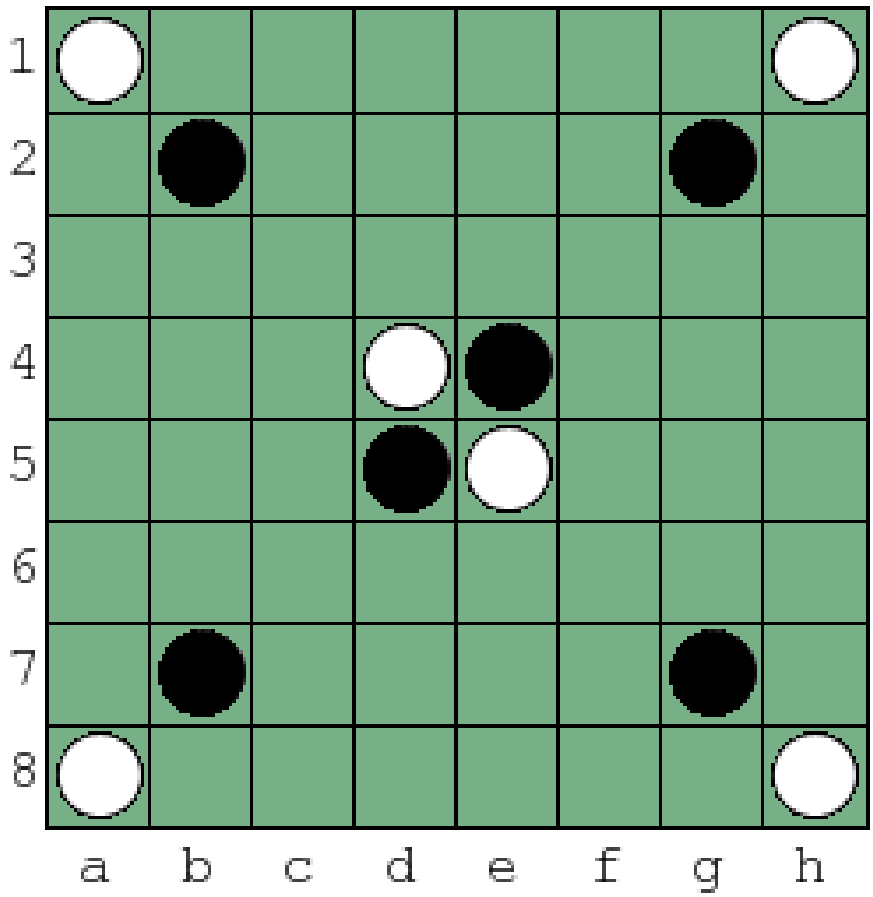

Рис. 1.3 Сильні та слабкі кути

Мобільність

Основний ключ до хорошої гри в реверс полягає в розумінні та вмінні створювати рухомі позиції з діапазоном можливих хороших або нейтральних ходів.

Що стосується мобільності, то стратегію гри можна розділити на два елементи підтримання власної мобільності та мінімізація мобільності суперника.

Межі та огорожі

Якщо ваш суперник створив широку демаркаційну лінію з фішок, намагайтеся якомога більше не порушувати її. Паркани з цих фішок значно обмежують рухливість суперника і збільшують вашу власну. Приклад таких зборів можна побачити на рисунку 1.4.

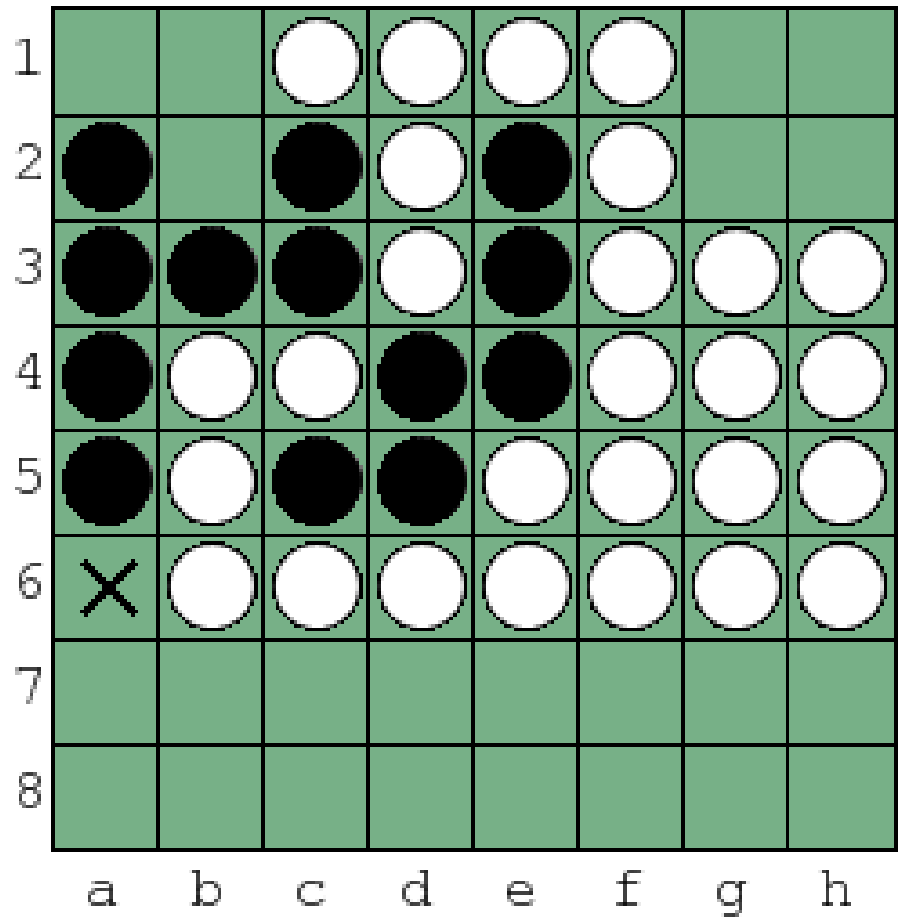

Рис. 1.4 Границі та забори

Внизу і праворуч білі створили великі кордони і, таким чином, позбавлені ходів на цих ділянках. У той же час чорні також мають багато ходів. Правильна стратегія чорних - не проривати кордони білих, а грати вліво (а6).

Пастка для курців (сходи для курців).

Пастка стоунера в протилежній руці полягає в атаці на неврівноважену сторону суперника з метою зміни кутів. При цьому, якщо пастка виконана правильно, суперник не може уникнути такого переходу. Стоунер-пастки виконуються в два етапи. Спочатку гравець отримує контроль над діагоналлю, граючи в області X, а потім атакує слабкі сторони суперника (в тому числі порожні області C) з загрозою взяття кута. Суперник не може контратакувати фігурою на полі X, яку він вже зіграв, тому він не може контратакувати фігурою на своєму полі.

Білі грають на b7 і втрачають контроль над діагоналлю b7-e4. Чорні не можуть одразу взяти поле a8, тому білі атакують слабке місце чорних на полі d8 і отримують щонайменше шість фіксованих фігур, взявши поле h8. Зауважте, що чорні не мають жодних засобів протистояти плану білих. (Рис. 1.5).

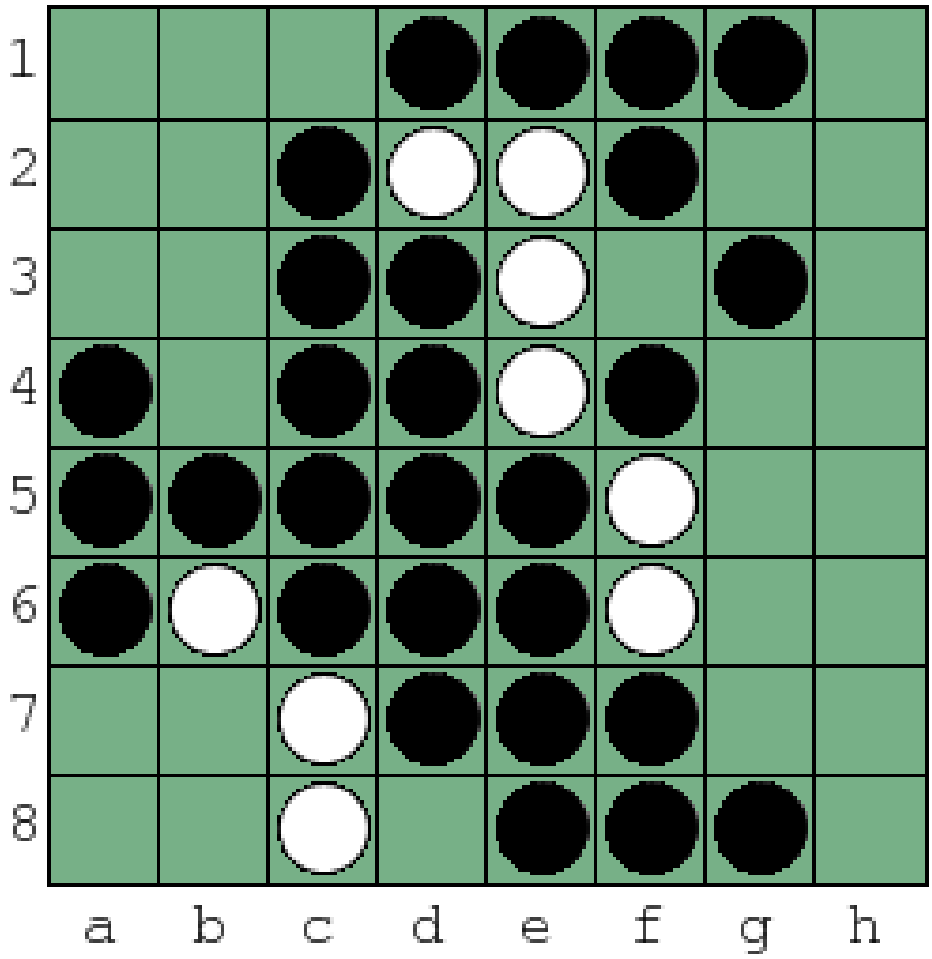

Рис. 1.5 Пастки Стонера

Послідовність ходу, паритет

Одним з найважливіших факторів, що впливають на результат партії, є порядок ходів. У більшості випадків (якщо в партії немає прохідних ходів) білі закінчують партію.

У класичних реверсивних іграх останній хід має невелику перевагу. Іншими словами, білі мають невелику перевагу, тому що у суперника немає

шансів відреагувати на останній хід і всі шашки на дошці врівноважуються цим ходом.

1.2 Аналіз існуючих алгоритмів логічних ігор

Метод "Грубої сили".

Самий простий алгоритм це метод повного перебору. Стратегія пошуку найкращого ходу здійснюється за допомогою рекурсії. Рекурсія – це коли функція визиває сама себе, й кожен раз при вході в функцію у стек програми записується копія змінних. Але не відноситься до статичних змінних, спільних для усіх викликів функції. Рекурсія дозволяє описати деякі процеси, лише визначивши закони їх розвитку, а як в цілому буде розвиватись алгоритм, не так важливо. Деякі завдання, які легко вирішуються за допомогою рекурсії, часто неможливо або складно вирішити іншим способом. Теж саме відноситься і до пошуку здійснення найкращого ходу. Для цього потрібно отримати всі варіанти ходів (переміщення), зробити оцінку для кожного з них та знайти переміщення з максимальною оцінкою.

Мінімакс

Мінімакс – правило прийняття рішень, що використовується в теорії ігор для мінімізації можливих втрат з тих, які особа, яка приймає рішення не може уникнути при розвитку подій за найгіршим для неї сценарієм.

У комбінаторній теорії ігор існує алгоритм мінімаксу для прийняття ігрових рішень. Проста версія мінімаксного алгоритму, описана нижче, застосовується до таких ігор, як хрестики-нулики, де кожен гравець виграє, програє або грає внічию. Якщо гравець А може виграти за один хід, то це його найкращий хід. Якщо гравець Б знає, що один хід ставить гравця А в позицію, де він може виграти за один хід, а інший хід ставить гравця А в позицію, де він може претендувати на нічию за найкращих умов, то найкращий хід гравця Б - це хід, який призводить до нічиєї. В кінці гри легко побачити, який хід є "найкращим"; алгоритм мінімаксу допомагає працювати

в зворотному напрямку від кінця гри, щоб знайти найкращий хід. На кожному кроці передбачається, що гравець А намагається максимізувати свої шанси на перемогу, а на наступному кроці гравець В намагається мінімізувати цю ймовірність (тобто максимізувати свої шанси на перемогу).

Метод Альфа-бета відсічення.

Основний алгоритм оптимізації перебору є так званий метод alpha-beta. Сенс його полягає в тому, що для отримання оцінки тієї самої точності, як і при повному переборі, зовсім необов'язково розглядати всі варіанти.

До рекурсивної функції додаються дві нові змінні: максимальне значення для білого і максимальне значення для чорного. Наприклад, у випадку білих це значення збільшується, якщо результат перевищує його максимальне значення в даній позиції. Використовуючи цей алгоритм, якщо програма знаходить докази того, що цей хід гірший за попередньо оцінений, вона припиняє оцінювати цей хід взагалі. Такі ходи не потребують подальшої оцінки.

При першому виклику метод (функція) викликається з максимальним вікном. Під час рекурсивного виклику змінні альфа і бета змінюються на протилежні, щоб "звузити" діапазон пошуку.

Альфа-бета-алгоритм будує менше дерево пошуку з оптимальною послідовністю кроків. Вона приблизно дорівнює квадратному кореню з кількості позицій, знайдених під час повного пошуку. Альфа-бета дуже чутливий до порядку переходів. Тому необхідно враховувати, що в найгіршому випадку, тобто коли бета-відсікання буде останнім рухом, альфа-бета перегляне таку ж кількість позицій, що і мінімальна. Швидкість розрахунку також залежить від фактичного діапазону можливих прогнозів. Наприклад, якщо враховується тільки матеріал, то бали за всі ходи, крім прийому, будуть дорівнювати нулю. Це означає, що існує багато точок розриву, особливо якщо прийом враховується першим.

Перебір з нульовим вікном.

Використовується варіант alpha-beta алгоритму з амортизацією відмов. При виклику пошуку alpha-beta вікно приймало значення від -INFINITY до INFINITY. Результат виходить тієї ж точності, що і при повному переборі.

Якщо ми надамо alpha деяке значення, наприклад 0, то beta при нульовому вікні буде alpha+1. Пошук при нульовому вікні набагато швидше, ніж при повному вікні. Це пов'язано з тим, що кількість відсічень буде більшою. Припустимо, ми викликали alpha-beta функцію пошуку з нульовим вікном і отримали деякий результат:

```
score = AlphaBeta(alpha, alpha+1);
```

Якщо score менше або дорівнює alpha, то результат розрахунку при повному вікні буде від -INFINITY до score, включно. Говорячи іншими словами, score – це можливий максимум.

Якщо score більше alpha, то істинний результат буде від score, включно, до INFINITY. score – це наш мінімум.

NegaScout

NegaScout – найпоширеніший на сьогоднішній день алгоритм перебору. Він досить простий і дає деяке прискорення, не вносячи ніякої додаткової похибки. Головна ідея цього методо у відсіканні вузла за перебором з нульовим вікном (від alpha до alpha+1). Якщо результат розрахунку покращує alpha, то знову здійснюємо перебір. Якщо після перебору отриманий результат більше ніж alpha, то це – мінімум, завдяки якому перебір повторюється у межах не від alpha до beta, а від отриманої оцінки до beta.

Ці алгоритми можна використовувати для кожної перерахованої вище гри. Відмінності в правилах ігор та оцінці ігрової позиції. Тому серед поставлених задач і є розробити спосіб оцінки позицій для гри "Реверсі” з подальшою їх реалізацією в програмному продукті.

1.3. Машинне навчання

1.3.1. Методи машинного навчання

Індуктивне логічне програмування

Індуктивне логічне програмування (ІЛП) - це підхід до навчання на основі правил, який використовує логічне програмування як універсальне представлення вхідних прикладів, зворотного поширення та гіпотез. На основі кодування відомих фонових знань і набору прикладів, представлених у вигляді логічної бази даних фактів, система ІЛПвиводить гіпотетичну логічну програму з наслідками для всіх позитивних прикладів і без наслідків для негативних прикладів. Близьким напрямком є індуктивне програмування. Воно використовує всі види мов програмування (не тільки логічне програмування) для вираження гіпотез, наприклад, функціональні програми..

Дерева прийняття рішень

Дерево прийняття рішень являє собою набір правил у ієрархічній, послідовної структурі, де кожному вхідному даному відповідає вузол, що дає рішення (рис. 1.6). Такі дерева бувають бінарними, що по суті є великою кількістю вкладених логічних умов.

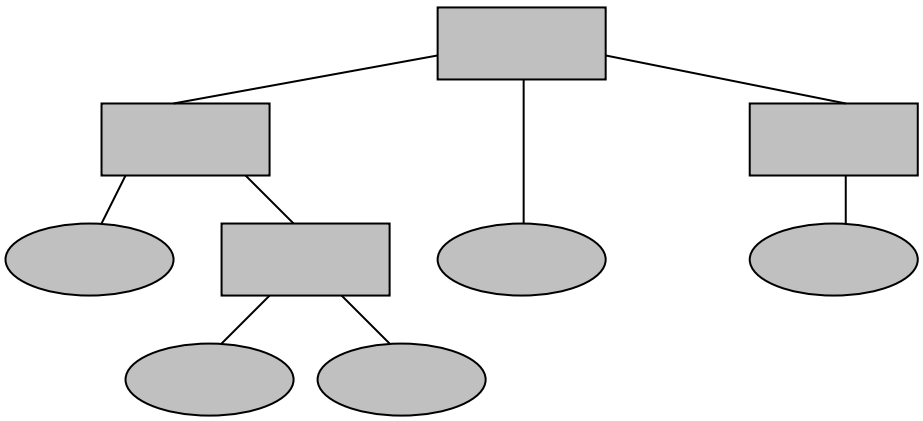

Рис. 1.6 Дерево прийняття рішень

Штучна нейронна мережа.

Штучні нейронні мережі - це математичні моделі та їх програмно-апаратні реалізації, побудовані на функціональних принципах біологічних нейронних мереж (мереж нервових клітин in vivo). Поняття виникло в результаті вивчення процесів, що відбуваються в мозку, і спроб моделювання цих процесів.

ШНМ - це система простих процесорів (штучних нейронів), які взаємодіють між собою (рис. 1.7). Такі процесори зазвичай дуже прості. Кожен процесор у такій мережі цікавиться лише сигналами, які він регулярно отримує, і сигналами, які він регулярно надсилає іншим процесорам. Однак, будучи підключеними до досить великої мережі через контрольовану взаємодію, такі локально прості процесори можуть об'єднуватися для виконання дуже складних завдань.

Нейронні мережі здатні до навчання, а не до програмування в звичайному розумінні. Здатність до навчання є однією з головних переваг нейронних мереж над традиційними алгоритмами. Технічно навчання полягає у визначенні коефіцієнтів зв'язку між нейронами. У процесі навчання нейронна мережа може виявляти складні залежності між вхідними та вихідними даними, а також може узагальнювати. Це означає, що якщо мережа успішно навчена, вона може давати точні результати на основі даних, яких немає в навчальній вибірці.

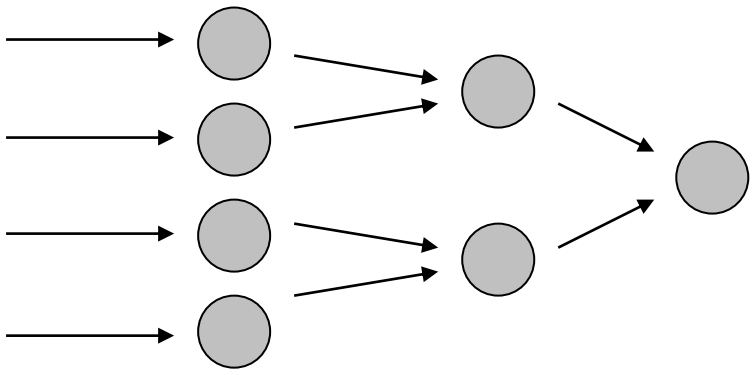

Рис 1.7 Штучна нейронна мережа

Елементи мережі:

- Синапс зручно представити як об'єкт, схожий на чергу (канал) і має певний пріоритет. Він потрібен для обміну інформацією між нейронами.
- Нейрон - об'єкт, який приймає значення з синапсів, робить певні обчислення і віддає їх результат у синапси або аксони.
- Аксон - об'єкт, схожий на синапс, але відрізняється тим, що в нього можна тільки поміщати дані з нейромережі, забирає ж їх хтось із зовні.

Дельта-правило.

Дельта-правилом називають математичну, трохи загальнішу форму запису правил Хеба. Нехай вектор $X = x_1, x_2, \ldots x_m$ — вектор вхідних сигналів, а вектор $D = d_1, d_2, \ldots d_n$ — вектор сигналів, які повинні бути отримані від перцептрона під впливом вхідного вектора. Тут n — кількість нейронів, що входять до перцептрону. Вхідні сигнали, поступивши на входи перцептрону, були зважені і підсумовані, в результаті чого отримано вектор $Y = y_1, y_2, \ldots y_n$ вихідних значень перцептрона. Тоді можна визначити вектор помилки $E = e_1, e_2, \ldots e_n$, розмірність якого збігається розмірністю вектором вихідних сигналів. Компоненти вектора помилок визначаються як різниця між очікуваним і реальним значенням вихідного сигналу нейрону перцептрона. За таких позначеннях формулу (1.1) для коригування j-ї ваги і-го нейрона можна записати так:

$$\omega_j (t + 1) = \omega_j(t) + e_i x_j \qquad (1.1)$$

Номер сигналу j змінюється в межах від одиниці до розмірності вхідного вектора m. Номер нейрону змінюється в межах від одиниці до кількості нейронів n. Величина t — номер поточної ітерації навчання. Таким чином, вага вхідного сигналу нейрона змінюється у бік зменшення помилки пропорційно величині сумарної помилки нейрона. Часто вводять коефіцієнт пропорційності η, на який множиться величина помилки. Цей коефіцієнт

називають швидкістю навчання. Таким чином, підсумкова формула (1.2) для коректування ваг зв'язків перцептрону має наступний вигляд:

$$\omega_j (t + 1) = \omega_j(t) + \eta e_i x_j \qquad (1.2)$$

1.3.2. Навчання штучної нейронної мережі

Точність.

Не завжди необхідно отримати найправильнішу відповідь. Залежно від призначення, одного наближення може бути достатньо. У таких випадках використання більш наближеного методу може значно скоротити час обробки. Ще однією перевагою більш наближених методів є те, що вони, природно, уникають надмірної точності.

Час навчання.

Час або хвилини, необхідні для навчання моделі, сильно варіюються від алгоритму до алгоритму. Час навчання зазвичай тісно пов'язаний з точністю, і одне часто впливає на інше. Крім того, деякі алгоритми чутливі до кількості точок даних. Там, де час має вирішальне значення, це може вплинути на вибір алгоритму, особливо на великих наборах даних.

Навчання з учителем.

Навчання з учителем – це метод машинного навчання, в якому наявний набір прикладів "стимул-реакція" використовується для навчання системи і визначення "реакції" на "стимул", який не належить до наявного набору прикладів. З кібернетичної точки зору, це свого роду кібернетичний експеримент.

Навчання без вчителя.

Навчання під наглядом - це метод машинного навчання, при якому система, що тестується, спонтанно вчиться виконувати завдання без втручання експериментатора. З точки зору кібернетики, це свого роду кібернетичний експеримент. Як правило, воно підходить для завдань, де відомо лише визначення множини об'єктів (навчальної вибірки) і необхідно виявити внутрішні зв'язки, залежності та закономірності, які існують між об'єктами.

Некероване навчання часто порівнюють з керованим навчанням. У керованому навчанні "правильна відповідь" є обов'язковою для кожного об'єкта навчання, і необхідно знайти залежності між стимулами та реакціями системи.

Навчання з підкріпленням.

Навчання з підкріпленням - це галузь машинного навчання, натхненна поведінковою психологією, яка займається проблемою того, яку дію може зробити програмний агент у певному середовищі, щоб максимізувати суму сприйнятої винагороди. Через загальність проблеми, вона також вивчається в багатьох інших дисциплінах, включаючи теорію ігор, теорію управління, дослідження операцій, теорію інформації, оптимізацію на основі імітаційного моделювання, мультиагентні системи, колективний інтелект, статистику та генетичні алгоритми. У літературі з дослідження операцій та управління один з напрямків навчання з підкріпленням називається наближеним динамічним програмуванням. Проблеми навчання з підкріпленням також вивчалися в теорії оптимального керування, але більша частина цієї роботи зосереджена на існуванні та властивостях оптимальних розв'язків, а не на аспектах навчання та апроксимації. В економіці та теорії ігор навчання з підкріпленням використовується для пояснення того, як виникає рівновага в умовах обмеженої раціональності.

У машинному навчанні ситуація часто формулюється як процес прийняття рішень Маркова (MDP), і в цьому контексті багато алгоритмів

навчання з підкріпленням використовують динамічне програмування. Основна відмінність між класичними методами та алгоритмами навчання з підкріпленням полягає в тому, що останні не потребують знань про МПР і зосереджуються на великих МПР, що зробило б строгі методи непрактичними.

Навчання з підкріпленням відрізняється від стандартного навчання з учителем тим, що правильні пари вхід-вихід ніколи не подаються, а неоптимальна поведінка явно не виправляється. Також акцент робиться на інтерактивній діяльності, включаючи пошук балансу між розвідкою та використанням. Компроміс між дослідженням і використанням у навчанні з підкріпленням найбільш детально вивчався на прикладі задачі про багаторукого бандита і моделей скінченних елементів.

1.4. Постановка задачі

Виконати аналіз існуючих методів стосовно машинного навчання. Серед розглянутих методів, обрати той, який найкраще задовольняє поставленій задачі.

Оскільки проводити навчання програми, вводячи дані власноруч, займе досить велику кількість часу, то слід виконати наступні завдання:

Необхідно створити власний формат з ходами зіграних партій.

Реалізувати функціонал для імпортування тренувальних даних.

Розробити інтуїтивно зрозумілий інтерфейс гри та її навчання.

Висновки до розділу 1

Були розглянуті найпоширеніші методи для реалізації машинного навчання, які використовуються під час вибору найкращого ходу в логічних іграх. В результаті, розглянувши усі переваги та недоліки, було вирішено обрати побудову штучної нейронної мережі. В силу своєї універсальності, є можливість її використання для гри "Реверсі".

РОЗДІЛ 2 РОЗРОБКА АЛГОРИТМУ ГРИ «РЕВЕРСІ»

2.1. Генерація ходів

Здійснення ходу в грі реалізовано завдяки функції f_move, яка приймає п’ять аргументів:

x, y – координати запланованого кроку;

player – передає колір (1 – білий, 2 – чорний) гравця;

mas – двовимірний масив даних, який містить ігрову дошку з фішками;

fstep – змінна визначає тип виконаного ходу (1 – остаточне рішення, 0 – хід виконаний на віртуальній дошці для прогнозування подальшого розвитку гри);

Загальний вигляд алгоритму здійснення ходу на дошці має вигляд:

флаг, відповідальний за здійснений хід, встановлюємо в нуль
якщо знайшли фішку супротивника навколо клітини (x, y), то
якщо існує вільна клітина за цією фішкою (або за рядом фішок), то
додаємо у дане вільне місце фішку свого кольору (згідно player)
усі фішки між доданою та (x, y) перефарбовуємо у свій колір
флаг здіснення ходу приймає значення 1
якщо хід являє собою остаточне рішення, то
якщо запланований хід був виконаний успішно (флаг дорівнює 1), то
передаємо хід іншому гравцю
здійснюємо перевірку на кінець гри
якщо гра закінчилась, то

обираємо переможця

Кінець гри відбудеться лише у двох випадках: якщо дошка немає пустих клітин, або якщо один з гравців втрачає усі фішки.

Тому алгоритм перевірки кінця гри має наступний вид:

флаг встановлюємо в нуль

змінна p1, відповідальна за кількість фішок першого гравця, дорівнює нулю

змінна p2, відповідальна за кількість фішок другого гравця, дорівнює нулю

перебираємо масив дошки

якщо клітина не містить фішки, то

флаг дорівнює одиниці

якщо клітина містить фішку першого гравця, то

збільшуємо значення p1 на одиницю

якщо клітина містить фішку другого гравця, то

збільшуємо значення p2 на одиницю

якщо p1 або p2 дорівнює нулю, то

флаг приймає значення нуль

якщо флаг дорівнює нулю, то

якщо p1 більше p2, то

переможець гравець номер 1

якщо p1 менше p2, то

переможець гравець номер 2

якщо p1 дорівнює p2, то

грає закінчується нічиєю

Наступною важливою функцією є сканування ймовірно можливих ходів для гравця, де ймовірно можливий хід позначено числом 3:

перебираємо масив дошки
якщо клітина не містить фішки, то
якщо клітина не біля верхньої границі, то
якщо зверху від клітини фішка супротивника, то
клітина приймає значення 3
якщо клітина не біля правої чи верхньої границь, то
якщо справа зверху від клітини фішка супротивника, то
клітина приймає значення 3
якщо клітина не біля правої границі, то
якщо справа від клітини фішка супротивника, то
клітина приймає значення 3
якщо клітина не біля правої чи нижньої границь, то
якщо справа знизу від клітини фішка супротивника, то
клітина приймає значення 3
якщо клітина не біля нижньої границі, то
якщо знизу від клітини фішка супротивника, то
клітина приймає значення 3
якщо клітина не біля лівої чи нижньої границь, то
якщо зліва внизу від клітини фішка супротивника, то
клітина приймає значення 3
якщо клітина не біля лівої границі, то
якщо зліва від клітини фішка супротивника, то
клітина приймає значення 3
якщо клітина не біля лівої чи верхньої границь, то
якщо зліва зверху від клітини фішка супротивника, то
клітина приймає значення 3

Таким чином, після знаходження всіх ймовірних можливих ходів, масив містить клітини які не являють собою можливого для здійснення ходу (рис. 2.1).

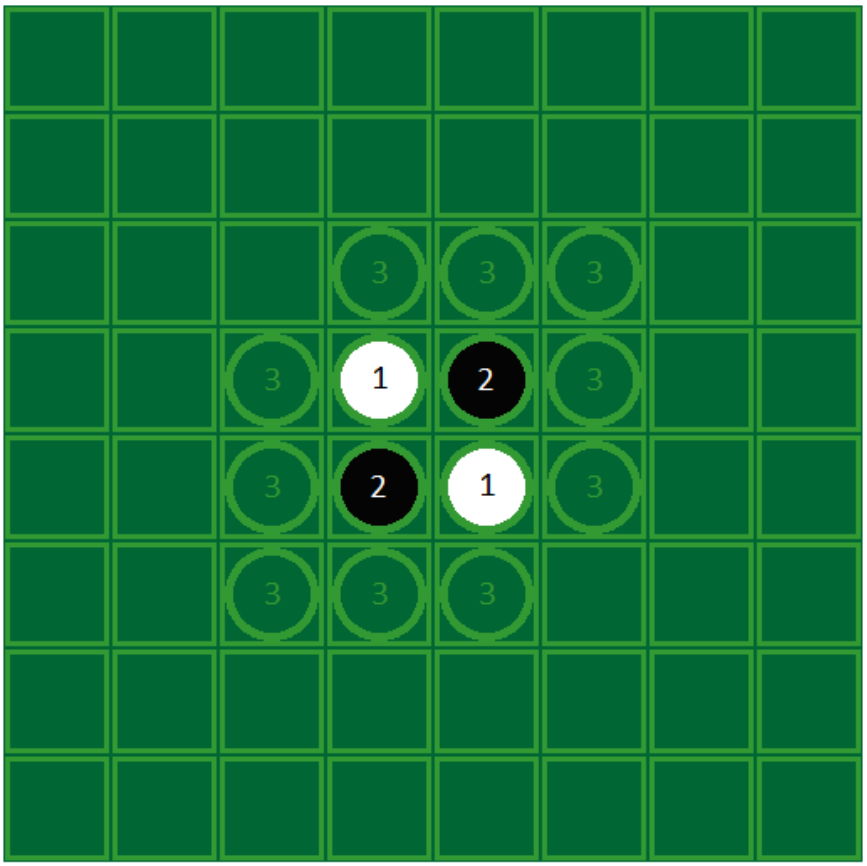

Рис. 2.1 Приклад ймовірних можливих ходів для білих

Тому, при знаходженні саме можливих ходів, використовується наступний підхід: якщо при визові функції f_move, яка здійснює хід в указану клітину (в нашому випадку ту, що містить значення 3), є зміни на дошці, то цей хід є можливим. Це зумовлено, тим що функції не виконується у разі неможливості здійсненого ходу. Алгоритм аналізу змін на дошці має наступний вигляд:

змінна score_before дорівнює нулю

змінна score_after дорівнює нулю

перебираємо масив дошки

якщо клітина містить фішку гравця, то

збільшуємо значення score_before на одиницю

викликаємо функцію f_move для даної позиції

перебираємо масив дошки

якщо клітина містить фішку гравця, то

збільшуємо значення score_after на одиницю

У разі нерівності змінних score_before та score_after хід є можливим для здійснення. Таким чином усі інші клітини позначені як «ймовірні можливі» відсіюються (рис. 2.2).

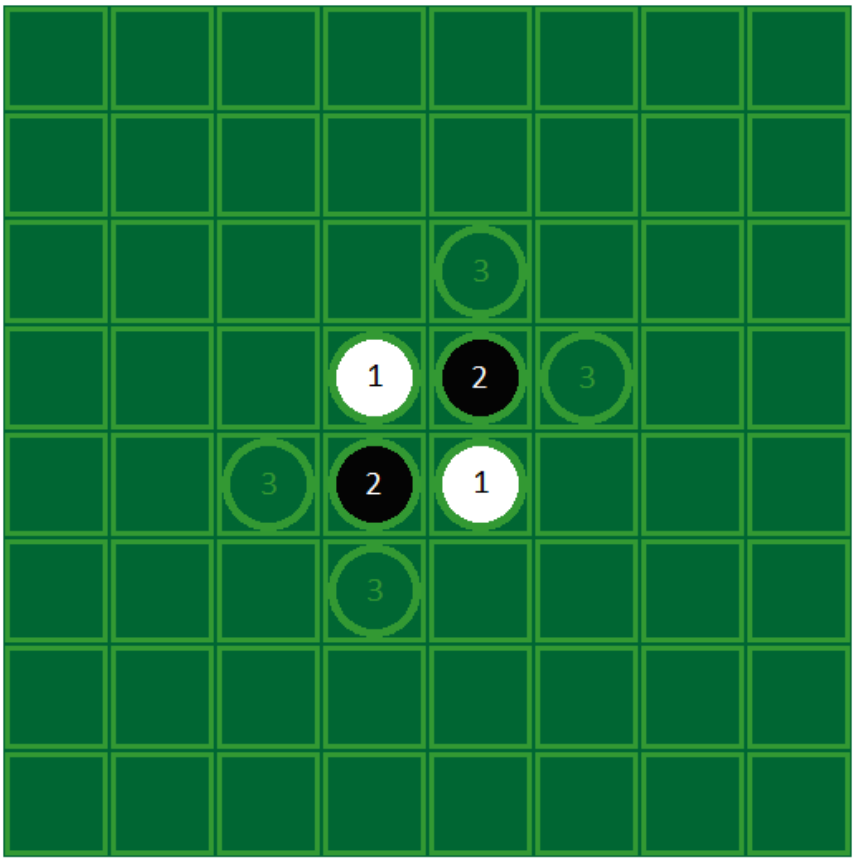

Рис. 2.2 Приклад можливих ходів для білих

Прогнозування розвитку гри здійснюється за допомогою використання рекурсивної функції (рис. 2.3) з динамічним заглиблення, відносно того яка складність гри була обрана користувачем.

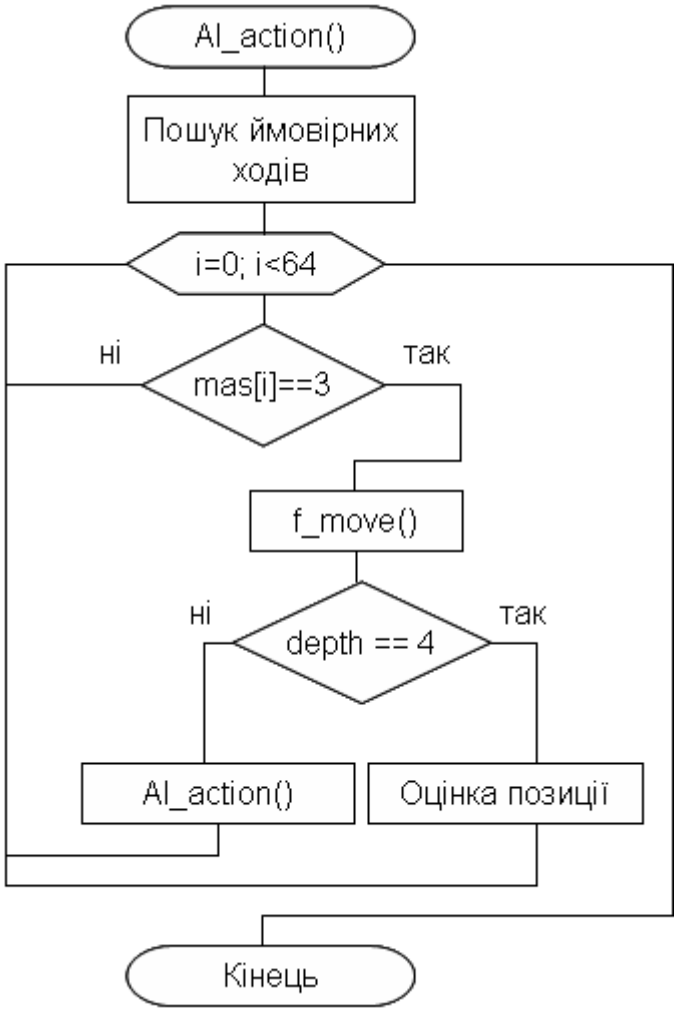

Рис. 2.3 Блок-схема рекурсивної функції

Функція має ім'я «AI_action» і приймає наступні параметри:

mas_original – двовимірний масив даних, який містить ігрову дошку з фішками та можливими ходами;

depth – глибина рекурсії;

new_play_now – передає колір (1 – білий, 2 – чорний) гравця за якого виконується хід;

Загальний вигляд алгоритму генерації ходів на дошці має вигляд:

копіюємо прийнятий масив mas_original у локальний массив mas

перебираємо масив mas

якщо клітина не містить фішки, то

знаходимо ймовірні можливі ходи
перебираємо масив mas
якщо клітина помічена як ймовірний можливий хід, то
копіюємо масив mas_original з назвою tmas
підраховуємо кількість фішок у змінну score_before
викликаємо функцію f_move для масиву tmas
підраховуємо кількість фішок у змінну score_after
якщо score_before не дорівнює score_after, то
якщо depth дорівнює нулю, то зберігаємо здійснений хід
якщо depth дорівнює максимальній глибині, то
оцінюємо позицію
інакше
рекурсивно викликаємо функцію AI_action
обираємо найкращій хід

2.2. Оцінка позиції

Розглянуті тут ігри один на один зазвичай характеризуються набором "позицій" і набором правил переходу з однієї позиції в іншу, при цьому передбачається, що гравці ходять по черзі. Вважається, що правила допускають лише скінченну послідовність позицій, а кожна позиція має лише скінченну кількість ходів. Тоді для кожної позиції p існує таке число N(p), що не більше N(p) партій, починаючи з p, можуть продовжуватись.

Позиція ендшпіля - це позиція, у якій заборонено робити ходи. Кожна позиція має цілочисельну функцію f(p), яка визначає виграш гравця, який робить хід у цій позиції, а виграш інших гравців дорівнює -f(p).

Якщо з позиції p при p_d дозволено ходи $p_1,...,p_d$, то виникає проблема вибору найкращого ходу. Припускаючи, що суперники обирають найкращий хід для себе (в тому ж сенсі), хід, який призводить до найбільшого виграшу в

кінці гри, називається найкращим; нехай F(p) - максимальний виграш, який наступний гравець може отримати на позиції p при оптимальному захисті. Отже, після переміщення на позицію pi виграш цього гравця дорівнює -F(pi) і, отже, формула 2.1.

$$F(p) = \begin{cases} f(p) & d = 0, \\ \max\{-F(p_1),...,-F(p_d)\}, & \text{якщо } d > 0. \end{cases} \qquad (2.1)$$

Ця формула дозволяє індуктивно визначити F(p) для кожної позиції p. Так як нам звичайно легше оцінювати позиції з погляду одного якогось гравця, іноді зручніше використовувати "мінімаксний" підхід.

З точки зору програмної реалізації, було розроблено декілька алгоритмів, обов’язком яких є оцінка позиції гравця в залежності від обраної складності гри.

Перший рівень складності має назву «легкий». У цьому випадку увага надається лише розташуванню фішок й маємо наступну оцінку за кожну з них:

Будь-яка фішка надає 1 очко;

Кожна фішка, що розташована вздовж границі дошки дає 3 очка:

Кожна фішка, що розташована у кутових позиція дає 15 очок;

Загальний вигляд алгоритму оцінки позиції має вигляд:

перебираємо масив дошки

якщо у клітині фішка гравця, то

визначаємо вагу даної позиції

додаємо відповідне значення до змінної score

якщо у клітині фішка супротивника, то

визначаємо вагу даної позиції

віднімаємо відповідне значення від змінної score

Таким чином, кожна з фішок має свою вагу, і залежно від положення фішки (або ряду фішок в цілому) вага приймає більше або менше значення стосовно нуля.

Також слід зазначити, що даний рівень передбачає прогнозування розвитку гри на два кроки вперед.

Другий рівень складності має назву «середній». Він включає в себе вище названі методи оцінювання, але також враховує кількість можливих ходів суперника.

Таким чином, алгоритм приймає наступний вигляд:

```
перебираємо масив дошки
    якщо у клітині фішка гравця, то
        визначаємо вагу даної позиції
        додаємо відповідне значення до змінної score
    якщо у клітині фішка супротивника, то
        визначаємо вагу даної позиції
        віднімаємо відповідне значення від змінної score
    якщо не містить фішки, то
        якщо клітина позначена як можливий хід гравця, то
            додаємо відповідне значення до змінної score
        якщо клітина позначена як можливий хід супротивника, то
            віднімаємо відповідне значення від змінної score
```

Слід також додати, що рівень заглиблення на цьому рівні складності дорівнює чотирьом. Такий підхід надає змогу краще спрогнозувати, яку лінію розвитку гри потрібно обрати для того, щоб у майбутньому супротивник мав меншу кількість ходів для вибору.

Третій рівень, він же «складний», має змогу використовувати різні стратегії, в залежності від етапу гри.

Наприклад, під час початкової стадії, коли дошка є незаповненою, то перевага надається скороченню кількості можливих ходів супротивника. У

середині гри пріоритети змінюється й вага фішок стає більше значущою. А вже ближче до кінця партії, програма намагається зосередитися лише на ключових позиціях фішок та їх кількості.

Саме тому алгоритм приймає більш складний вигляд ніж раніше:

визначаємо на якому етапі знаходиться гра
якщо етап гри початковий, то
    коефіцієнти можливих ходів приймають більше значення
    перебираємо масив дошки
        якщо у клітині фішка гравця, то
            визначаємо вагу даної позиції
            додаємо відповідне значення до змінної score
        якщо у клітині фішка супротивника, то
            визначаємо вагу даної позиції
            віднімаємо відповідне значення від змінної score
        якщо не містить фішки, то
            якщо клітина позначена як можливий хід гравця, то
                додаємо відповідне значення до змінної score
            якщо клітина позначена як можливий хід супротивника, то
                віднімаємо відповідне значення від змінної score
якщо етап гри середній, то
    коефіцієнти можливих ходів приймають менше значення
    перебираємо масив дошки
        якщо у клітині фішка гравця, то
            визначаємо вагу даної позиції
            додаємо відповідне значення до змінної score
        якщо у клітині фішка супротивника, то
            визначаємо вагу даної позиції

```
                віднімаємо відповідне значення від змінної score
            якщо не містить фішки, то
                якщо клітина позначена як можливий хід гравця, то
                    додаємо відповідне значення до змінної score
                якщо клітина позначена як можливий хід
супротивника, то
                    віднімаємо відповідне значення від змінної
score
якщо етап гри кінцевий, то
    перебираємо масив дошки
        якщо у клітині фішка гравця, то
            визначаємо вагу даної позиції
            додаємо відповідне значення до змінної score
        якщо у клітині фішка супротивника, то
            визначаємо вагу даної позиції
            віднімаємо відповідне значення від змінної score
```

РОЗДІЛ 3 РОЗРОБКА АЛГОРИТМУ САМОНАВЧАННЯ

3.1. Серверна частина

Серверна части являє собою додаток, розроблений на платформі «Node.js». Сервер має наступний алгоритм:

ініціалізація файлової частни
ініціалізація http протоколу для публікації клієнтської частини
підключення бібліотеки socket.io
підключення бази даних
прослуховування socket портів

Програмно це виглядає наступним чином:

```
var fs = require('fs');
var express = require('express');
var app = express();
app.use(express.static(__dirname + "/public"));
var httpServer = require('http').Server(app);
httpServer.listen(80);
var io = require('socket.io')(httpServer);
var position = JSON.parse(fs.readFileSync('db.json', 'utf8'));
var w = JSON.parse(fs.readFileSync('w.json', 'utf8'));

io.on('connection', function (socket) {
    ...
}
```

Здійснення самонавчання у грі «Реверсі» реалізовано завдяки дельта-правилу. Слідуючи з цього, потрібно об’явити наступні змінні, які зберігатимуться у базі даних:

```
var w1 = w.w1, w2 = w.w2, w3 = w.w3;
var T=5, L=0.1;
```

Де об’єкт w містить вагові коефіцієнти, а w1, w2, w2 коефіцієнти виграних, програних та зіграних в нічию, партій відповідно. T – порогове значення, L – коефіцієнт швидкості навчання.

База даних заграних партій представляє собою файл формату JSON з назвою «db.json». Вона містить дані про позицію, де:

0 – пуста клітина;

1 – Перший гравець;

2 – Другий гравець;

3 – позиція зробленого ходу.

Також база даних включає в себе статистику виграних, програних та зіграних в нічию, партій.

Після підключення гравця до серверу, кожен згенерований програмою хід звіряється з сервером, який в свою черго перевіряє чи була ця позиція раніше, і з якою ймовірністю програма має шанс на перемогу. За це відповідає функція check_move:

```
socket.on('check_move', function (data) {
if (position[data] !== undefined) {
var k = 1/(position[data].w+position[data].l);
var x1 = k*position[data].w;
var x2 = k*position[data].l;
var x3 = k*position[data].n;
var T_new = x1*w1 + x2*w2 + x3*w3;
if (T_new >= T) {
     socket.emit('action', '1');
```

```
} else {
     socket.emit('action', '0');
}} else {
log('Unknow');
socket.emit('action', '1');
}});
```

Якщо дана позиція міститься у базі і задовольняє порогове значення T, то надати можливість здійснення ходу. В іншому випадку надіслати команду, що слід здійснити хід з більшим рівнем заглиблення.

По закінченню гри, сервер приймає усі позиція партії. Після чого, у разі перемоги, проводиться зміна вагових коефіцієнтів, таким чином програма самонавчається, роблячи висновки на хибних прийнятих рішеннях:

```
for (var i in data) {
if (position[i] === undefined) {
     position[i] = {w:0, l:0, n:0};
}
if (socket.who_win==2) {
     position[i].w++;
     var k = 1/(position[i].w+position[i].l+ position[i].n);
     var x1 = k*position[i].w;
     var x2 = k*position[i].l;
     var x3 = k*position[i].n;
     var T_new = x1*w1 + x2*w2 + x3*w3;
     var D = T-T_new;
     w1 = w1 + x1*D*L;
     w2 = w2 + x2*D*L;
     w3 = w3 + x3*D*L;
     log('Новые весовые: '+w1+' / '+w2+' / '+w3);
} else    {
     position[i].l++;
}
}
```

Після уточнення коефіцієнтів, усі дані зберігаються у відповідні файли формату JSON:

```
var position_json = JSON.stringify(position);
fs.writeFile('db.json', position_json, 'utf8');
fs.writeFile(
     'w.json',
     '{"w1":'+w1+',"w2":'+w2+',"w3":'+w3+'}',
     'utf8'
);
```

3.2. Клієнтська частина

Генерація ходів

Здійснення ходу в грі реалізовано завдяки функції f_move, яка приймає п’ять аргументів:

x, y – координати запланованого кроку;

player – передає колір (1 – білий, 2 – чорний) гравця;

mas – двовимірний масив даних, який містить ігрову дошку з фішками;

fstep – змінна визначає тип виконаного ходу (1 – остаточне рішення, 0 – хід виконаний на віртуальній дошці для прогнозування подальшого розвитку гри);

Загальний вигляд алгоритму здійснення ходу на дошці має вигляд:

флаг, відповідальний за здійснений хід, встановлюємо в нуль
якщо знайшли фішку супротивника навколо клітини (x, y), то
якщо існує вільна клітина за цією фішкою (або за рядом фішок), то
додаємо у дане вільне місце фішку свого кольору (згідно player)

усі фішки між доданою та (x, y) перефарбовуємо у свій колір

флаг здіснення ходу приймає значення 1

якщо хід являє собою остаточне рішення, то

якщо запланований хід був виконаний успішно (флаг дорівнює 1), то

передаємо хід іншому гравцю

здійснюємо перевірку на кінець гри

якщо гра закінчилась, то

обираємо переможця

Кінець гри відбудеться лише у двох випадках: якщо дошка немає пустих клітин, або якщо один з гравців втрачає усі фішки.

Тому алгоритм перевірки кінця гри має наступний вид:

флаг встановлюємо в нуль

змінна p1, відповідальна за кількість фішок першого гравця, дорівнює нулю

змінна p2, відповідальна за кількість фішок другого гравця, дорівнює нулю

перебираємо масив дошки

якщо клітина не містить фішки, то

флаг дорівнює одиниці

якщо клітина містить фішку першого гравця, то

збільшуємо значення p1 на одиницю

якщо клітина містить фішку другого гравця, то

збільшуємо значення p2 на одиницю

якщо p1 або p2 дорівнює нулю, то

флаг приймає значення нуль

якщо флаг дорівнює нулю, то

якщо p1 більше p2, то

переможець гравець номер 1

якщо p1 менше p2, то

переможець гравець номер 2

якщо p1 дорівнює p2, то

грає закінчується нічиєю

Наступною важливою функцією є сканування ймовірно можливих ходів для гравця, де ймовірно можливий хід позначено числом 3:

перебираємо масив дошки

якщо клітина не містить фішки, то

якщо клітина не біля верхньої границі, то

якщо зверху від клітини фішка супротивника, то

клітина приймає значення 3

якщо клітина не біля правої чи верхньої границь, то

якщо справа зверху від клітини фішка супротивника, то

клітина приймає значення 3

якщо клітина не біля правої границі, то

якщо справа від клітини фішка супротивника, то

клітина приймає значення 3

якщо клітина не біля правої чи нижньої границь, то

якщо справа знизу від клітини фішка супротивника, то

клітина приймає значення 3

якщо клітина не біля нижньої границі, то

якщо знизу від клітини фішка супротивника, то

клітина приймає значення 3

якщо клітина не біля лівої чи нижньої границь, то

якщо зліва внизу від клітини фішка супротивника, то
клітина приймає значення 3
якщо клітина не біля лівої границі, то
якщо зліва від клітини фішка супротивника, то
клітина приймає значення 3
якщо клітина не біля лівої чи верхньої границь, то
якщо зліва зверху від клітини фішка супротивника, то
клітина приймає значення 3

Таким чином, після знаходження всіх ймовірних можливих ходів, масив містить клітини які не являють собою можливого для здійснення ходу (рис. 3.1).

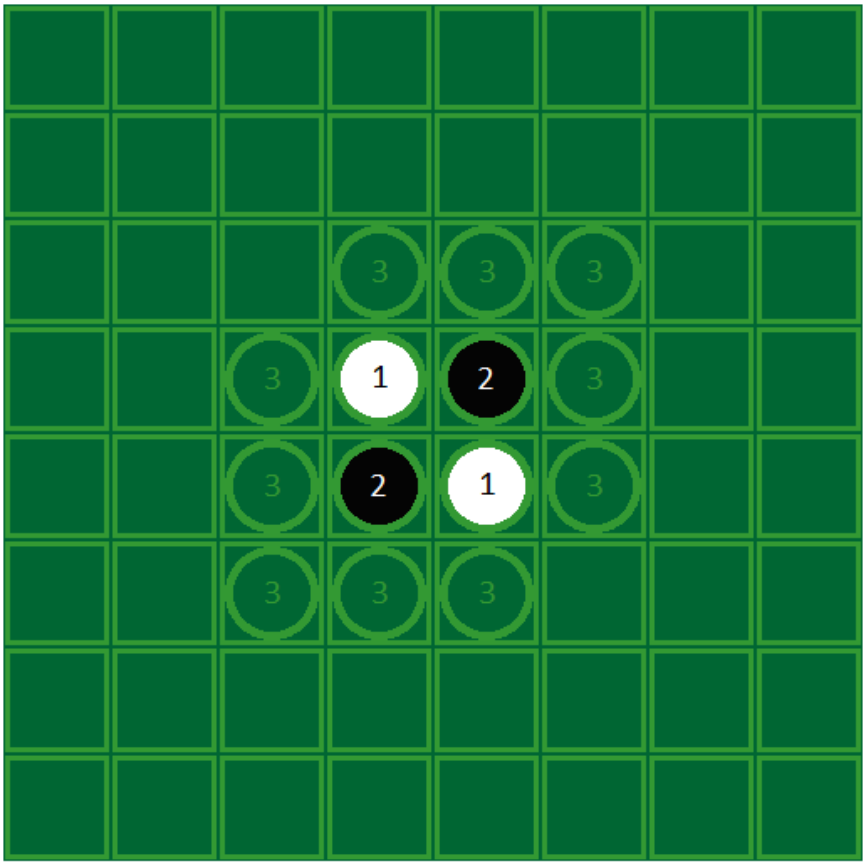

Рис. 3.1 Приклад ймовірних можливих ходів для білих

Тому, при знаходженні саме можливих ходів, використовується наступний підхід: якщо при визові функції f_move, яка здійснює хід в указану клітину (в нашому випадку ту, що містить значення 3), є зміни на дошці, то цей хід є можливим. Це зумовлено, тим що функції не виконується у разі неможливості здійсненого ходу. Алгоритм аналізу змін на дошці має наступний вигляд:

```
змінна score_before дорівнює нулю
змінна score_after дорівнює нулю
перебираємо масив дошки
        якщо клітина містить фішку гравця, то
                збільшуємо значення score_before на одиницю
викликаємо функцію f_move для даної позиції
перебираємо масив дошки
        якщо клітина містить фішку гравця, то
                збільшуємо значення score_after на одиницю
```

У разі нерівності змінних score_before та score_after хід є можливим для здійснення. Таким чином усі інші клітини позначені як «ймовірні можливі» відсіюються (рис. 3.2).

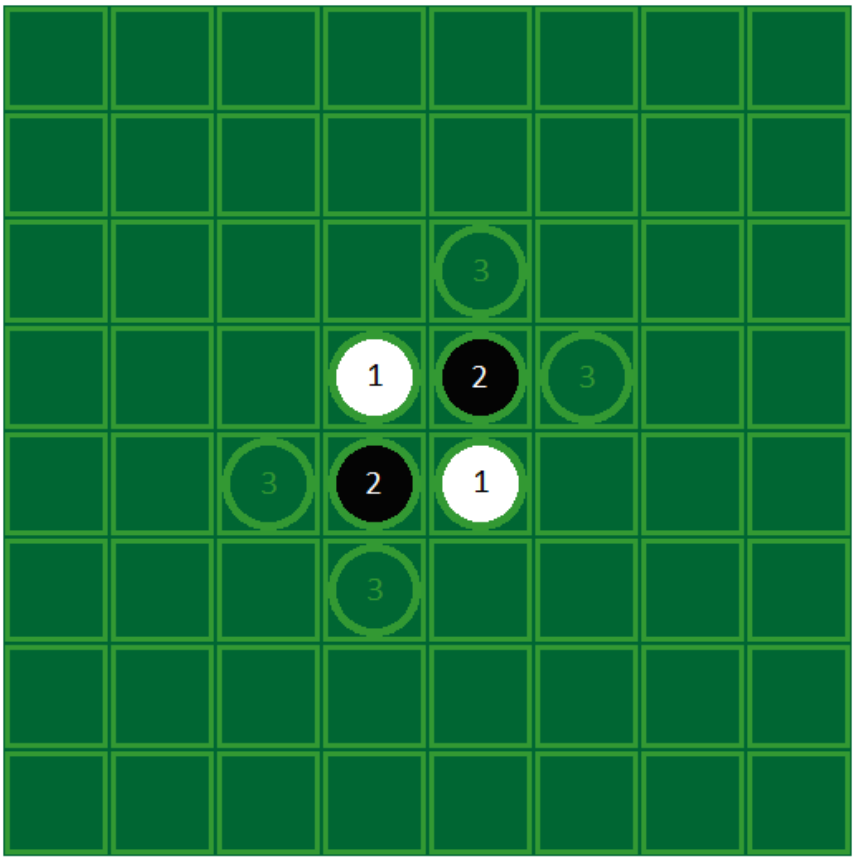

Рис. 3.2 Приклад можливих ходів для білих

Прогнозування розвитку гри здійснюється за допомогою використання рекурсивної функції (рис. 3.3) з динамічним заглиблення, відносно того яка складність гри була обрана користувачем.

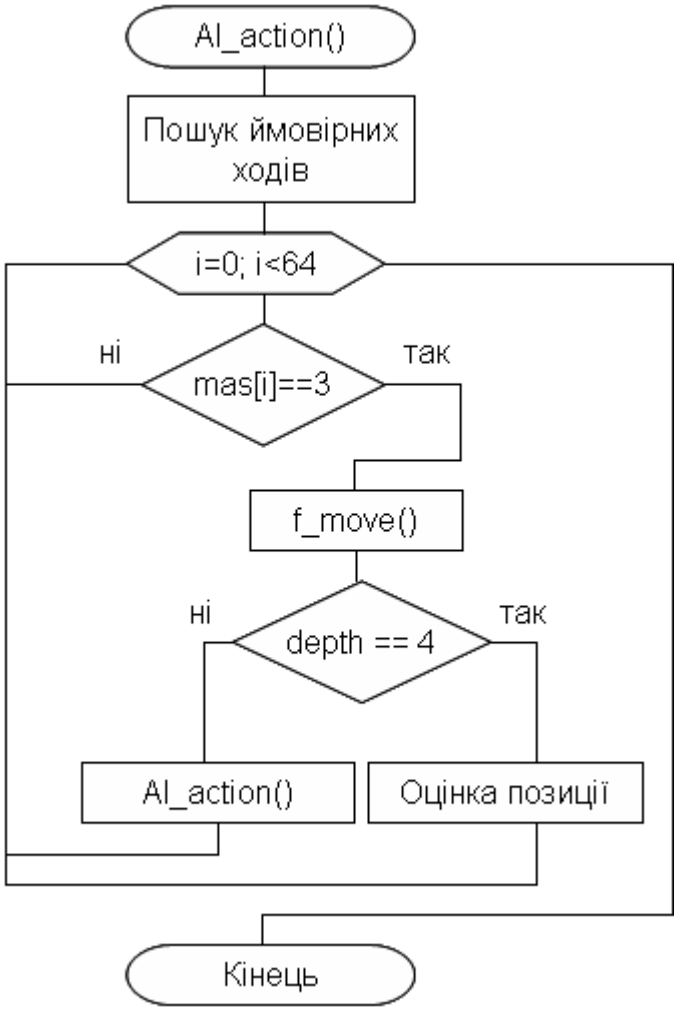

Рис. 3.3 Блок-схема рекурсивної функції

Функція має ім'я «AI_action» і приймає наступні параметри:

mas_original – двовимірний масив даних, який містить ігрову дошку з фішками та можливими ходами;

depth – глибина рекурсії;

new_play_now – передає колір (1 – білий, 2 – чорний) гравця за якого виконується хід;

Загальний вигляд алгоритму прийняття рішень має вигляд (рис. 2.4):

копіюємо прийнятий масив mas_original у локальний массив mas

перебираємо масив mas

якщо клітина не містить фішки, то

знаходимо ймовірні можливі ходи

перебираємо масив mas

якщо клітина помічена як ймовірний можливий хід, то

копіюємо масив mas_original з назвою tmas

підраховуємо кількість фішок у змінну score_before

викликаємо функцію f_move для масиву tmas

підраховуємо кількість фішок у змінну score_after

якщо score_before не дорівнює score_after, то

якщо depth дорівнює нулю, то зберігаємо здійснений хід

якщо depth дорівнює максимальній глибині, то

оцінюємо позицію

інакше

рекурсивно викликаємо функцію AI_action

обираємо найкращій хід

посилаємо обраний найкращій хід серверу

якщо відповідь сервера дорівнює 1, то

здійснюємо хід

інакше

виклик функції f_move з більшою глибиною

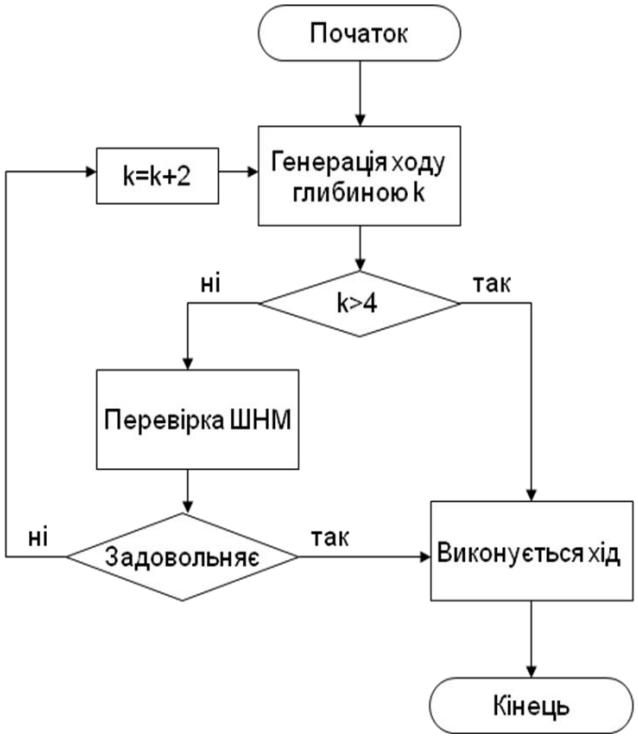

Рис. 2.4 Блок-схема алгоритму прийняття рішень

Оцінка позиції

Розглянуті тут ігри один на один зазвичай характеризуються набором "позицій" і набором правил переходу з однієї позиції в іншу, при цьому передбачається, що гравці ходять по черзі. Вважається, що правила допускають лише скінченну послідовність позицій, а кожна позиція має лише скінченну кількість ходів. Тоді для кожної позиції p існує таке число N(p), що не більше N(p) партій, починаючи з p, можуть продовжуватись.

Позиція ендшпіля - це позиція, у якій заборонено робити ходи. Кожна позиція має цілочисельну функцію f(p), яка визначає виграш гравця, який робить хід у цій позиції, а виграш інших гравців дорівнює -f(p).

Якщо з позиції p при p_d дозволено ходи $p_1,...,p_d$, то виникає проблема вибору найкращого ходу. Припускаючи, що суперники обирають найкращий хід для себе (в тому ж сенсі), хід, який дає найбільший можливий виграш в кінці гри, називається найкращим; нехай F(p) - максимальний виграш, який

гравець, чия черга, може отримати в позиції p при оптимальному захисті. Таким чином, після переходу на позицію p_i виграш цього гравця дорівнює - $F(p_i)$, звідціля маємо формулу 2.1.

$$F(p) = \begin{cases} f(p) & d = 0, \\ \max\{-F(p_1),...,-F(p_d)\}, & \text{якщо } d > 0. \end{cases} \quad (3.1)$$

Ця формула дозволяє індуктивно визначити F(p) для кожної позиції p. Так як нам звичайно легше оцінювати позиції з погляду одного якогось гравця, іноді зручніше використовувати "мінімаксний" підхід.

З точки зору програмної реалізації, було розроблено декілька алгоритмів, обов’язком яких є оцінка позиції гравця в залежності від обраної складності гри.

Перший рівень складності має назву «легкий». У цьому випадку увага надається лише розташуванню фішок й маємо наступну оцінку за кожну з них:

Будь-яка фішка надає 1 очко;

Кожна фішка, що розташована вздовж границі дошки дає 3 очка:

Кожна фішка, що розташована у кутових позиція дає 15 очок;

Загальний вигляд алгоритму оцінки позиції має вигляд:

```
перебираємо масив дошки
    якщо у клітині фішка гравця, то
        визначаємо вагу даної позиції
        додаємо відповідне значення до змінної score
    якщо у клітині фішка супротивника, то
        визначаємо вагу даної позиції
        віднімаємо відповідне значення від змінної score
```

Таким чином, кожна з фішок має свою вагу, і залежно від положення фішки (або ряду фішок в цілому) вага приймає більше або менше значення стосовно нуля.

Також слід зазначити, що даний рівень передбачає прогнозування розвитку гри на два кроки вперед.

Другий рівень складності має назву «середній». Він включає в себе вище названі методи оцінювання, але також враховує кількість можливих ходів суперника.

Таким чином, алгоритм приймає наступний вигляд:

```
перебираємо масив дошки
    якщо у клітині фішка гравця, то
        визначаємо вагу даної позиції
        додаємо відповідне значення до змінної score
    якщо у клітині фішка супротивника, то
        визначаємо вагу даної позиції
        віднімаємо відповідне значення від змінної score
    якщо не містить фішки, то
        якщо клітина позначена як можливий хід гравця, то
            додаємо відповідне значення до змінної score
        якщо клітина позначена як можливий хід супротивника, то
            віднімаємо відповідне значення від змінної score
```

Слід також додати, що рівень заглиблення на цьому рівні складності дорівнює чотирьом. Такий підхід надає змогу краще спрогнозувати, яку лінію розвитку гри потрібно обрати для того, щоб у майбутньому супротивник мав меншу кількість ходів для вибору.

Третій рівень, він же «складний», має змогу використовувати різні стратегії, в залежності від етапу гри.

Наприклад, під час початкової стадії, коли дошка є незаповненою, то перевага надається скороченню кількості можливих ходів супротивника. У

середині гри пріоритети змінюється й вага фішок стає більше значущою. А вже ближче до кінця партії, програма намагається зосередитися лише на ключових позиціях фішок та їх кількості.

Саме тому алгоритм приймає більш складний вигляд ніж раніше:

```
визначаємо на якому етапі знаходиться гра
якщо етап гри початковий, то
    коефіцієнти можливих ходів приймають більше значення
    перебираємо масив дошки
        якщо у клітині фішка гравця, то
            визначаємо вагу даної позиції
            додаємо відповідне значення до змінної score
        якщо у клітині фішка супротивника, то
            визначаємо вагу даної позиції
            віднімаємо відповідне значення від змінної score
        якщо не містить фішки, то
            якщо клітина позначена як можливий хід гравця, то
                додаємо відповідне значення до змінної score
            якщо клітина позначена як можливий хід
супротивника, то
                віднімаємо відповідне значення від змінної
score
якщо етап гри середній, то
    коефіцієнти можливих ходів приймають менше значення
    перебираємо масив дошки
        якщо у клітині фішка гравця, то
            визначаємо вагу даної позиції
            додаємо відповідне значення до змінної score
        якщо у клітині фішка супротивника, то
            визначаємо вагу даної позиції
```

віднімаємо відповідне значення від змінної score
якщо не містить фішки, то
якщо клітина позначена як можливий хід гравця, то
додаємо відповідне значення до змінної score
якщо клітина позначена як можливий хід супротивника, то
віднімаємо відповідне значення від змінної score
якщо етап гри кінцевий, то
перебираємо масив дошки
якщо у клітині фішка гравця, то
визначаємо вагу даної позиції
додаємо відповідне значення до змінної score
якщо у клітині фішка супротивника, то
визначаємо вагу даної позиції
віднімаємо відповідне значення від змінної score

3.3. Вибір середовища

Програмна платформа Node.js

Node.js призначений для відокремленого виконання високопродуктивних мережних застосунків на мові JavaScript.

Node.js характеризується такими властивостями:

- асинхронна однопотокова модель виконання запитів
- неблокуючий ввід/вивід
- система модулів CommonJS
- рушій JavaScript Google V8

Можливості платформи не обмежуються створенням серверних скриптів для Інтернету, але також можуть бути використані для створення загальних клієнтських і серверних мережевих додатків. Для виконання

JavaScript-коду використовується движок V8, розроблений компанією Google.

Для надійної обробки великої кількості паралельних запитів Node.js використовує асинхронну модель виконання коду, засновану на обробці подій в неблокуючому режимі і визначенні обробників зворотного виклику..

Мова програмування JavaScript

JavaScript (JS) - це динамічна об'єктно-орієнтована мова програмування, яка є реалізацією стандарту ECMAScript. Зазвичай використовується як частина браузера, де код на стороні браузера може взаємодіяти з користувачем, керувати браузером, асинхронно обмінюватися даними з сервером і змінювати структуру і зовнішній вигляд веб-сторінок. JavaScript також використовується для програмування на стороні сервера (подібно до таких мов програмування, як Java і C#), розробки ігор, настільних і мобільних додатків, написання сценаріїв у прикладному програмному забезпеченні і в PDF-документах.

JavaScript класифікується як мова прототипування (підмножина об'єктно-орієнтованих), динамічно типізованих скриптових мов програмування. На додаток до прототипування, JavaScript частково підтримує інші парадигми програмування (імперативну і частково функціональну) і має різні пов'язані з ними динамічну і слабку типізацію, автоматичне управління пам'яттю, успадкування прототипів, функції як першокласні об'єкти і т.д. Він також має архітектурні особливості.

JavaScript має багато рис об'єктно-орієнтованої мови, але його підтримка об'єктів відрізняється від традиційних мов ООП завдяки концепції прототипів. Крім того, JavaScript має багато особливостей, характерних для функціональних мов - функції як першокласні об'єкти, об'єкти як списки, каррі, анонімні функції та закриття - що додає мові додаткової гнучкості.

JavaScript має C-подібний синтаксис, але в порівнянні з мовою Сі має такі корінні відмінності:

- об'єкти, з можливістю інтроспекції і динамічної зміни типу через механізм прототипів
- функції як об'єкти першого класу
- обробка винятків
- автоматичне приведення типів
- автоматичне прибирання сміття
- анонімні функції

JavaScript має різні вбудовані об'єкти: Global, Object, Error, Function, Array, String, Boolean, Number, Math, Date, RegExp. Існує ряд вбудованих операцій, які не обов'язково є функціями або методами, а також ряд вбудованих операторів, які керують логікою виконання програми. Синтаксис JavaScript в основному відповідає синтаксису Java (тобто, в кінцевому рахунку, він успадкований від мови C), але мова сценаріїв спрощена в порівнянні з Java, щоб полегшити вивчення мови сценаріїв. Наприклад, оголошення змінних не мають типів, властивості не мають типів, а оголошення функцій можуть стояти після функцій у тексті програми.

Семантика мови подібна до самописних скриптів..

Бібліотека Socket.IO

Socket.IO - це JavaScript бібліотека для веб-додатків та обміну даними в режимі реального часу. Вона складається з двох компонентів: клієнтського, що працює в браузері, і серверного для Node.js. Обидва компоненти мають схожі API; як і node.js, Socket.IO керується подіями.

Socket.IO в основному використовує протокол WebSocket, але також використовує інші методи, коли це необхідно, такі як сокети Adobe Flash, JSONP запити, AJAX запити і т.д. Socket.IO не тільки використовується як обгортка для WebSockets, але також здійснює трансляцію на декілька сокетів, зберігання даних, пов'язаних з кожним клієнтом, асинхронний ввід/вивід і багато іншого.

JSON

JSON - це текстовий формат, який використовується для обміну даними між комп'ютерами. JSON базується на тексті і є зрозумілим для людини. Формат дозволяє визначати об'єкти та інші структури даних. Формат в основному використовується для передачі структурованої інформації через мережі.

Основне застосування JSON знайшов у визначенні веб-додатків, а саме при використанні технології AJAX: JSON, що використовується в AJAX, замінює XML (який використовується в AJAX) при асинхронній передачі структурованої інформації між клієнтом і сервером. При цьому переваги JSON над XML полягають у тому, що він дозволяє використовувати складні структури атрибутів, займає менше місця і інтерпретується безпосередньо в об'єкти мовою JavaScript..

Сервер Apache

Веб-сервер Apache - це незалежний, некомерційний, вільно розповсюджуваний продукт. Продукт підтримує широкий спектр функцій, багато з яких реалізовано у вигляді скомпільованих модулів, що розширюють основну функціональність. Вони варіюються від підтримки мов програмування на стороні сервера до схем автентифікації, з інтерфейсами, що підтримують Perl, Python, Tcl і PHP.

Серед поширених методів стиснення, які використовує Apache, є зовнішній модуль mod_gzip, призначений для зменшення розміру веб-сторінок, що надсилаються через HTTP.

Віртуальний хостинг дозволяє використовувати одну інсталяцію Apache для обслуговування різних веб-сайтів. Наприклад, на одній машині з однією інсталяцією Apache можуть одночасно розміщуватися www.example.com, www.test.com, test47.test-server.test.com і т.д.

Apache в основному використовується для обслуговування статичних і динамічних веб-сторінок у всесвітній павутині через HTTP. Багато веб-

додатків розроблено з урахуванням середовища і функціональності, що надаються цим веб-сервером.

HTML5

HTML5 - це наступна версія мови HTML. Специфікація HTML5 виходить за рамки розмітки і включає в себе набір веб-технологій, які формують відкриту веб-платформу. Це програмне середовище для запуску крос-платформних додатків з підтримкою відео, графіки та анімації, які можуть взаємодіяти з апаратним забезпеченням і надавати розширені мережеві можливості.

Загалом з нової версії мови розмітки пропонується вилучити близько 15 тегів.

При прийнятті рішення про додавання нових тегів було взято до уваги більшість популярних веб-сайтів і визначено основні елементи, спільні для всіх веб-сторінок.

Ця технологія допомагає полегшити навігацію користувачам, позначаючи області на сторінці певними елементами. Наприклад, вони можуть легко пропускати розділи навігації або швидко переходити від однієї статті до іншої. Автори також отримують вигоду від досягнення чистого, легкого для читання вихідного коду, замінюючи велику кількість divs невеликою кількістю відповідних елементів.

Відео- та аудіоелементи полегшують це завдання: Більшість API є спільною для цих елементів, єдині відмінності стосуються візуальних і невізуальних медіа. Всі сучасні браузери, окрім Internet Explorer, вже реалізують підтримку цих елементів. Найпростіший спосіб вбудувати відео - використати тег video і змусити браузер відображати інтерфейс за замовчуванням. Булевий атрибут controls визначає, чи буде цей інтерфейс включений за замовчуванням.

Необов'язковий атрибут poster можна використовувати, щоб вказати зображення, яке буде показано перед початком відтворення відео. Хоча деякі відеоформати (наприклад, Mpeg-4) підтримують власні прев'ю, цей метод є незалежним від формату відео.

Також легко прив'язати аудіо за допомогою елемента Audio: тег audio не має атрибутів height, width і poster зі зрозумілих причин, але більшість атрибутів є спільними для відео та аудіо.

У HTML5 є елемент source для вказівки альтернативного відео- або аудіофайлу, щоб браузер міг вибрати той, який відповідає підтримуваним типам медіа та кодекам. атрибут media визначає вибір медіа-запиту на основі обмежень пристрою, атрибут audio визначає тип медіа та кодек, атрибут type визначає потужність типу медіа або кодека. при використанні атрибуту source в елементі video (audio) слід опустити src.

Canvas

Полотно - це елемент HTML5, який можна використовувати для малювання графіки за допомогою скриптів (особливо JavaScript). Наприклад, його можна використовувати для малювання графіки, створення фотографій або анімації.

Полотно створює поверхню для малювання, яка надає один або декілька контекстів рендерингу, що використовуються для створення та модифікації відображуваного вмісту. У цій главі розглядається 2D (двовимірний) контекст візуалізації, який є єдиним визначеним на даний момент контекстом. Наприклад, може бути додано тривимірний контекст на основі OpenGL ES.

CSS

Каскадні таблиці стилів (CSS) - це спеціальна мова, яка використовується для опису сторінок, написаних мовою розмітки даних.

CSS використовується авторами та відвідувачами веб-сторінок для визначення зовнішнього вигляду сторінки, таких як кольори, шрифти та макет. Однією з його основних переваг є можливість відокремити вміст сторінки (або контент, зазвичай HTML, XML або подібну мову розмітки) від зовнішнього вигляду документа.

Таке відокремлення може покращити читабельність і доступність контенту, забезпечити більшу гнучкість і контроль над тим, як контент відображається в різних середовищах, зробити контент більш структурованим і простим, а також усунути повторення. CSS також можна використовувати для адаптації контенту до різних умов перегляду..

Текстовий редактор Sublime Text

У якості середовища розробки був обраний багатофункціональний текстовий редактор SublimeText (рис. 3.5).

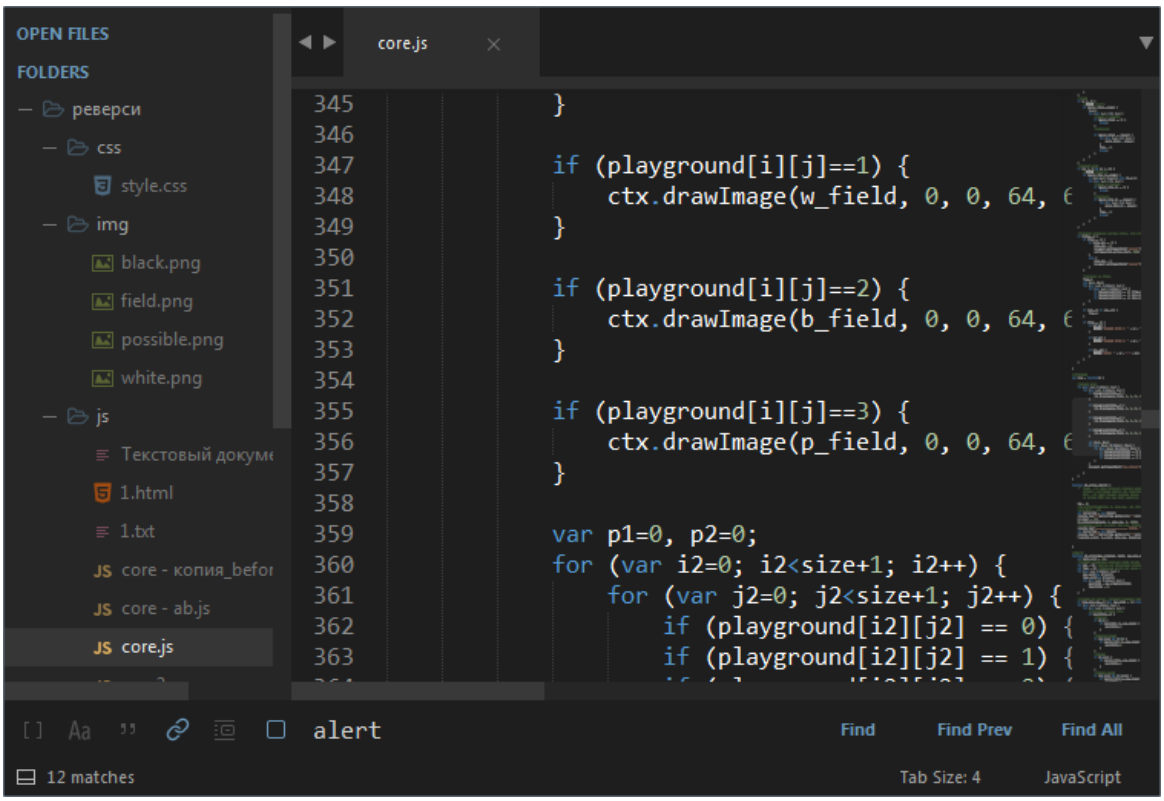

Рис. 3.5 Інтерфейс текстового редактору SublimeText

Sublime Text - це швидкий, крос-платформний редактор вихідного коду, який підтримує плагіни, розроблені за допомогою мови програмування Python.

Sublime Text не є вільним або відкритим програмним забезпеченням, але деякі плагіни поширюються під вільною ліцензією і розробляються та підтримуються спільнотою розробників.

За замовчуванням Sublime Text працює з окремими файлами, але він також може працювати з проектами, і цей режим роботи має ряд важливих переваг: відображається деревоподібна файлова структура відкритих папок проекту і відкритих файлів, файли і папки можна створювати, перейменовувати, видаляти, а також здійснювати розширений пошук у файлах проекту.

Висновки до розділу 3

1. Розроблено алгоритм самонавчання. Він базується на взаємодії серверної та клієнтської частин. Зв’язок реалізовується за допомогою WebSocket. Сервер звіряє прийняте рішення программи, використовуючи нейронну мережу.

2. Вибрано засоби реалізації розроблених алгоритмів самонавчання, оцінки позиції та генерації ходів.

РОЗДІЛ 4 РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ ПРОГРАМИ

4.1. Розробка інтерфейсу

Серверна частина інтерфейсу

Серверна частина, в свою чергу, має консольний інтерфейс, використовуючи програмну платформу Node.js. Вона також є кросплатфоменною, тому теж зберігає вище зазначені переваги.

Сервер містить змогу переглядати статистику по кожній зіграній позиції, за допомогою браузера. Статистика та візуальне відображення ігрової позиції реалізовано завдяки звичайним таблицям, комірки яких приймають колір в залежності від значення клітини на дошці. Також враховується крок, який має бути здійсненний. Колір комірок задається за допомогою спеціальної мови CSS:

```
#b {background: #000;}
#w {background: #FFF;}
#e {background: #1B8F3E;}
#s {background: #FD1;}
```

Вивід дошки реалізований мовою JavaScript:

```
     var htmlcode="";
     for (var j=0; j<64; j++)
{
          if (i.charAt(j)=="1") {htmlcode+="<div
id='w'></div>";}
          if (i.charAt(j)=="2") {htmlcode+="<div
id='b'></div>";}
          if (i.charAt(j)=="3") {htmlcode+="<div
id='s'></div>";}
```

```
                if (i.charAt(j)=="0") {htmlcode+="<div
id='e'></div>";}
      }
```

Де htmlcode – дані для відображенні у вигляді HTML.

Клієнтська частина інтерфейсу

Під час розробки клієнтської частини інтерфейсу, було прийнято рішення у використанні веб-інтерфейсу. Це зумовлено тим, що додатки, які розроблені за допомогою веб-технологій, наділяються властивістю кросплатформенності. Завдяки цьому, у більшості користувачів не буде виникати ніяких проблем у тестуванні даного програмного засобу.

Перше питання яке виникає при розробці продукту це засіб, яким буде реалізований ігровий інтерфейс. Було обрано використовувати графічний об’єкт canvas з функціоналу html5.

Ініціалізація графічного полотна поділяється на два етапи.

Перший, це додавання наступного фрагменту коду у тіло html-файлу, який визначає розмір графічного полотна:

```
<canvas width='512' height='512' id='canvas1'></canvas>
```

Другий, це визначення об’єкту, для подальшого звернення у коді скриптової мови JavaScript, де його ім’ям виступає назва ідентифікатору вказаної у фрагменті html-розмітки.

Таким чином ми зв’язуємо два вище вказаних параметра:

```
canvas1 = document.getElementById("canvas1");
ctx = canvas1.getContext('2d');    // графіка у 2D режимі
ctx.clearRect(0, 0, 512, 512);     // очищення екрану
```

Наступним питання, яке виникає при розробці, це визначення формату виведення даних. А тому знаючи, що наш додаток має вигляд дошки 8 на 8 і містить усі дані в масиві цього самого розміру, можна зробити висновки й прийняти за розмір клітини 64 квадратних пікселів.

Усього для відображення положення на дошці гравців, та самої дошки, знадобиться лише три графічних файли:

```
var e_field = new Image();
e_field.src = 'img/field.png'; //пуста клітина
var w_field = new Image();
w_field.src = 'img/white.png'; //біла фішка
var b_field = new Image();
b_field.src = 'img/black.png'; //чорна фішка
```

Для їх подальшого відображення було створено функції draw(), яка перебирає масив значень та малює їх на графічному полотні canvas, де змінна size відповідає за розмір дошки:

```
var draw = function() {
for (var j=0; j<size; j++)
for (var i=0; i<size; i++) {
     if (playground[i][j]==0)
     {    ctx.drawImage(e_field, 0, 0, 64, 64, i*64, j*64, 64,
64); }

     if (playground[i][j]==1)
     {    ctx.drawImage(w_field, 0, 0, 64, 64, i*64, j*64, 64,
64); }

     if (playground[i][j]==2)
     {    ctx.drawImage(b_field, 0, 0, 64, 64, i*64, j*64, 64,
64); }
}
```

Останньою складовою розробки інтерфейсу є реалізація взаємодії користувача з програмним продуктом. У нашому прикладі це здійснюється за допомогою використання маніпулятора «мишка». Щоб отримати значення координат курсору, знадобилось використовувати спеціальні методи:

1. OnMouseMove, який відповідає за координати курсору, під час переміщення відносно дошки:

```
canvas1.onmousemove = function(e) {
     mouseX = e.offsetX;
     mouseY = e.offsetY;
}
```

2. OnMouseDown, який при натисканні на праву кнопки миші, визначає яка клітина на дошці малась на увазі, та здійснює хід:

```
canvas1.onmousedown = function(e) {
     if(e.button == 0) {
     clickedX = (mouseX - mouseX % 64) / 64;
     clickedY = (mouseY - mouseY % 64) / 64;
     if (playground[clickedX][clickedY]==0)
     f_move(clickedX, clickedY, play_now, playground, 1);
     }
}
```

Таким чином, взаємодія користувача з веб-додатком є досить простою і не вимагає детальних пояснень.

4.2. Інструкція користувача

Для запуску гри слід у кореневому каталозі проекту запустити файл start.bat. Це діє лише у разі використання операційної системи Windows. У іншому випадку, користувачу потрібно встановити програмну платформу

Node.js, після чого, за допомогою консолі, виконати команду «nodejs server.js».

Далі користувач має дочекатись запуску серверної частини та підключення бази даних (рис. 4.1).

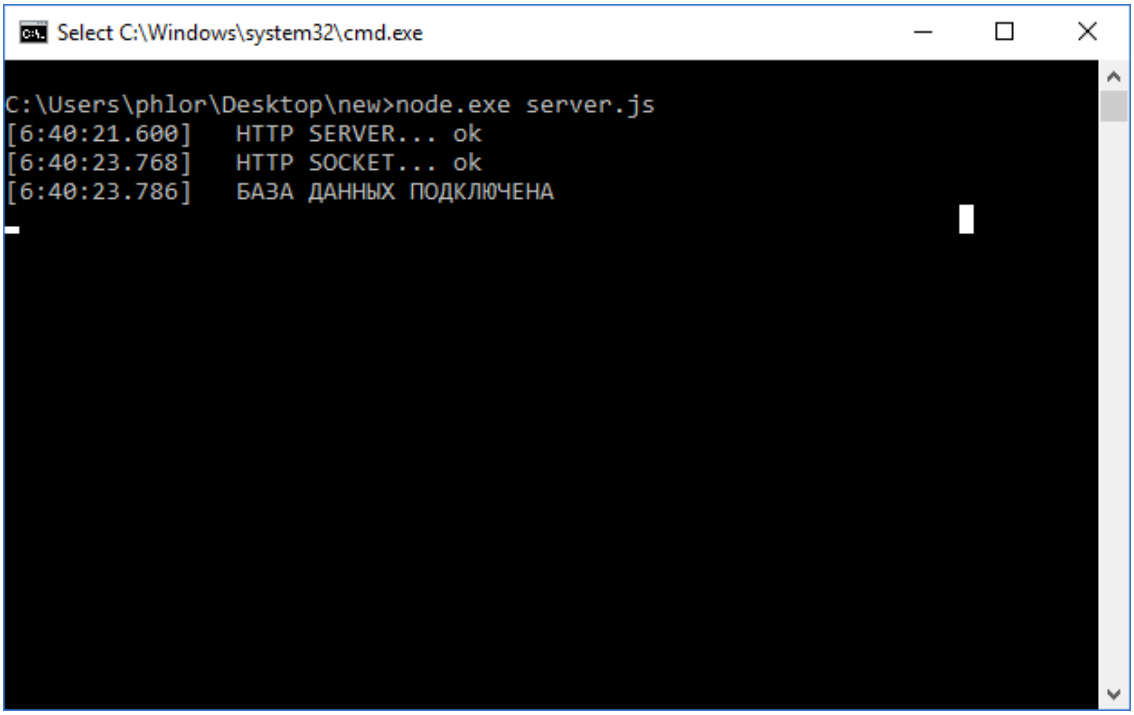

Рис. 4.1 Завантаження сервера

У разі успішного підключення бази даних, користувач повинен запустити браузер та перейти за локальною адресою http://localhost або http://127.0.0.1.

Перед початком гри проти комп'ютера, гравець має ознайомитись з правилами гри «Реверсі».

Чорні зупиняються на d5 та e4, білі - на d4 та e5. Чорні роблять перший хід. Потім вони ходять по черзі.

Роблячи хід, гравець повинен поставити свою фігуру на одну з клітинок дошки, а між цією фігурою та фігурами свого кольору на дошці розмістити неперервний ряд фігур суперника по горизонталі, вертикалі або

діагоналі. Усі фігури суперника, які потрапляють у цей "закритий" ряд, змінюють колір і належать гравцеві, який зробив цей хід.

Якщо в результаті одного ходу одночасно "закриваються" два або більше рядів фігур суперника, то всі фігури в усіх "закритих" рядах перевертаються.

Гравець має право вибрати одну з можливих комбінацій. Якщо у гравця є допустима рука, вона не може бути відкинута. Якщо немає жодної можливої комбінації, роздача переходить до суперника.

Гра закінчується, коли всі фішки розміщені на дошці, або коли жоден гравець не може зробити хід, або коли у одного з суперників не залишилося фішок. Наприкінці гри підраховуються фішки кожного кольору, і перемагає гравець, який має найбільшу кількість фішок на дошці. Якщо у гравців однакова кількість фішок, гра закінчується внічию..

Ознайомившись з правилами, користувач має здійснити перехід на веб-сторінку, де і розпочинає свою гру.

Інтерфейс є досить інтуїтивно зрозумілим (рис. 4.2).

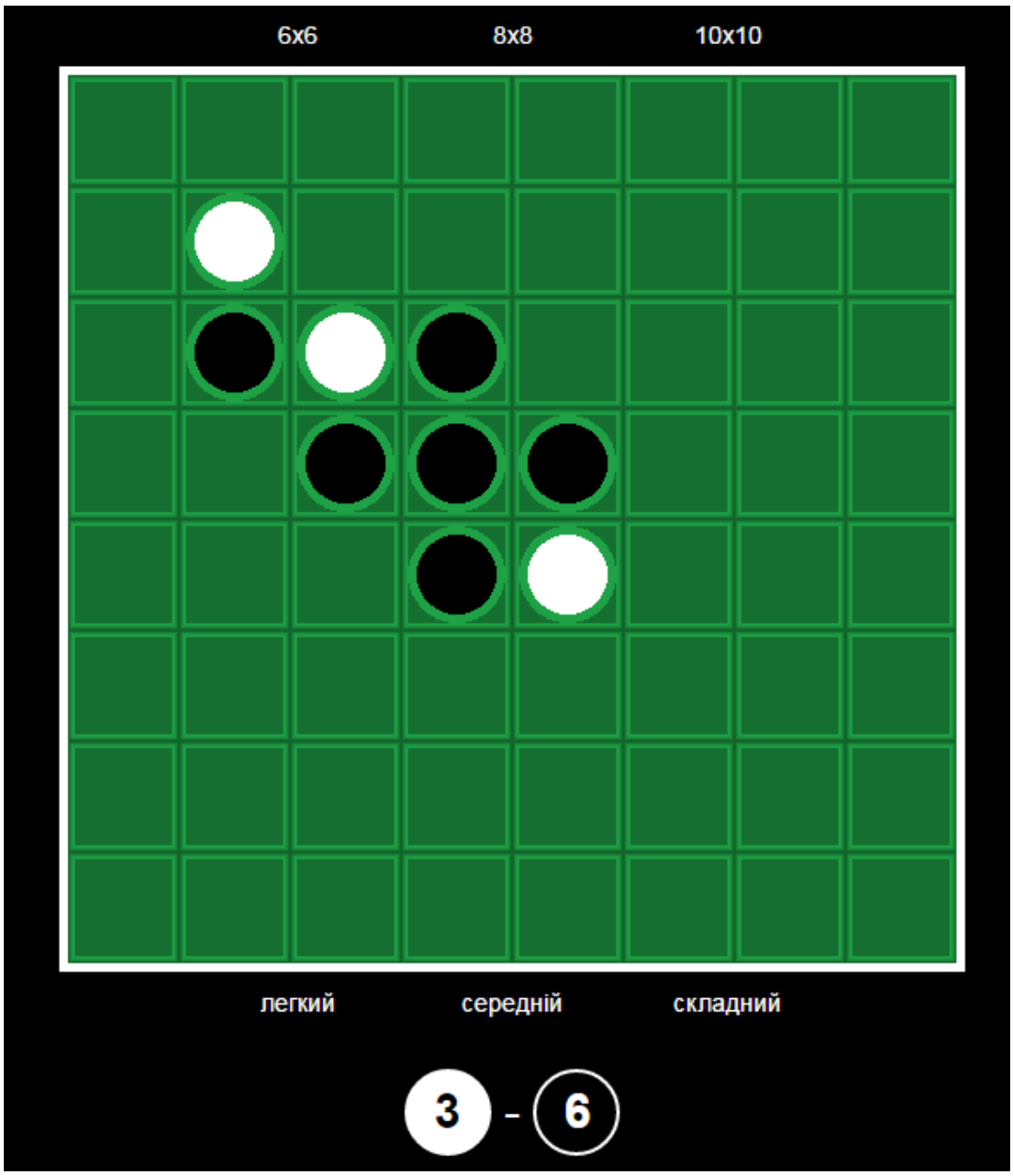

Рис. 4.2 Інтерфейс гри «Реверсі»

Насамперед користувачу надається змога обрати рівень інтелекту супротивника: легкий, середній та важкий. Обравши один з цих варіантів гравець повинен здійснити хід.

Аналіз прогресу гри здійснюється за допомогою панелі в нижній частині екрану, яка інформує користувача стосовно кількості своїх фішок та фішок супротивника.

Закінчити гру можна ще до того, як буде заповнене усе ігрове поле.

Коли гра досягає логічного кінця, користувачу надається інформація про гравця, який виграв дану партію (рис 4.3). Після натиснення на кнопку «Ок», фішки на дошці знову приймають стартове положення.

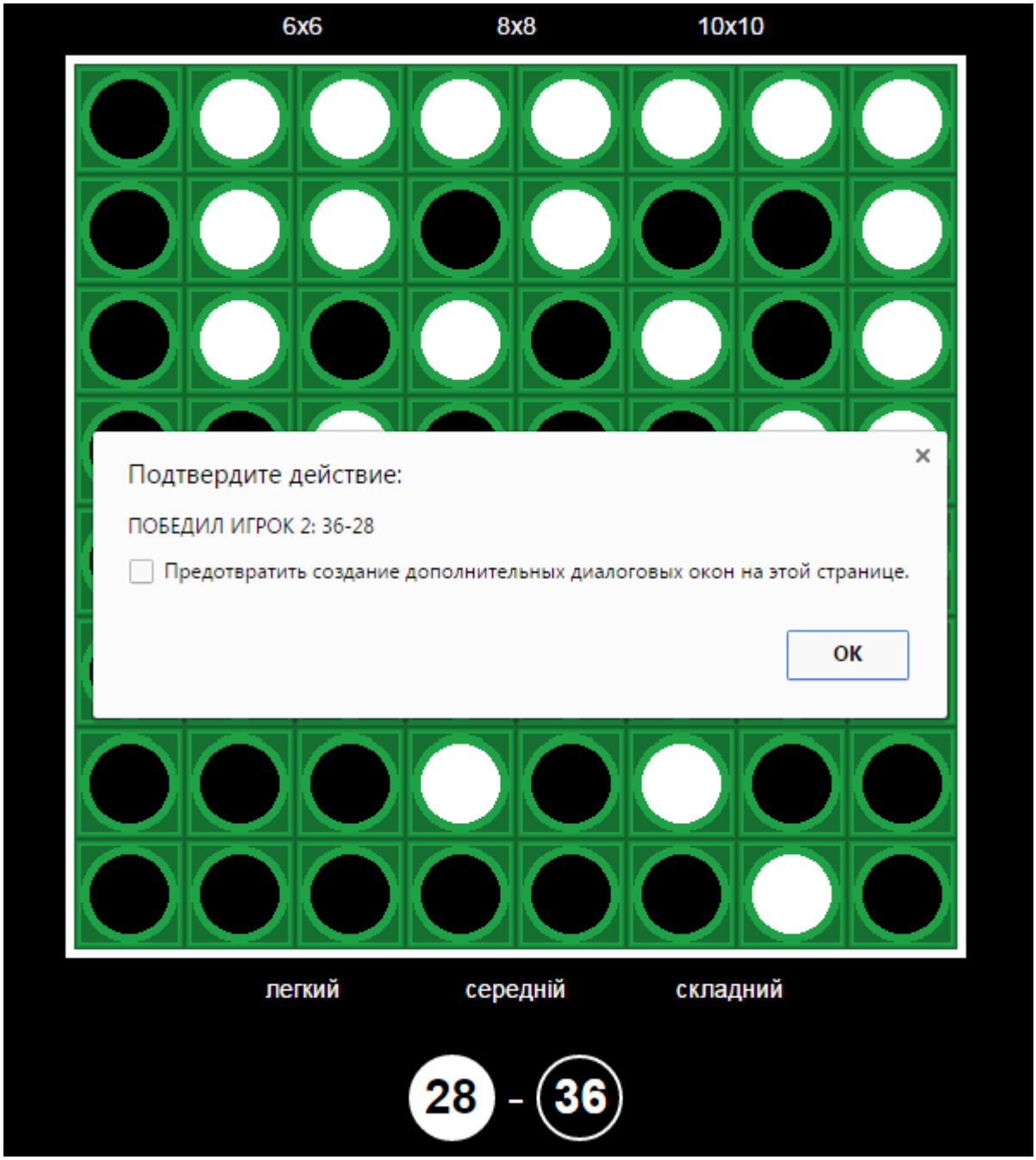

Рис. 4.3 Демонстрації завершення ігрового процесу

Також у програму була додана можливість обрати один з трьох видів розміру ігрового поля (рис. 4.4).

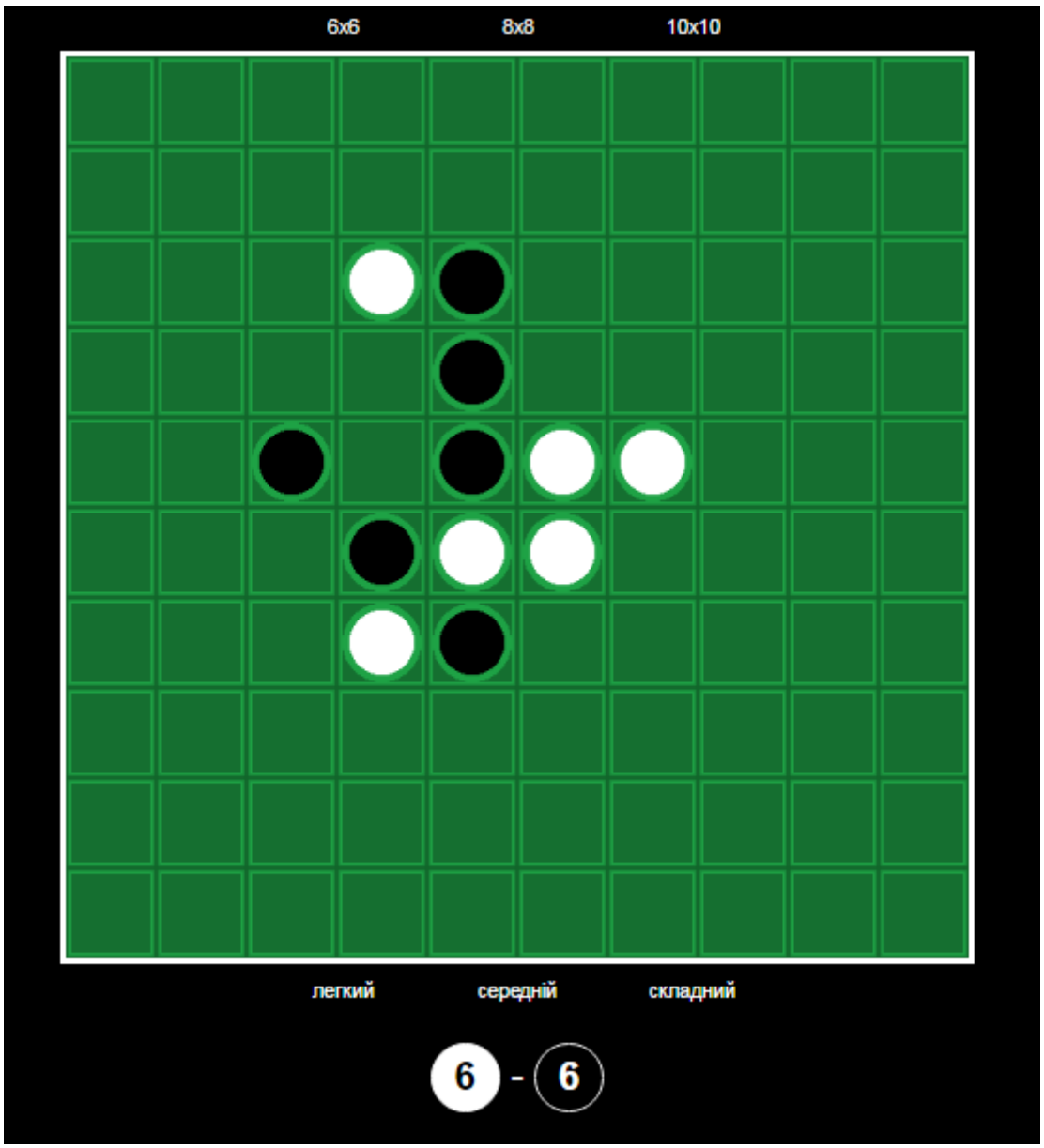

Рис. 4.4 Дошка розмірністю 10 на 10

База даних навчання програми та статистика усіх зіграних партій зберігається у файлах «db.json» та «w.json», тому якщо потрібно перенести її на інший сервер – достатньо лише замінити ці два файли.

Висновки до розділу 4

1. Було розроблено інтерфейс програмного продукту. Він базується на подійно-орієнтованому програмуванні. Тобто зв'язок користувача з функціоналом програми відбувається через обробку подій (рух та натискання кнопок маніпулятора миші).

2. Розроблене програмне забезпечення є веб-додатком. Користувач має можливість вибрати рівень складності та розмір ігрового поля. Для виконання ходу необхідно натиснути ліву кнопку маніпулятора миші.

Дане програмне забезпечення було протестовано на кросплаформенність.

ВИСНОВКИ

В результаті виконаної роботи досягнуто поставленої мети. Розроблено алгоритми оцінки позиції, генерації ходів та самонавчання для гри реверсі.

В результаті виконаної роботи можна зробити такі висновки:

1. Виконано аналіз існуючих методів машинного навчання. Для даної задачі, було обрано дельта-метод, за допомогою якого здійснюється навчання нейронної мережі.

2. Розроблено алгоритм самонавчання для гри "Реверсі. Він базується на нейронній мережі, яка враховує статистичні данні у кожні позиції.

3. Створено програмний продукт, який реалізовує розроблені алгоритми. Програма реалізована на об'єктно-орієнтованій мові JavaScript. Серверна частина розроблена за допомогою програмної платформи Node.js. Зв’язок між клієнтом та сервером здійснюється завдяки протоколу WebSocket. Клієнтська частина створена у якості веб-додатку. Графічна складова базується на використанні елементу HTML5 – canvas, який забезпечує відображення 2D графіки. Взаємодія користувача з програмою здійснюється лише за допомогою використання маніпулятора миші.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Ясницкий Л. Н. Введение в искусственный интеллект. М.: Издательский центр «Академия», 2005. — 176с.
2. Корнилов Е. Н. Программирование шахмат и других логических игр / Е. Н. Корнилов. – СПб.: БХВ-Петербург, 2005. – 272 с.
3. Рассел С. Искусственный интеллект. Современный подход / С. Рассел, П. Норвиг. – М.: Вильямс, 2007. – 1410 с.
4. Гудман Д. JavaScript. Библия пользователя. / Д. Гудман, М. Моррисон. – М.: Вильямс, 2006. – 1175 с.
5. Демьянов В. Ф. Введение в минимакс / В. Ф. Демьянов, В. Н. Малоземов. – М.: Наука, 1972. – 368 с.
6. Лахатин А. С. Языки программирования. Учеб. пособие / А.С. Лахатин, Л.Ю. Искакова. – Екатеринбург, 2013. – 548 с.
7. Петюшкин А. В. HTML. Экспресс-курс. / А. В. Петюшкин. – СПб.: БХВ-Петербург, 2004. – 258 с.
8. Кингсли-Хью Э. JavaScript 1.5. Учебный курс. / Э. Кингсли-Хью, К. Кингсли-Хью. – СПб.: Питер, 2001. – 272 с.
9. Гончаров А. Г65 Самоучитель HTML / А. Гончаров. – СПб.: Питер, 2002. – 240 с.
10. Хеник Б. HTML и CSS. Путь к совершенству. / Б. Хеник. – СПб.: Питер, 2011. – 336 с.
11. Хольцнер С. HTML5 за 10 минут / С. Хольцнер. – М.: Вильямс, 2011. – 237 с.
12. Флэнаган Д. JavaScript. Подробное руководство. / Д. Флэнаган. – Символ-Плюс, 2012. – 1081 с.
13. Стефанов С. JavaScript. Шаблоны. / С. Стефанов. – Символ-Плюс, 2015. – 272 с.

14. Мейер Э. CSS. Каскадные таблицы стилей. Подробное руководство. / Э. Мейер. – Символ-Плюс, 2008. – 576 с.
15. Кормен Т. Алгоритмы. Построение и анализ. / Т. Кормен, Ч. Лейзерсон, Р. Ривест, К. Штайн. – М.: Вильямс, 2012. – 1296 с.
16. Макконнелл С. Совершенный код / С. Макконнелл – Питер, 2007. – 886 с.