ЗМІСТ

СПИСОК ТЕРМІНІВ, СКОРОЧЕНЬ ТА ПОЗНАЧЕНЬ

SGF (Simple Game Format) – формат файлів для збереження партій настільних ігор.

GTP (Go Text Protocol) – протокол, який використовують більшість програм, що грають в Го для комунікації між собою.

MCTS (Monte-Carlo Tree Search) – метод Монте-Карло для пошуку в дереві. Евристичний алгоритм для пошуку та прийняття рішень. Його часто використовують програмах, які грають в настільні ігри

UCT (Upper Confidence bounds applied to Trees) – метод верхньої оцінки значущості для дерев. Один з методів Монте-Карло пошуку в дереві.

ВСТУП

Го [1] – стародавня китайська стратегічна гра на двох гравців. Вона є антагоністичною [2] грою з повною інформацією. В Го грають два гравці – “Чорні” та “Білі”. Вони по черзі розміщують на дошці що складається з перетину 19 на 19 ліній камені свого кольору. Го територіальна гра, тобто гравець, що під кінець гри має більшу територію виграє. Як і для багатьох інших подібних ігор, були спроби створити комп'ютерні програми, що гарно грають в Го, але це виявилося справжнім викликом для програмістів. Складність обчислення партій в Го на кілька порядків більша за шахи. На кожному кроці можливі близько 200-300 ходів, статична ж оцінка життя груп каменів фактично неможлива. Одним ходом тут можна цілком зіпсувати всю гру, навіть коли решта ходів були дуже добрі. Тому програми для гри в Го не використовують таких алгоритмів, як шахові програми, а замість цього зазвичай мають кілька десятків модулів для оцінки різних аспектів гри і під час аналізу намагаються використовувати ті ж самі поняття, що й люди. Попри це вони і далі грають дуже слабко та програють навіть не дуже сильним аматорам.

Усі програми, що грають в Го повинні оперувати деякими структурами даних (наприклад такими, що репрезентують поточний стан дошки або гри). Більшість з таких програм засновані на створенні великої кількості різних партій, та подальшому їх аналізі. Саме тому питання пошуку у подібних структурах даних дуже актуальне, адже його оптимізація корінним чином вплине на роботу та якість програм, що грають в Го.

1. АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ СИСТЕМ ПОШУКУ У СТРУКТУРАХ ДАНИХ ГРИ ГО

1.1. Огляд існуючих програмних продуктів для пошуку

Кожна програма, що грає в Го повинна вміти ефективно шукати ходи/партії у своїй внутрішній базі даних. Однак партії професійних гравців у Го мають важливе навчальне значення самі по собі. Тому існують програми, що єдиною своєю метою ставлять роботу з партіями Го. Вони об'єднують багато партій в одну базу даних і маніпулюють цією базою.

1.1.1. Kombilo

Kombilo [3] – програма, що працює з базою партій гри Го. Основне завдання такої програми – пошук деяких підпослідовностей в колекції SGF-партій (наприклад пошук усіх партій з деяким початком). Також ця програма дозволяє шукати по деяким властивостям партій (таким, як гравці, події, дати).

Особливості програми:

- Можливість пошуку як по повному ігровому полю, так і по позиціями у куті або на стороні дошки. Пошук ведеться також враховуючи симетрію та поворот. Існує можливість інвертувати пошук по кольору гравців, тобто поміняти їх місцями.
- Kombilo також комплектується повним редактором SGF: тобто існує можливість редагувати файли, коментувати їх та інше. Також редактор дозволяє бачити дерево варіантів партії. Він дозволяє повертати/віддзеркалювати партії.
- Kombilo також має у собі механізм по відображенню посилань. Цей механізм додає підказки до ігор, що впізнає. На поточний момент база цих посилань налічує 2000 записів.

- Можливо використовувати складні запити, використовуючи напряму SQL-базу, що зберігає розібрані партії.
- Можливо застосувати будь-яку комбінацію пошукових запитів, та у будь-який момент отримати список партій, що відповідають заданому пошуковому запиту.

Говорячи про реалізацію програми, можна відзначити що вона написана на Python, тому може вважатися багатоплатформною. Основне ж ядро, що присвячене пошуку, також написане на C++. Компілюючи його у модуль та підключаючи до основної програми ми отримаємо збільшення швидкодіє більше, ніж у два рази. Також це відкрита програма, і програмний код доступний для читання та аналізу.

1.1.2. Master Go

Це комерційна програма, що працює з базами даних гри го і призначена для пошуку поширених початків та розвитків ігор (вивченні фусекі і джосекі). База даних містить 53059 професійних ігор у власному форматі. Оновлення доступні для всіх зареєстрованих користувачів. Можливо додавати ігри в базу, що постачається з Master Go [4] шляхом придбання цих ігор в Японії, Китаї і Кореї і копіюванням їх в базу. Також Master Go працює тільки на операційній системі MS Windows.

Якщо порівнювати цю програму з Kombilo, та можна помітити декілька речей:

1. Master Go, комерційна та платна програма.
2. Вона використовує внутрішній формат даних, що також є не дуже зручним.

1.1.3. BiGo Assistant

BiGo Assistant [5] – програма для роботи з базами фусекі та джосекі, створена для гравців, що хочуть підняти свій рівень гри в Го.

Основні особливості:

- Можливість не тільки вивчати ігри го в цілому, але й переглядати ігри професіоналів у декількох режимах.
- Можливість вивчати початковий розвиток партії в го (фусекі) за прикладом бази даних професійних ігор.
- Також присутня база даних джосекі.
- Можливість аналізувати власні початки різних партій.
- Динамічний інтерфейс – можливість створення будь-якої кількості вікон з різними партіями.
- Можливість роздруковувати партії.
- Можливість застосувати до дошки всі 8 можливих перетворень (симетрія та поворот).
- Програма відслідковує дублікати ігор.
- Можливість збирати статистику по списку ігор у вигляді статистики по варіаціям можливих рухів або оціночну статистику позицій у грі обох гравців.
- Пошук по заданій позиції (по одній або декільком частинам дошки).

Програма має багато можливостей, однак через те, що цікавим представляється тільки метод пошуку, ця програма має одну важливу відмінність від усіх інших: вона дозволяє шукати по декільком частинам дошки.

1.2. Огляд існуючих алгоритмів пошуку у структурах даних гри Го

Єдиний вибір, що повинна зробити програма, що грає в Го, це куди поставити наступний камінь. Однак, цей вибір ускладняється тим, що навіть один камінь може дуже сильно впливати на ситуацію на дошці в цілому. У той же час не можна забувати про об'єднання каменів – їх групи,

та про взаємодію цих груп між собою. Для вирішення цієї проблеми використовуються різні підходи. Розглянемо деякі з них.

1.2.1. Мінімаксний пошук по дереву варіантів

Мінімаксний пошук [6] може використовуватися для того, щоб моделювати велику кількість різних партій. Загалом, алгоритм простий: спочатку алгоритм по черзі грає усі варіанти ходів до деякого моменту. Потім використовується функція оцінки позицій, для вираховування того, наскільки поточна позиція гарна для кожного гравця. Далі, на основі цього зваженого дерева ходів, робиться розрахунок оптимальної стратегії для одного з гравців – вибираються ті ходи, що дають найбільше переваги цьому гравцю.

Хоча цей метод був досить ефективним для шахів, він не дуже підходить для Го. По-перше, через те, що досі не було створено відповідної функції оцінки для позиції в Го. Гра Го все ще не формалізована математично, тому поточні функції оцінки партії запрограмовані робити висновки як люди. Тобто програмісти намагалися навчити свої програми грати, як вони самі. По-друге, через те, що для партії в Го притаманний великий фактор розгалуження. У кожний момент гри для кожного з гравців існує дуже багато коректних ходів. Також самі партії в Го довші, ніж у шахах. Саме тому такі методи дуже обчислювально коштовні. На даний момент програми, що використовують подібні алгоритми можуть грати тільки на дошках менших ніж 9x9.

1.2.2. Порівняння з шаблоном

Аналіз результатів ігор між різними комп'ютерними програми, що використовують різні методи, дозволило дійти висновку, що кращий спосіб гри в Го – поєднання методів порівняння з шаблоном із методами швидкого локалізованого тактичного пошуку.

Методи, що використовують порівняння з шаблоном маніпулюють послідовністю ходів, що є прийнятною для обох гравців. Це відомі маленькі шматочки з яких найчастіше складаються локальні ситуації. Такі послідовності добре вивчені і обґрунтовані, тому достатньо їх правильно використовувати всередині гри. Пошук цих зразків є дуже важливим як для гравців-людей, так я для програм, що грають в Го. Розглянемо один із можливих алгоритмів пошуку таких зразків в грі Го.

Виберемо деяку зону на дошці, яку будемо вважати зразком. Наприклад квадрат 5x5. Потім потрібно обчислити хеш цього квадрату, тобто представити у вигляді int64 числа. Однак перед цим потрібно привести його до деякого базового вигляду. Адже навіть однакові зразки з точністю до повороту або симетрії будуть виглядати різними на дошці, бо не будуть співпадати в усіх клітинках. Всього вісім різних позицій будуть однаковими. Чотири повороти (на 0, 90, 180, 270 градусів відповідно) і 4 повороти віддзеркаленого зразка. Нехай базовий вигляд буде вигляд, у якого хеш найменше число. Тоді порахувавши 8 хешів і обравши менший, ми зведемо всі подібні зразки до одного. Далі, якщо зберігати багато подібних зразків у базу, то ця база може бути використана для пошуку у ній часток поточної гри. З великою вірогідністю, декілька початкових каменів дадуть змогу знайти відповідний гарний шаблон, який і треба буде далі відіграти програмі.

1.2.3. Методи, засновані на ймовірності

Базуючись на попередньому методі, можна отримати базу зразків партій досвідчених гравців. Використовуючи цю базу, можна отримати розподіл ймовірностей ходів для професійних ігор, який можна використовувати для відтворення цих ходів у окремій програмі. Цей розподіл можна використовувати не тільки для програми, що грає в Го, але й в якості навчального посібника для гравців у Го.

Цей метод має дві основні складові:

- Схема вилучення шаблону з експертних партій гри (реалізовано у попередньому пункті).
- Байесовський алгоритм навчання, який навчається розподілу аналізуючи ходи у локальному місці дошки.

1.2.4. Методи, засновані на базі знань

Якщо використовувати все ті ж самі шаблони, що згенерував метод порівняння зі зразком, та додати до них інтелект програміста, то вийде досить сильна програма для гри в Го. Під інтелектом програміста, мається на увазі можливість вирішувати локальну позицію у шаблоні використовуючи деякий набор евристик. Програмісту достатньо тільки перевести ці правила в комп'ютерний код та використати пошук за зразком, щоб знаходити ситуації, де ці правила доречні. Основний недолік – складність цих правил, а точніше можливість програмування їх, залежить насамперед від здатності та навику гри в Го самого програміста. Зважаючи на те, що математичного апарату для подібної роботи нема, кожен програміст намагається навчити свою програму грати, як він. Тому найчастіше програми мають більше сотні модулів, що вираховують найкращий хід кожен окремо для своєї ситуації. Однак такі методи страждають від проблем, аналогічних попереднім – нерозуміння глобальної ситуації. Це призводить до того, що вони роблять помилки у стратегічному плані. Відомо, що можливо програти гру, якщо у вирішальний момент обрати неправильний хід.

1.2.5. Методи Монте-Карло

Однією з головних альтернатив використанню жорстко запрограмованих методів пошуку – використовувати методи Монте-Карло. Якщо говорити про Го, то цей метод полягає в наступному: згенеруємо список потенційних ходів, які ми хочемо перевірити; для кожного такого

ходу зіграємо тисячу випадкових партій (тобто наступних випадкових ходів до остаточного кінця гри – стану на дошці, при якому можливо точно визначити переможця); виберемо з списку ходів той, при якому найбільша кількість з випадкових партій виграна програмою.

Перевага цього методу полягає в тому, що він потребує дуже малих знань про предметну область, у якій працює, недоліком є те, що він використовує більше пам'яті та процесорного часу. Однак через те, що ходи генеруються випадковим чином, ми можемо неправильно оцінити якість ходу. Наприклад, якщо у відповідь на якийсь хід буде згенеровано 100 ходів-відповідей, у більшості з який перший гравець виграє, ми будемо оцінювати цей хід як дуже сильний. Однак існує можливість того, що є дуже добра відповідь на наш хід, яка зведе нашу перевагу нанівець, однак, через те, що ми не згенерували цей хід, ми про це не дізнаємося. В результаті цього, програма буде сильна в загальному стратегічному сенсі, однак буде дуже слаба тактично. Цю проблему можна вирішити, якщо додати деякі проблемно-орієнтовні знання в генерацію ходів та підвищити глибину пошуку.

У 2006 році був створений новий пошуковий алгоритм, що був використаний для гри на дошках розміру 9x9 та зарекомендував себе дуже гарно. Він називається Upper Confidence Bounds algorithm і базується на методі Монте-Карло. Алгоритм UCT [7] змінює правила за якими визначається важливість ходів у дереві пошуку. Він вибирає те піддерево, у якому вірогідність перемоги більше 50%. Якщо ж не існує такого піддерева, то вибір робиться навмання.

2. ВИБІР ЗАСОБІВ РЕАЛІЗАЦІЇ

Темою даної дипломної роботи є розробка системи для паралельного пошуку у структурах даних гри Го. Реалізована бібліотека повинна дозволяти шукати у структурах даних гри Го декількома різними способами. Ця бібліотеку може бути використана програмами, що грають в Го, аналізують партії в Го, збирають статистику з партій або ж шукають деякі закономірності у партіях. Щоб ця бібліотека була корисною і використовувалася, важливо вибрати правильну платформу та мову програмування для неї.

Основними властивостями розроблювальної бібліотеки повинні бути:

1. Платформо незалежність – це зробить бібліотеку більш поширеною та зручною.
2. Можливість використання з багатьма популярними мовами програмування.
3. Простота реалізації – внутрішні структури даних змодельовані використовуючи вбудовані у мову структури даних.

2.2. Вибір платформи

Вибір платформи для програми – дуже важливе рішення, адже платформа одразу не тільки поставить обмеження на доступні мови програмування але і додасть свої плюси та мінуси, до розроблюваного програмного забезпечення.

Три основні варіанти реалізації програмних продуктів:

- Без використання платформи.
- На основі платформи Java.
- На основі платформи Microsoft .NET.

2.1.1. Без використання платформи

Найочевиднішим вибором буде не використовувати ніяку платформу. Однак програмування, використовуючи платформу, таку як Java Platform або Microsoft .NET, має свої значні переваги. Однак при програмування бібліотеки методів пошуку не обов'язково використовувати якусь платформу, тож розглянемо цей варіант.

Основними перевагами написання програми без використання будь-якої платформи є:

- відсутність прошарків між програмою та ОС (найчастіше);
- не прив'язаність ні до яких інструментів;
- можливість написати простий standalone-додаток.

Основними мовами, що розглядалися були:

- C та C++;
- Python;
- Common Lisp.

Мови C та C++ є не тільки мовами системного програмування. Їх можливо з успіхом використовувати для розробки різноманітних додатків. Основною перевагою цих мов є швидкодія отриманої програми. Недоліком є складність та витратність розробки програмного забезпечення. Також не дуже зручно програмувати багатозадачні системи якщо використовувати такі мови.

Python – широко використовувана, скриптова, інтерпретована, високорівнева мова програмування. Вона доступна для більшості платформ. Вона підтримує концепти функціонального програмування, зокрема у ній функції є об'єктами першого класу (тобто можуть передаватися як змінна та бути збереженими у змінну). Також Python має багато вбудованих типів, що гарно підходять для даної задачі. Серед мінусів слід зазначити, що ця мова інтерпретована. Існування

інтерпретатора накладає свої мінуси, серед яких зменшення швидкодії та залежність від додаткових програмних засобів.

Common Lisp – це діалект Lisp-у, що набув значного розповсюдження у програмному світі. Існує реалізації під більшість платформ. Мова високорівнева, підтримує багато парадигм програмування, компільована. Основні переваги для роботи з деревами ця мова має через те, що вона є Lisp-мовою, тобто вона створена для маніпулювання такими структурами даних, як списки. У програмуванні дерево найчастіше подається у вигляді списку списків, тому більшість алгоритмів для роботи з деревами гарно програмуються на Lisp-і. Серед мінусів слід зазначити невелику популярність (якщо порівнювати з іншими не Lisp-ами), та невелику кількість бібліотек.

Загалом, серед розглянутого, Common Lisp був би найкращим вибором, якщо якось подолати його мінуси.

2.1.2. На основі платформи Java

Віртуальна машина Java – набір комп'ютерних програм та структур даних, що використовують модель віртуальної машини для виконання інших комп'ютерних програм чи скриптів. JVM використовує байт-код Java, який генерується з вихідних кодів мови програмування Java. Віртуальну машину також застосовують для виконання коду, згенерованого з інших мов програмування. JVM доступна для всіх основних сучасних операційних систем, тому про програми, що скомпільовані у Java байткод теоретично можна сказати “Написано один раз, працює скрізь”.

Основними мовами для розглядання були:

- Groovy;
- Scala;
- Clojure.

Groovy – об'єктно-орієнтована динамічна мова програмування, що працює в середовищі JRE. Мова Groovy запозичила деякі корисні якості Ruby, Haskell і Python, але створена для роботи всередині віртуальної машини Java (JVM) і підтримує тісну інтеграцію з Java програмами.

Scala – мова програмування, що підтримує багато парадигм програмування та поєднує властивості об'єктно-орієнтованого та функціонального програмування.

Clojure – сучасний діалект мови програмування Lisp. Це мова загального призначення, що підтримує інтерактивну розробку, зорієнтовану на функціональне програмування, спрощує програмування декількох потоків у одній програмі та містить риси сучасних скриптових мов. Clojure працює на Java Virtual Machine і Common Language Runtime.

Підхід Clojure до паралельності характеризується концепцією тотожностей, що представляють серію незмінних станів протягом часу. Оскільки стани є незмінними значеннями, будь-яка кількість обробників може паралельно обробляти їх і конкуренція зводиться до питання керування змінами від одного стану до іншого. З цією метою, Clojure надає декілька типів змінюваних посилань, кожен з яких має добре визначену семантику переходу між станами.

2.1.3. На основі платформи Microsoft .NET

Microsoft .NET – платформа від фірми Microsoft для створення як звичайних програм, так і веб-застосунків. Багато в чому є продовженням ідей та принципів покладених в технологію Java Platform. Одною з ідей Microsoft .NET є сумісність служб написаних різними мовами. Хоча ця можливість рекламується як перевага Microsoft .NET, платформа Java має таку саму можливість.

Основні мови, що розглядалися:

- C#;
- C++/CLI;

- F#.

C# – об'єктно-орієнтована мова програмування з безпечною системою типізації для платформи Microsoft .NET. Розроблена Андерсом Гейлсбергом, Скотом Вілтамутом та Пітером Гольде під егідою Microsoft Research.

C++/CLI – прив'язка мови програмування C++ до платформи .NET фірми Microsoft. Ця мова підтримує стандарт C++ ISO. Також до його підтримки додається Об'єднана система типів (Unified Type System, UTS), що розглядається як частина Загальної мовної інфраструктури (Common Language Infrastructure, CLI). Вона підтримує і рівень програмного коду, і функціональну сумісність виконуваних файлів, скомпільованих із керованого C++. C++/CLI являє собою еволюцію C++.

F# – багатопарадигмова мова програмування розроблена в підрозділі Microsoft Research і призначена для виконання на платформі .NET. Вона поєднує в собі виразність функціональних мов, таких як OCaml і Haskell, з можливостями і об'єктною моделлю Microsoft .NET. Функціональна мова максимально адаптована до використання на платформі Microsoft .NET, відповідно, вона не заперечує і імперативного підходу.

2.1.4. Висновки

Написання бібліотеки без платформи хоч і має свої плюси, однак мінуси у вигляді втрати доступу до великої кількості розробників, що все використовують платформу, важать більше, ніж плюси. До того ж більшість розглянутих мов без платформи не дуже підходить до задачі. Виняток складає мова Common Lisp, що гарно підходить для програмування вибраних методів, однак її популярність ще менша.

Платформа Microsoft .NET сама по собі досить актуальна, але набір мов, що доступні для розробки на цій платформі, також не дуже підходить для розробки системи пошуку у структурах даних гри Го.

У свою чергу платформа Java демонструє багато позитивних сторін, що знадобляться для реалізації методів пошуку. Вона популярна, багатоплатформна, має значний набір мов програмування, найкраще з яких – Clojure – гарний вибір для розробки бібліотеки.

Розглянемо мови програмування платформи Java більш детально.

2.2. Вибір мови програмування

2.2.1. Groovy

Groovy [8] є більш високорівневою мовою програмування порівняно з Java, а отже розробка на ньому зазвичай відбувається швидше. Цьому сприяють перш за все динамічна природа мови, а по друге існуючі елементи функціонального програмування, зокрема замикання.

Функціональній спрямованості мови розробники надають один з найбільших пріоритетів. Нові можливості з'являються досить регулярно. Існує режим статичної компіляції для забезпечення підвищеної продуктивності для критичних до швидкості виконання ділянок коду.

2.2.2. Scala

На Scala [9] вплинуло багато мов. Однорідна об'єктна модель вперше з'явилася у Smalltalk і згодом у Ruby. Універсальність вкладеності присутня у Algol, Simula, Beta. Принцип однорідного доступу для виклику методу і звернення до поля походить з мови Eiffel. Підхід до функціонального програмування подібний до підходу родини мов ML, таких як OCaml і F#. Багато функцій вищого порядку у стандартній бібліотеці Scala також наявні у ML або Haskell. Неявні параметри у Scala аналогічні класам типів Haskell. Заснована на акторах бібліотека багатозадачності подібна до Erlang.

2.2.3. Clojure

Clojure [10] компільована мова, вона геренує байткод для JVM. Вона має значну інтеграцію з Java: відкомпільовані в байткод JVM, програми на Clojure можуть пакуватися та запускатися на JVM-серверах без додаткових ускладнень. Мова також надає макроси, які полегшують використання існуючих Java API. Всі структури даних Clojure реалізують стандартні інтерфейси Java, що робить простим запуск з Java коду, розробленого на Clojure. Вона має мультиметоди (аналог перевантажень функцій), що підтримують динамічний вибір метода за типами та значеннями довільного набору аргументів.

Найбільша відмінність Clojure – послідовності. Це не окремий тип колекцій, особливо враховуючи, що їм не обов'язково бути саме списками. Доступні також такі вбудовані структури даних, як вектор, хеш-словник, набір та інші. При спробі отримати з порожньої колекції послідовність її елементів (викликом `seq`) повертається `nil`. При спробі отримати з послідовності (на її останньому елементі) залишок (`rest`) буде повернуто іншу логічну послідовність. Це дозволяє послідовностям та протоколу послідовностей бути лінивими.

2.2.4. Висновки

Серед розглянутих мов найбільше до вирішення даної задачі підходить мова Clojure. Вона має прив'язку до Java, тому це гарний варіант для розроблювання бібліотеки. Також ця мова – діалект Lisp-у, тобто вміє добре працювати з списками та деревами. Не слід забувати про гарну підтримку паралельного програмування, яка знадобиться при розроблюванні даної програмної системи.

3. РОЗРОБКА СИСТЕМИ ПАРАЛЕЛЬНОГО ПОШУКУ У СТРУКТУРАХ ДАНИХ ГРИ ГО

Як було показано в першому розділі існує немало програм для пошуку у структурах даних гри Го, але практично всі вони є комерційними. Некомерційною є програма Kombilo, але вона має такі недоліки:

- Складність роботи з системою. Якщо хтось захоче додати підтримку Го у свою систему, йому доведеться розбиратися з великою кількістю коду на C++.
- Графічний інтерфейс на Python. Хоча Python багатоплатформна мова, але для графічного інтерфейсу краще використовувати щось більш стандарте, що не залежить від додаткових програмних модулів.

Інша некомерційна програма, для гри в Го – Fuego [11]. Вона також написана на C++, але якщо аналізувати структуру її модулів, то вона краще організована та простіша у використанні. Однак у цій системі бракує методів для пошуку у структурах даних гри Го.

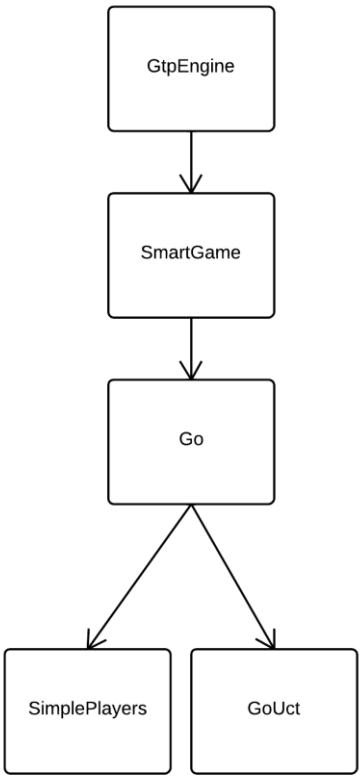

Рис. 3.1. Структура модулів Fuego

Розроблювальна система має схожу структуру модулів, вона зображена на кресленні “Схема взаємодії програмних модулів”. Подібна структура бібліотеки дає змогу модулям бути самостійними одиницями і використовуватися окремо.

Після огляду існуючих розробок можна зробити висновок про актуальність даної програмної розробки. Саме тому є доцільним розробити систему пошуку для гри Го, яка буде написана, використовуючи можливості платформи Java та мови Clojure. Така система повинна бути модульною та зручною для розробників у використанні. Це дасть змогу великій кількості людей використовувати цю систему, додавати до неї нові модулі та робити дослідження у області штучного інтелекту для гри в Го.

3.1. Загальний огляд системи

Розроблювана система пошуку у структурах даних гри Го складається з чотирьох модулів:

- модуль трансформації SGF-файлів у внутрішнє дерево;
- модуль методу Негамакс;
- модуль методу Монте-Карло пошуку в дереві;
- модуль методу пошуку шаблонів.

Перший модуль – модуль трансформації у внутрішнє дерево. Він отримує на вході SGF-файл та оброблює його використовуючи парсер, що перетворює цей файл на послідовність лексем та лексер, що перетворює послідовність лексем на внутрішнє дерево. У процесі розробки даного модуля була створена система функцій, що дозволяє ліниво читати файл блоками по 512 байт. Даний підхід гарно корелює з функціональною моделлю структур даних у мові Clojure, адже дозволяє зменшуючи обчислювальні витрати та витрати пам’яті, маніпулювати тим самим списком символів з файлу.

Функція `(lazy-read file len)` отримує шлях до файлу, що потрібно читати та розмір блоків для читання. Було знайдено, що значення 512 байт для параметру `len` дає гарні результати швидкості читання файлу.

```
(defn lazy-read [file len]
  (let [rdr (clojure.java.io/reader file)
        is-closed? (atom false)
        array (char-array len)]
    (fn next-char []
      (if @is-closed?
        nil
        (let [result (.read rdr array)]
          (if (= -1 result)
            (do (.close rdr)
                (swap! is-closed? complement)
                nil)
            array))))))
```

Лістинг 3.1. Функція lazy-read

Ця функція повертає замикання на функцію, що читає файл масивами по `len` байт. Але лексеру необхідно читати файл по одному символу інколи повертаючи прочитаний символ назад у файл. Використовуючи `java.io.reader` такої поведінки досягнути неможливо. Тому були створені допоміжні функції `(read-next-char lazy-read-fn stack)` та `(unread-char chr stack)`. Вони використовують функцію, отриману від `lazy-read` та стек, в який зберігають чергу символів для читання.

Другий модуль реалізує метод пошуку Негамакс – різновид методу Мінімакс. Цей метод – спрощення методу Мінімакс, що дозволяє використовувати одну й ту саму функцію для мінімізуючого та максимізуючого гравця. Цей підхід базується на зміні знаку змінної, що повертається з попереднього вузла дерева на протилежний.

Також цей метод потребує модуль правил гри Го, що складається з модуля генерування допустимих ходів та модуля оцінки поточної ситуації на дошці. Модуль генерування допустимих ходів повинен повертати список ходів, що не є забороненими в даній ситуації. Він повинен враховувати правила зняття з дошки каменів та правило Ко. Модуль оцінки

поточної ситуації на дошки найчастіше реалізовується у вигляді евристичної функції оцінки ситуації. Розробка цих модулів та функцій не входить в завдання на даний дипломний проект.

Третій модуль реалізує один із методів Монте-Карло пошуку в дереві, а саме – метод верхньої оцінки значущості для дерев. Цей модуль також потребує модуль правил гри Го. Він модуль генерує велику кількість випадкових ігор на основі початкової позиції та використовує отримане дерево партій для пошуку найкращого варіанту ходу.

Четвертий модуль призначений для роботи з шаблонами в іграх Го. Він надає можливість шукати шаблони в партіях з Го використовуючи заздалегідь згенеровану базу шаблонів.

3.2. Трансформація у внутрішнє дерево

Smart Game Format (SGF) – комп'ютерний формат даних, що зберігає партії ігор, таких як Го, шахи, шашки та реверсі. Це простий текстовий формат, що зберігає партії у вигляді дерев.

Були розроблені функції, що дозволяють маніпулювати файлами у цьому форматі: читати його, перетворювати у внутрішній деревовидний формат, спрощувати дерево, перетворювати у формат, що відображає положення дошки та візуалізовувати отриману дошку.

Читання та перетворення SGF-файлу запрограмовано у вигляді лексеру та парсеру. Лексер приймає на вході послідовність символів з файлу, що читаються по мірі необхідності, а генерує послідовність лексем, тобто ключових слів, що притаманні даному формату. Лексер працює як словник, що перетворює вхідну послідовність символів на лексеми та значення властивостей. До значень властивостей відносяться числа та рядки. До списку реалізованих лексем належать такі лексеми:

- `:treestart;`
- `:treeend;`
- `:nodestart;`

- `:propident;`
- `:propvalue;`
- `:propvalueend.`

Далі використовується парсер – функція, що парсить список лексем, генеруючи дерево властивостей. Саме ця функція повинна знати граматичні правила даного формату. Парсер працює як кінцевий автомат станів, що репрезентують поточний момент в обробці дерева партії. Реалізація даного підходу виглядає як набір рекурсивних функцій, що розпізнають та повертають в обробленому форматі кожна свою структуру даних формату SGF.

Серед структур, що розглядалися у файлах SGF основними є такі:

- Файл – складається з дерев партій.
- Партія – складається зі списку піддерев.
- Піддерево – представляється рекурсивно як список піддерев або пустий список.
- Вузол – складається зі списку властивостей. Може містити піддерева.
- Властивість – задається як пара двох елементів: ідентифікатора властивості та значення властивості.
- Ідентифікатор властивості – у форматі SGF це дві великі латинські літери, що дають назву властивості.
- Значення властивості – число або рядок. Кінцевий елемент SGF-файлу.

Дерево партії, що побудоване на основі SGF-файлу виходить дуже надлишковим. Воно зберігає багато інформації, що потрібна людям, але не потрібна для пошуку на комп’ютері у такій структурі даних. Найчастіше для пошуку потрібні тільки ті властивості у дереві гри, що відповідають позиціям каменів гравців, усі ж інші властивості – зайві. Тому була реалізована функція `simplify`, що фільтрує та спрощує дерево гри, залишаючи тільки послідовність ходів гравців. З такою структурою даних

набагато легше працювати, до того ж вона займає менше пам’яті комп’ютера.

Частіше доцільно використовувати іншу структуру даних, що репрезентує поточний стан дошки. Ця структура вже не дерево, вона має вигляд матриці значень клітинок дошки (тобто ця клітинка пуста, зайнята білим каменем або зайнята чорним каменем). Ця структура більш зручна, якщо треба аналізувати поточний стан дошки, адже дерево не несе в собі подібну інформацію. Дерево зберігає зміни в дошці на момент кожного ходу, а для пошуку по стану дошки зручніше використовувати структуру, що може зберігати кожен стан дошки окремо. Також така структура даних використовується методом пошуку шаблону, саме на її основі рахується хеш поточного стану дошки.

Перетворенням у таку структуру даних з дерева займається функція `tree2board`. Її лістинг наведено нижче.

```
(defn tree2board [tree]
  (let [board (vec (repeatedly 19 (fn [] (vec (repeat 19 :empty)))))
        player (mkplayer)]
    (reduce (fn [board move]
              (let [y (char2index (first move))
                    x (char2index (second move))]
                (assoc board x
                  (assoc (board x) y (stone (player))))))
            board tree)))
```

Лістинг 3.2. Функція tree2board

Ця функція повертає вектор рядків дошки, що складаються з послідовності ідентифікаторів, що репрезентують поточний стан клітинки дошки. Для цього використовуються 3 ідентифікатори:

- `:empty;`
- `:black;`
- `:white.`

Для візуалізації поточного стану дошки використовується функція `print-board`, що виводить на монітор дошку з камінням гравців. Вивід цієї функції схожий на вивід дошки у програмі GnuGo.

3.3. Алгоритм Негамакс

Мінімакс – правило прийняття рішень, що використовується в теорії ігор, теорії прийняття рішень, дослідженні операцій, статистиці і філософії для мінімізації можливих втрат з тих, які особа, яка приймає рішення не може уникнути при розвитку подій за найгіршим для неї сценарієм. Критерій мінімаксу спочатку був сформульований в теорії ігор для гри двох осіб з нульовою сумою для випадків послідовних і одночасних ходів, згодом отримав розвиток у складніших іграх і прийнятті рішень в умовах невизначеності.

Даний модуль реалізує метод Негамакс, що є модифікацією методу Мінімакс. Цей метод спрощує метод Мінімакс використовуючи одну й ту саму функцію як для максимізуючого, так і для мінімізуючого гравця.

Мінімаксний алгоритм потребує функцію генерування можливих ходів (ходи, що не є заборонені правилами гри) та функцію евристичної оцінки поточного стану дошки. Розробка функції оцінки стану виходить за рамки даної роботи, тому замість неї буде використовуватися випадкова величина.

Функція генерування можливих ходів (`movegen` у коді) використовує поточний стан дошки `board` та колір гравця, що робить крок `player`. Вона повертає список можливих ходів у вигляді пар координат. Ця функція повинна розуміти правило зняття з дошки каменів та правило Ко. Розробка цієї функції виходить за рамки розроблювальної системи, тому замість неї буде використовуватися функція-заплата, що повертає всі пусті клітинки поточної дошки.

Алгоритм Негамакс можна легко зробити паралельним. Достатньо шукати максимум кожного піддерева у окремому потоці. В мові Clojure це можна зробити використовуючи функцію `pmap`. Ця функція працює як звичайний `map`, але запускає функції у окремих потоках, повертаючи `Future`. Використовуючи об’єкт такого типу можливо дізнатися коли закінчиться виконання відповідної функції та отримати її результат.

Така реалізація дозволяє просто зробити виконання функції Негамакс паралельним, але вона не дає змогу контролювати цей процес. Якщо є така задача, то паралелізм можна реалізувати самостійно, використовуючи ті самі об’єкти типу `Future`.

Приклад псевдокоду реалізації методу Негамакс наведено у лістингу 3.2.

```
function negamax(node, depth) ◁ returns integer - value of best play
      if node is a terminal node or depth ≤ 0 then
            return the heuristic value of node
      end if
      α ← −∞
      for child in node do ◁ evaluation is identical for both players
            α ← max (α , -minimax(child, depth-1))
      end for
      return α
end function
```

Лістинг 3.2. Псевдокод методу Негамакс

3.4. Альфа-бета відсічення

Метод Негамакс генерує повне дерево гри. Тобто він розглядає усі варіанти ходів та усі можливі варіанти відповідей на ці ходи. Саме тому він дуже обчислювально витратний. Однією з можливих оптимізацій цього методу може бути відсічення вершин, що можливо не розглядати. Тобто не рахувати ті піддерева, обчислення яких не змінить результат.

Альфа-бета відсічення – модифікація методу Мінімакс, алгоритму пошуку, що зменшує кількість вузлів, які необхідно оцінити в дереві пошуку мінімаксного алгоритму і при цьому дозволяє отримати ідентичний результат. Цей алгоритм використовується в програмуванні ігор, де грають два гравці (хрестики-нулики, шахи, Го).

Використовуючи цей алгоритм, програма повністю припиняє оцінювати піддерево, якщо знайшла доказ того, що всі ходи в піддереві гарантовано гірші, ніж якийсь хід, оцінений раніше. Таке піддерево та ходи не потребують подальшого розглядання.

В основі роботи алгоритму стоять два значення: α – найкраще значення маршруту для максимізуючого гравця та β – найкраще значення

маршруту для мінімізуючого гравця. Зрозуміло, що α прямує до плюс нескінченності, а β прямує до мінус нескінченності.

Приклад дерева та роботи методу Альфа-бета відсічення продемонстрований на рис. 3.2. На ньому квадратні вершини – це вершини, де приймає рішення максимізуючий гравець, круглі вершини – де мінімізуючий. Позначені сірим та закреслені рисками ті піддерева, розгляд яких не плине на вирішальний результат, тому їх обчисленням можна знехтувати.

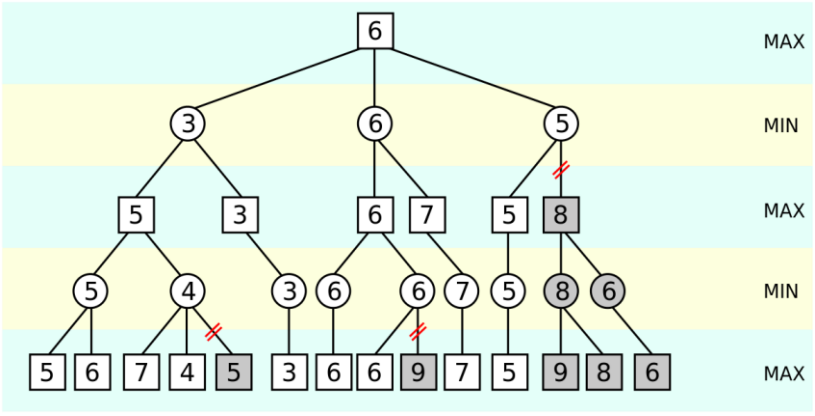

Рис. 3.2. Ілюстрація роботи алгоритму Альфа-бета відсічення

Альфа-бета алгоритм, при найкращому порядку ходів, побудує значно менше дерево перебору. Кількість вузлів приблизно дорівнює кореню квадратному з числа позицій, що переглядаються при повному переборі. Альфа-бета відсічення дуже чутливе до порядку ходів, тому потрібно врахувати, що при найгіршому порядку ходів, тобто коли відсічення за β розглядає останній хід, Альфа-бета відсічення прогляне стільки ж позицій, скільки і Негамакс. Швидкість прорахунку на практиці також дуже залежить від можливого діапазону оцінок.

У цьому алгоритмі також можливо додати паралельність виконання функції. Обробка гілки піддерева може виконуватися у окремому потоці. Таке рішення створює проблему: можливо виконати надмірні обчислення, якщо піддерево, яке можливо не оброблювати обраховується раніше, ніж піддерево, яке містить шуканий шлях. Дивлячись на приклад псевдокоду методу Альфа-бета відсічення на лістингу 3.3, зрозуміло, що доцільно тіло

циклу, що виконується для кожного можливого хода, виконувати як паралельні процеси, що не залежать один від одного.

```
function AlphaBeta(color, depth, α, β)
      if depth = 0 then
            return Evaluate(color)
      end if
      moves ← GenerateMoves
      for move in moves do
            makeMove(move)
            eval ← -AlphaBeta(-color, depth-1, -β, -α)
            unmakeMove(move)
            if eval ≥ β then
                  return β
            end if
            if eval > α then
                  α ← eval
                  if depth = defaultDepth then
                        bestmove ← move
                  end if
            end if
      end for
      return α
end function
```

Лістинг 3.3. Псевдокод методу Альфа-бета відсічення

3.5. Порівняння з шаблоном

Шаблоном у грі Го називається деяка послідовність ходів на локальній частині дошки, що часто використовується і приносить приблизно однакову користь для обох гравців.

Ефективне розпізнавання шаблонів у грі Го та їх грамотне використання є вирішальним як для гравця-людини, так і для програми. На поточний момент було створено базу шаблонів різних етапів партії на основі професійних партій в Го. Така база використовуються майже усіма програмами, що грають у Го.

Створення такої бази не входить в завдання даного дипломного проекту, тому методи пошуку за шаблоном використовують заздалегідь згенеровану базу. Однак розглянемо псевдокод можливого методу генерації подібної бази шаблонів гри Го. В даному прикладі шаблон має заданий розмір 5 на 5. До того ж, для спрощення пошуку шаблонів та їх подальшого використання, нехай перший хід у шаблоні буде завжди в його центр.

```
for each game record R in the game record collection do
      for each move M in R do
            Play M on the game board
            Obtain the 5-by-5 region R centered by M
            Rotate and flip R into its canonical form
            if R is not in our pattern database then
                  Add R into the pattern database
                  Set frequency number of R to be 0
            end if
            Increase the frequency number of R by 1
      end for
end for
```

Лістинг 3.4. Псевдокод алгоритму створення бази шаблонів за базою партій в Го

Для пошуку по такій базі шаблонів доречно використовувати хешування. Якщо метод хешування задовольняє умові адитивності (тобто маючи хеш послідовності можна легко обрахувати та знайти хеш нової послідовності, отриманої декількома ходами з початкової, просто додаючи оператором XOR хеши нових ходів), то пошук по базі шаблонів буде значно спрощено.

Відповідність хешів надає інформацію про входження послідовності ходів партії до шаблону з бази. Якщо ж використовувати метод хешування, що не залежить від послідовності, а тільки від результуючого набору каменів, то можна покращити пошук, адже тепер по результуючому набору каменів можна буде знайти різні піддерева, що привели до нього.

Одним з найпопулярніших методів хешування для настільних ігор є хешування Зобріста. Якщо говорити про це хешування, то порівняння хешів зводиться до виконання операції XOR над хешем шаблону та хешем підпослідовності поточного стану партії. Це є дуже ефективним способом знаходження шаблонів, однак недолік полягає у тому, що таку перевірку треба робити для всіх станів дошки партії починаючи з якогось стану. Тобто цей метод має виконувати повний перебір усіх підпослідовностей та усіх шаблонів з бази. Для знаходження шаблонів у партії з 200 ходів, використовуючи базу даних з 1000 шаблонів треба виконати 200 тисяч операцій XOR, не говорячи про рахування хешу Зобріста. Однак, це є недоліком усіх подібних методів хешування.

Алгоритм пошуку підходящого шаблону складається з таких кроків:

1. Розбиття поточного стану дошки на локальні ситуації, до кожної з яких можна застосувати метод пошуку шаблону.
2. Обчислити хеш Зобріста кожної ситуації.
3. Для кожної ситуації, з усієї бази шаблонів знайти ті, що частково співпадають з нею. Тобто ситуаціє є частиною послідовності ходів даного шаблону.
4. Аналіз знайдених шаблонів та вибір серед них найбільш доречних.
5. Прийняття рішення, щодо подальшого кроку на основі знань, отриманих з знайдених шаблонів.

Більшість пунктів виходять за рамки даної роботи. Розроблювана система містить у собі реалізацію функції, що рахує хеш Зобріста по поточному стану дошки або її частини і функцію пошуку входження шаблону локальної ситуації у базу шаблонів партій з Го.

3.6. Хешування Зобріста

Одним із методів пошуку деякого положення каменів на дошці може бути хешування. Якщо використовувати хешування, по якому можна легко дізнатися, що стоїть на якій клітинці, то функція пошуку деякого зразка на дошці буде простою. Прикладом такого хешування може бути хешування Зобріста.

Хешування Зобріста – це хеш-функція, що використовується в комп’ютерних програмах, які грають в абстрактні настільні ігри, такі як шахи або Го і використовують транспозиційні таблиці, особливий вид хеш-таблиць, що індексуються позиціями дошки і використовується, щоб уникнути аналізу однієї і тієї ж самої позиції декілька разів.

Хешування Зобріста починається з генерування випадкового бітового рядка для кожного можливого елемента настільної гри, тобто для кожної комбінації фігури і положення. Наприклад для шах це 12 штук та 64

позицій дошки, тобто 768 різних бітових рядків; для Го це 2 фігури та 361, тобто 722 рядки. Використовуючи такі рядки будь-яка конфігурація дошки може бути розбита на незалежні компоненти фігура/положення, кожен з яких має відповідний бітовий рядок. Результуючий хеш дошки рахується використовуючи побітове `XOR` між усіма бітовими рядками усіх компонент, тобто фігур та їх положень, на дошці.

Такий самий метод можна використовувати для підрахунку хешу частини дошки, а не усієї. Цей метод дуже універсальний, він підходить майже для всіх настільних ігор.

Безсумнівним доводом використання такого методу хешування для пошуку за базою шаблонів є те, що використовуючи лише одну операцію XOR та маючи хеші можливо перевірити їх відношення між собою та зробити відповідні висновки.

У даній системі хешування Зобріста реалізовано функцією `(zobrist board)`, що приймає стан дошки або її частину (тобто структуру даних типу `board`) і повертає число типу `bigint`.

3.7. Метод Монте-Карло пошуку в дереві

Методи Монте-Карло – загальна назва групи чисельних методів, заснованих на одержанні великої кількості реалізацій стохастичного (випадкового) процесу, який формується у той спосіб, щоб його ймовірнісні характеристики збігалися з аналогічними величинами задачі, яку потрібно розв’язати. Використовується для розв’язування задач у фізиці, математиці, економіці, оптимізації, теорії управління тощо. Метод Монте-Карло – це метод імітації для приблизного відтворення реальних явищ. Він об’єднує аналіз чутливості (сприйнятливості) і аналіз розподілу ймовірностей вхідних змінних. Цей метод дає змогу побудувати модель, мінімізуючи дані, а також максимізувати значення даних, які використовуються в моделі. Побудова моделі починається з визначення функціональних залежностей у реальній системі. Після чого можна

одержати кількісний розв’язок, використовуючи теорію ймовірності й таблиці випадкових чисел.

Для пошуку у структурах даних гри Го існує метод Монте-Карло пошуку в дереві. Він генерує велику кількість випадкових партій до кінця гри. Потім він рахує статистику виграшу та програшу у цих іграх. Використовуючи цю статистику, він приймає рішення щодо наступного ходу, вибираючи хід, що має найбільшу вірогідність виграшу.

Як видно на плакаті 1, цей метод складається з чотирьох кроків:

1. Вибір – перший крок методу Монте-Карло пошуку в дереві. На цьому кроці алгоритм обирає найбільш підходящу вершину дерева для продовження моделювання. Функція, що реалізує цей крок виконана у вигляді рекурсивного спуску у дереві, обираючи вершини з найбільшим значенням `value` у кожній.
2. Розширення – єдиний крок, на якому створюється нова вершина. Нова вершина створюється випадковим чином як продовження вершини, обраної на попередньому кроці. Вершина обирається випадковим чином зі списку можливих ходів, згенерованих функцією `genmove`.
3. Моделювання – процес створення у пам’яті піддерева великої кількості випадковим чином згенерованих партій з Го. Як і на попередньому етапі ходи обох гравців обираються випадковим чином зі списку можливих ходів. Для спрощення подальшого переобчислення ще на етапі моделювання дерева варто зберігати winrate – статистику виграшів та програшів першого гравця (того, за якого грає комп’ютер). Для методу Монте–Карло пошуку в дереві потрібна статистика включає в себе тільки відношення виграшів до програшів у піддереві вершини, що розглядається. Інші модифікації цього алгоритму потребують більшу кількість параметрів для розглядання та збереження у вузлах дерева.

4. Переобчислення – останній етап методу Монте-Карло пошуку в дереві. Після моделювання великої кількості ігор створену статистику виграшів та програшів потрібно зберегти для подальшого використання. Враховуючи те, що деякі значення winrate у всіх вершинах вже є (вони були створенні на попередніх ітераціях алгоритму), на даному кроці алгоритму потрібно оновити ці значення, тобто – переобчислити їх.

Після останнього кроку алгоритму виконання переходить знову до першого. Подібне зациклення відбувається до того моменту, поки не буде прийнято рішення зупинити роботу алгоритму. Подібне рішення може бути прийнято на основі кількості змодельованих ігор, поточній глибині пошуку або ж на основі розміру дерева у пам’яті або обчислювальної витратності останньої ітерації алгоритму.

3.8. Метод верхньої оцінки значущості для дерева

Існують методи, що покращують роботу попереднього методу. Вони змінюють функцію оцінки значущості вузлів, що змінює порядок обходу та створення дерева варіантів партій у пам’яті. Один із прикладів може бути метод UCT – варіація методу Монте-Карло для пошуку в дереві що використовує змінену функцію верхньої оцінки значущості для дерева.

Цей метод потребує розширення додаткової інформації, що зберігається в структурі дерева варіантів гри. Він додає нове значення до статистики виграшів та програшів у кожний вузол – кількість раз, що цей вузол був відвіданий. На основі цих значень, використовуючи функцію, показану нижче, алгоритм вираховує значення значущості вузлів дерева. На першому етапі вибору шляху для продовження аналізу використовуються саме ці значення:

$$UCTValue(parent, n) = winrate + \sqrt{\frac{\ln(parent.visits)}{5 \times n.nodevisits}}.$$

Така нова функція додає до статистики виграшів нове значення, що зменшується кожен раз, коли дану вершину було обрано на якійсь ітерації і збільшується кожен раз, коли було обрано якусь іншу дочірню вершину батьківської вершини (тобто іншу вершину, що має спільну батьківську вершину і яка знаходиться на тому самому рівні глибини). Таке змінення оцінювальної функції дозволяє сконцентруватися на варіантах, які дають велику ймовірність виграшу у той же час не забуваючи про створення нових варіантів гри.

4. ПОРІВНЯЛЬНИЙ АНАЛІЗ ТА МЕТОДИКА ТЕСТУВАННЯ

Системи, аналогічні створюваній, завжди є частинами більших систем що грають в Го або аналізують партії з Го. Саме тому дуже важко порівнювати їх швидкодії, адже їх не можна використовувати без глобального контексту системи. У розробленій системі це зробити можливо, адже у ній методи пошуку реалізовані у вигляді окремих модулів.

Тому краще оцінювати і порівнювати розроблену систему з аналогічними по розміру та простоті використання.

4.1. Порівняння розміру систем пошуку інформації у структурах даних гри Го

Розроблена система, на відміну Master Go, поставляється у вигляді бібліотеки, тобто jar-файлу, який можливо використовувати в усіх мовах програмування, що підтримують модулі мови програмування Java. Тому це можливо майже в усіх мовах, що використовують платформу Java.

Інші ж системи розроблені використовуючи мову програмування C++, до того ж вони не створені у вигляді бібліотек. Тому для використання їх як окремих модулів у своїй системі, спочатку треба виділити їх у бібліотеку, що може бути досить складною задачею.

У системі Fuego модулі розроблені так, що їх можливо використовувати окремо. Fuego складається з п’яти окремих бібліотек:

- GtpEngine – реалізація протоколу GTP.
- SmartGame – додаткові функції, що можуть бути використані у різних настільних іграх.
- Go – класи та функції, специфічні для Го.
- SimplePlayers – проста реалізація гравця для Го.
- GoUct – реалізація методу UCT для Го.

У розробленій системі модулі розбиті приблизно таким самим чином, але вона має більше методів для пошуку у структурах даних гри Го. Однак їй бракує системи, що реалізує протокол GTP. Це текстовий протокол, що найчастіше використовується програмами, що грають в Го для комунікації між собою. При подальшому розвитку системи, цей протокол має бути реалізований, адже він є ключовим для комунікації між системами.

Якщо порівнювати розміри програмних кодів продуктів, то програмний код Kombilo займає 1.9 мегабайт, програмний код Fuego займає 1.1 мегабайт, а програмний код розробленої системи займає близько 100 кілобайт. Однак таке порівняння не зовсім доречно, адже розглянуті системи мають зовсім різний рівень функціональних можливостей.

Однією з суттєвих відмінностей розробленої системи безумовно є використання платформи Java. Саме завдяки такій реалізації є можливим використовувати бібліотеку у будь-якій програмі, що використовує можливості даної платформи. Розроблена система постає у вигляді модуля Java з назвою `com.kpi.diploma.sgl`. Наведемо приклад, у якому розроблена система використовується у програмах, що написані на мові програмування Java.

```
import com.kpi.diploma.sgl;

public class Main {

    public static void main(String[] args) {
        System.out.println("parsed tree: " +
            sgl.sgf2tree("resources/game.sgf"));
    }
}
```

Лістинг 4.1. Використання розробленої системи у програмі на мові програмування Java

Через те, що більшість систем, аналогічних розроблюваній не модульні і порівнювати їх безпосередню швидкодію буте досить важко,

доречно буде порівняти загальну швидкодію мов, на яких написані порівнювані системи, тобто мов C++ та Java.

Для порівняння була обрана мова Java, а не Clojure, адже розроблена система Clojure все одно компілюється в Java байт-код та поставляється у вигляді jar-пакету.

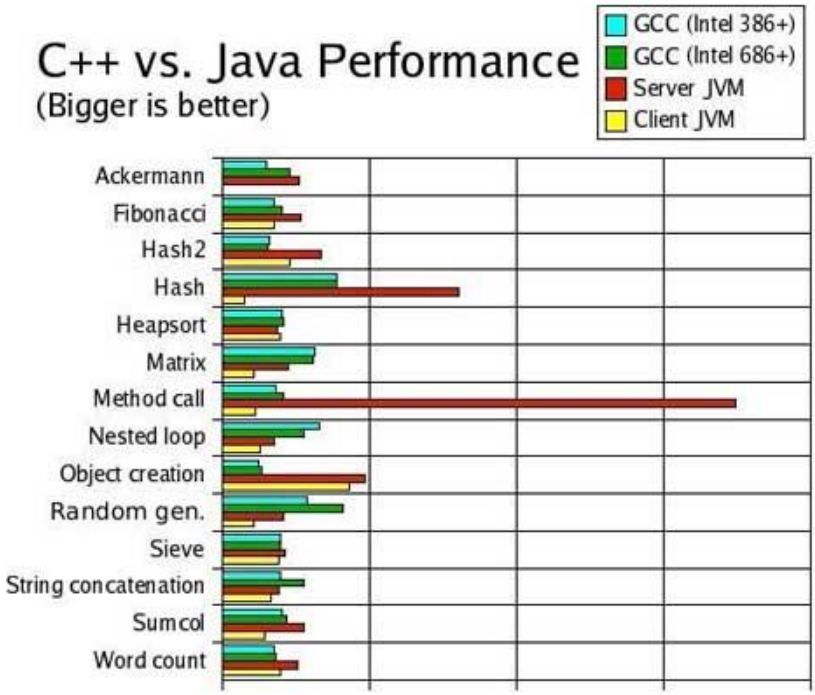

Рис. 4.1. Графік порівняння швидкодії мов програмування C++ та Java

4.2. Методика тестування системи пошуку у структурах даних гри Го

Основна увага при тестуванні була зосереджена на тестах модулів та функцій, тобто unit-тестуванню. Було створено набір тестів до кожної функції пошуку та роботи з SGF-файлами.

```
(deftest sgf2tree_test
    (is (= first-result (sgf2tree first-file)))
    (is (= second-result (sgf2tree second-file))))
 (deftest tree2board_test
    (is (= first-board (tree2board first-result)))
    (is (= second-board (tree2board second-result))))
```

Лістинг 4.2. Приклад unit-тестів для функцій, що працюють з SGF-файлами

5. ОХОРОНА ПРАЦІ

Дана дипломна робота передбачає розробку програмного засобу для пошуку у структурах даних гри Го. Розробка даної програми відбувається в кімнаті офісу на чотирьох осіб, у кожної з яких є робоче місце на один комп’ютер.

Правильно організована робота по забезпеченню безпеки праці підвищує дисциплінованість працівників, поліпшує умови праці, що в свою чергу, призводить до підвищення продуктивності праці, зниження кількості нещасних випадків, запобігання виходу з ладу обладнання та інших нештатних ситуацій.

Покращення умов праці та її безпека призводить до зменшення виробничого травматизму, професійних хвороб, що зберігає здоров’я працівників та одночасно призводить до зменшення затрат на оплату пільг та компенсацій, на оплату наслідків такої роботи, на лікування, перепідготовку працівників виробництва в зв’язку зі зміною кадрів через причини, що пов’язані з умовами праці.

5.1. Аналіз робочого місця

Приміщення, в якому розроблювалася система буде розташоване на другому поверсі п'ятиповерхового будинку та розраховано на чотири робочих місця. Схема приміщення представлена на рис. 5.1.

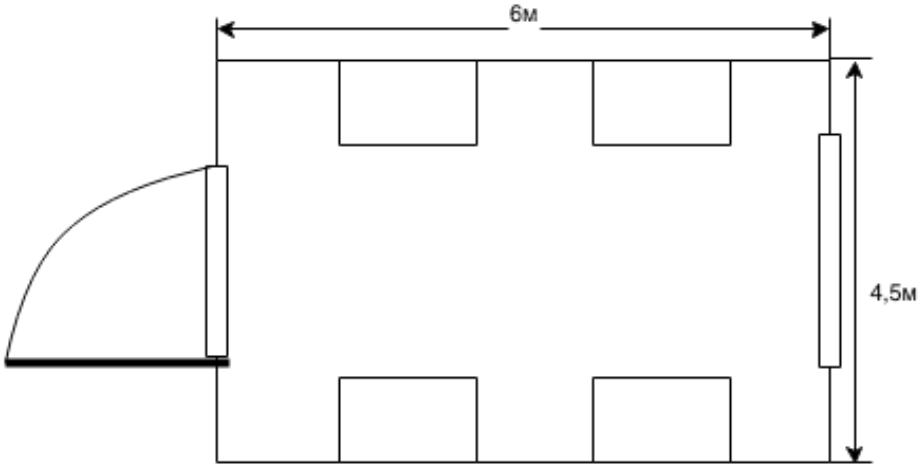

Рис. 5.1. Схематичний план приміщення

Виміри приміщення представлені в табл. 5.1.

Таблиця 5.1

Виміри приміщення

Розмір	Значення, м
Ширина	4.5
Довжина	6.0
Висота	3.5

В приміщенні є одне вікно розмірами: 1.5 м шириною і 2 м висотою.

Згідно з вимогами НПАОП 0.00-1.28-10 [12], розміри приміщення повинні задовольняти наступним вимогам: площа, не менше 6 м^2 і об’єм, не менше 20 м^3 з розрахунку на одне робоче місце. Розрахуємо значення цих параметрів для нашого приміщення:

Площа даного приміщення:

$$S = 6 \text{ м} * 4.5 \text{ м} = 27 \text{ м}^2.$$

Об'єм даного приміщення:

$$V = S * h = 27 \text{ м}^2 * 3.5 \text{ м} = 94.5 \text{ м}^3.$$

Таким чином, на одне робоче місце надано:

$$S = 6.75 \text{ м}^2, V = 23.625 \text{ м}^3.$$

Величина площі та об’єму, що припадають на одного працівника задовольняють вимогам.

5.2. Аналіз шкідливих і небезпечних факторів

5.2.1. Мікроклімат

Санітарні норми мікроклімату виробничих приміщень в цьому розділі описані згідно ДСН 3.3.6.042-99 [13]. Робота, виконувана в даному приміщенні відноситься до категорії робіт – «Легка 1б». У приміщеннях з використанням обчислювальної техніки рекомендується застосування

тільки оптимальних значень показників мікроклімату. Нижче приведено відповідні санітарні вимоги до мікроклімату в приміщенні, що повинні дотримуватися.

Таблиця 5.2

Оптимальні значення параметрів мікроклімату категорії робіт Легка-1б

	Холодний період року	Теплий період року
Температура повітря, С	21-23	22-24
Відносна вологість повітря, %	40-60	40-60
Швидкість руху повітря, м/с	0.2	0.1

У приміщенні встановлені батареї центрального водяного опалення, що включається в холодний період року. У теплу пору працює, система кондиціонування, що складається з кондиціонера спліт-системи Кондиціонер DELFA ADW-07C з потужністю 1100 Вт.

5.2.2. Освітлення

В приміщеннях для роботи з ЕОМ повинне використовуватися як природне так і штучне освітлення. Природне освітлення забезпечує вікно, загальна площа якого складає 3 м^2. Воно являється боковим та одностороннім.

Нормоване значення КПО, яке має забезпечувати природне освітлення розраховується за формулою:

$$e_n = \frac{S_{вік}}{S_n} = \frac{3}{27} = 0.11,$$

де e_n – значення КПО; $S_{вік}$ – загальна площа вікна, м^2; S_n – площа підлоги, м^2.

Отримане значення 0.11 менше ніж встановлено нормами (КПО має

бути не меншим за 0.15), тобто природного освітлення не вистачає для нормальної роботи. Тому потрібно використовувати штучне освітлення.

Штучне освітлення в приміщеннях з робочими місцями, обладнаними ВДТ ЕОМ та ПЕОМ, має здійснюватися системою загального рівномірного освітлення. У якості джерел світла для штучного освітлення мають застосовуватись переважно люмінесцентні лампи типу ЛБ, потужністю 20Вт. Для загального освітлення слід застосовувати 2 світильники серії ЛПО, розташовані у 2 ряди. Один світильник містить 2 лампи, кожна з яких має світловий потік 1060 лм. Нормативна освітленість 300-400 лк., згідно ДБН В.2.6-31 [14]. Штучне освітлення кімнати створює освітленість 340 лк, що задовольняє стандарту.

5.2.3. Шум

Основним джерелом шуму є системний блок комп'ютера, який містить такі компоненти як: жорсткий диск та кулер.

Таким чином у приміщенні мають місце шуми механічного і аеродинамічного походження. Шум, що створюється, умовно можна віднести до постійного.

Згідно з ДСН 3.3.6.037-99 [15] допустимий шум на постійних робочих місцях користувача складає до 50 дБА. Орієнтовні еквівалентні рівні звукового тиску джерел шуму, що діють на користувача на його робочому місці, представлені в табл. 5.3.

Розрахуємо середній рівень шуму на робочому місці користувача при роботі всієї вказаної техніки.

Рівень шуму, що виникає від декількох некогерентних джерел, що працюють одночасно, підраховується на підставі принципу енергетичного підсумовування рівня інтенсивності окремих джерел:

$$L = 10 \cdot \lg\left(\sum_{i=0}^{n} 10^{0.1 \cdot L_i} \right),$$

де L_i – рівень звукового тиску i-го джерела.

Підставивши значення рівня звукового тиску для кожного виду устаткування у формулу, отримаємо:

$$L = 10 \times \lg\left(4 \times 10^{2.6} + 4 \times 10^{2.8}\right) = 36\,(\text{дБА}).$$

Розраховане значення рівня шуму не перевищує гранично допустимого рівня шуму для робочого місця користувача (50 дБА), тобто спеціальні заходи по зниженню рівня шуму не потрібні.

Таким чином, робота з системою, розробленою в дипломній роботі, являється безпечною і не потребує додаткових улаштувань для зниження шуму, окрім загальних методів ізоляції від зовнішнього шуму. Для цього застосовуються спеціальні віконні профілі та звукоізоляція зовнішніх стін плитами зі звукоізоляційними наповнювачами.

5.2.4. Електромагнітні випромінювання

Більшість учених вважає, що як короткочасна, так і тривала дія всіх видів випромінювання від екрану монітора не небезпечна для здоров'я людини. Проте вичерпних даних щодо небезпеки дії випромінювання від моніторів на людей, що працюють з комп'ютерами не існує і дослідження в цьому напрямі продовжуються.

Допустимі значення параметрів не іонізуючого електромагнітного випромінювання від монітора комп'ютера представлені в табл. 5.3.

Таблиця 5.3

Допустимі значення параметрів не іонізуючого електромагнітного випромінювання

Найменування параметра	Допустимі значення
Напруженість електричної складової електромагнітного поля на відстані 50см від поверхні монітора	10 В/м
Напруженість магнітної складової електромагнітного поля на відстані 50см від поверхні монітора	0.3 А/м

Продовження табл. 5.3

Напруженість поля не повинна перевищувати: • для дорослих користувачів • для дітей дошкільних установ і що вчяться в середніх спеціальних і вищих учбових закладах	 20 кВ/м 15 кВ/м

Для зниження дії цих видів випромінювання рекомендується застосовувати монітори із зниженим рівнем випромінювання (MPR-II, TCO-92, TCO-99, TCO-03) [16], а також дотримувати регламентовані режими праці і відпочинку.

5.3. Електробезпека

Робоче місце підпадає під категорію без підвищеної небезпеки тому, що температура у кімнаті не перевищує 30 ^{0}C, вологість не перевищує 60%, кожен день робиться вологе прибирання.

Електроустаткування належить до приладів до 1000 В. Устаткування, що використовується, відповідно до ПУЄ належить до устаткування класів 0, 0I, і I за електрозахистом.

Оцінка небезпеки дотику до струмових частин відноситься до визначення сили струму, що протікає через тіло людини, і порівняння його із допустимим значенням відповідно до ГОСТ 12.1.038-88 [17].

Лінія електромережі для живлення персональних комп'ютерів, їх периферійних пристроїв (принтер) виконується як окрема групова три-провідна мережа, шляхом прокладання фазового, нульового робочого та нульового захисного провідників. Нульовий захисний провідник використовується для заземлення електроприладів. Провід мідний, ізоляція має бути закритою, марки ПУНП, перерізом не менше 2,5х2мм на жилу.

Частота струму не має перевищувати значення 50 Гц.

При виконанні розрахунків для дипломного проекту використовувався персональний комп'ютер – I і II клас захисту, що живиться напругою 220 В. Для правильного визначення необхідних засобів та заходів захисту від ураження електричним струмом необхідно знати допустимі значення напруги доторкання та струмів, що проходять через тіло людини.

Гранично допустимі значення напруги доторкання та сили струму для нормального (безаварійного) та аварійного режимів електроустановок при проходженні струму через тіло людини по шляху “рука – рука” чи “рука – ноги” регламентуються ГОСТ 12.1.038-88 [18] (табл. 5.3).

Таблиця 5.4

Граничнодопустимі значення напруги доторкання та сили струму, що проходить через тіло людини при нормальному режимі електроустановки

Вид струму	В (не більше)	мА (не більше)
Змінний, 50Гц	2	0.3
Змінний, 400Гц	3	0.4
Постійний	8	0.1

5.4. Пожежна безпека

Згідно з НАПБ Б.03.002-2007 таке приміщення відноситься до категорії В–пожежонебезпечна. При нормальному режимі роботи можливість виникнення пожежі мінімальна. Можливість виникнення вибухів повністю відсутня. Можливими причинами загоряння можуть бути пошкодження та замикання в електромережі та електрообладнанні, а також порушення правил безпеки при роботі з обладнанням.

На робочому місці наявні наступні пожежонебезпеці матеріали: папір, пластик, віконні рами, дерев’яні шафи, корпуси техніки, меблі. Робоче приміщення повинно бути обладнане двома вуглекислотними

вогнегасниками ВВК-5 з розрахунку два вогнегасника на приміщення до 25 м^2 включно, що задовольняє НАПБ Б.03.002-2007. Для захисту від блискавки будівля обладнана блискавковідводом стрижневого типу.

В приміщенні посередині стелі має бути встановлений один димовий пожежний сповіщувач СПД-3 відповідно до ДБН В.1.1.-7-2002 – з розрахунку один на висоту до 3.5 м та загальною площею не більше ніж 86 м^2.

ВИСНОВКИ

У даному дипломному проекті було проаналізовано програми, що дозволяють шукати у структурах даних гри Го, проаналізовано найпопулярніші алгоритми для пошуку у таких структурах даних. На основі проаналізованих даних, було прийнято рішення створити систему пошуку у структурах даних гри Го, що буде закривати недоліки розглянутих систем.

Серед розглянутих методів було обрано реалізувати такі: метод Негамакс, метод Монте-Карло пошуку в дереві та метод порівняння з шаблоном. Також було прийнято рішення реалізувати додаткові алгоритми, такі як метод Альфа-бета відсічення та метод верхньої оцінки значущості для дерева.

Також було проведено порівняльний аналіз платформ доступних для розробки та мов програмування на цих платформах. З усіх платформ було обрано платформу Java як одну з найбільш популярних та розповсюджених платформ на даний момент. Проаналізувавши мови програмування, що доступні на даній платформі було обрано мову програмування Clojure, адже вона найкраще підходить для розробки подібної системи через те, що являється Lisp-мовою та має гарну підтримку паралельного програмування.

Була розроблена система, що дозволяє працювати з SGF-файлами: відкривати їх, читати та перетворювати у формат внутрішнього дерева; система, що дозволяє використовувати чотири алгоритми пошуку в дереві гри Го: метод Негамакс, метод Альфа-бета відсічення, метод порівняння з шаблоном і метод Монте-Карло пошуку в дереві; і система, що дозволяє візуалізовувати результат виконання алгоритмів у вигляді простої дошки.

Порівнявши розроблену систему з існуючими аналогами було зроблено висновок, що вона не є такої ефективною, як аналоги, але потенційно може отримати більшу популярність та підтримку. До того ж

для любителя не стільки принципова швидкодія системи, скільки те, чи зручно її використовувати та які можливості ця система надає.

Розроблена система є модульною бібліотекою, тобто вона дозволяє використовувати розроблені методи окремо один від одного. Ця властивість надає можливість для простої підтримки та розвитку розробленої системи.

СПИСОК ВИКОРИСТАНИХ ЛІТЕРАТУРНИХ ДЖЕРЕЛ

1. Wikipedia. Game of Go [Електронний ресурс]. — Режим доступу : https://en.wikipedia.org/wiki/Go_(game). — Дата доступу : грудень 2013.
2. Wikipedia. Zero-sum game [Електронний ресурс]. — Режим доступу : https://en.wikipedia.org/wiki/Zero-sum. — Дата доступу : грудень 2013.
3. Kombilo – Go database program [Електронний ресурс]. — Режим доступу : http://www.u-go.net/kombilo/. — Дата доступу : грудень 2013.
4. Master Go at Sensei’s Library [Електронний ресурс]. — Режим доступу : http://senseis.xmp.net/?MasterGo. — Дата доступу : грудень 2013.
5. Bi Go Software [Електронний ресурс]. — Режим доступу : http://bigo.baduk.org/. — Дата доступу : грудень 2013.
6. Wikipedia. Minimax Tree Search [Електронний ресурс]. — Режим доступу : https://en.wikipedia.org/wiki/Minimax. — Дата доступу : грудень 2013.
7. UCT at Sensei’s Library [Електронний ресурс]. — Режим доступу : http://senseis.xmp.net/?UCT. — Дата доступу : грудень 2013.
8. Programming Language – Groovy [Електронний ресурс]. — Режим доступу : http://groovy.codehaus.org/. — Дата доступу : грудень 2013.
9. The Scala Programming Language [Електронний ресурс]. — Режим доступу : http://www.scala-lang.org/. — Дата доступу : грудень 2013.
10. Clojure – Home [Електронний ресурс]. — Режим доступу : http://clojure.org/. — Дата доступу : грудень 2013.
11. Fuego [Електронний ресурс]. — Режим доступу : http://fuego.sourceforge.net/. — Дата доступу : грудень 2013.
12. Правила охорони праці під час експлуатації електронно-обчислювальних машин. ДНАОП 0.00-1.31-99 (затверджено наказом

Державного комітету України з промислової безпеки, охорони праці та гірничого нагляду від 10.02.1999р. № 65) [Текст].

13. Санітарні норми мікроклімату виробничих приміщень. ДСН 3.3.6.042-99 (затверджено Постановою Головного державного санітарного лікаря України від 1.12.1999 р. № 42) [Текст].
14. Державні будівельні норми. Інженерне обладнання будинків і споруд. Природне і штучне освітлення. ДБН В.2.5-28-2006 (затверджено наказом Міністерства будівництва, архітектури та житлово-комунального господарства України від 15 травня 2006 р. № 168) [Текст].
15. Санітарні норми виробничого шуму, ультразвуку та інфразвуку. ДСН 3.3.6.037.99 (затверджено Постанова Головного Державного санітарного лікаря України від 1.12.1999 р. № 37) [Текст].
16. Электробезопасность. Защитное заземление, зануление. ГОСТ 12.1.030-81 (Затверджено Постановлением Государственного комитета СССР по стандартам от 15.05.81 № 2404) [Текст].
17. Норми визначення категорій приміщень, будинків та зовнішніх установок за вибухопожежною та пожежною небезпекою. НАПБ Б.03.002-2007. (затверджено наказом МНС України від 03.12.2007 № 833) [Текст].
18. Пожежна безпека на об’єктах будівництва ДБН В.1.1.7–2002 (затверджено наказом Держбуду України від 03.12. 2002 р. № 88) [Текст].

ДОДАТКИ