МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЖИТОМИРСЬКИЙ АГРОТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ

ВІДДІЛЕННЯ
«ІНЖЕНЕРНА ІНФРАСТРУКТУРА ТА КОМП’ЮТЕРНІ НАУКИ»

ЦИКЛОВА КОМІСІЯ
«ІНЖЕНЕРНА ІНФРАСТРУКТУРА ТА КОМП'ЮТЕРНІ НАУКИ»

ПОЯСНЮВАЛЬНА ЗАПИСКА

до дипломної роботи освітнього ступеня «фаховий молодший бакалавр»
за спеціальністю 122 Комп'ютерні науки
на тему:

«Гра Ґо»

Виконав здобувач освіти групи П-42
Постовітько Максим Дмитрович

Керівник роботи:
Устименко Людмила Миколаївна

Рецензент:
Устименко Ярослав Іванович

Житомир – 2025

Зміст

РЕФЕРАТ

Записка: 53 стор., 23 рис., 1 таблиця, 1 додаток, 20 джерел.

Ключові слова: гра Ґо, WPF, C#, програмування, інтерфейс, алгоритми, хешування, Undo/Redo.

Об’єкт дослідження — розробка гри Ґо, предмет — технологія WPF. Мета — створення локального додатка для двох гравців із правилами гри, вибором розміру дошки, підрахунком очок і Undo/Redo. Проаналізовано аналоги (Sabaki, OGS), обґрунтовано вибір WPF за гнучкість і продуктивність. Вивчено правила Ґо як основу алгоритмів.

Розроблено архітектуру з класами Game, Board, GameBoard, MainWindow. Реалізовано алгоритми перевірки ходів і підрахунку очок ($O(n^2)$), оптимізовано правило Ко хешуванням до $O(1)$. Програма підтримує дошки 9x9, 13x13, 19x19 та Komi. Інтерфейс створено в Visual Studio 2022 (.NET 4.8) з XAML і C#, включає дошку та панель керування. Вимоги: Windows 10, 2 ГБ ОЗП, 50 МБ пам’яті.

Додаток придатний для навчання та розваг, алгоритм Ко цінний для настільних ігор. Інструкція полегшує використання. Мета досягнута: створено стабільний додаток. Перспективи — ШІ-суперник, збереження партій, оптимізація групування.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

WPF	Windows Presentation Foundation
.NET	Платформа розробки від Microsoft
C#	Мова програмування
XAML	Extensible Application Markup Language (Розширювана мова розмітки додатків)
Go	Гра Ґо (настільна стратегічна гра)
Ko	Правило Ко (заборона повторення позиції дошки)
Komi	Компенсаційні очки для білих гравців
Undo/Redo	Скасування/повторення ходів
BFS	Breadth-First Search (Пошук у ширину)
$O(n^2)$	Часова складність алгоритму (квадратична)
boardStone	Двовимірний масив каменів на дошці
CellState	Стан клітинки (чорний, білий, порожній, поза межами)
Game	Клас гри
Board	Клас дошки
GameBoard	Клас графічного відображення дошки
Stone	Структура, що представляє камінь
scoreWhite	Очки білих гравців
scoreBlack	Очки чорних гравців
Player1	Перший гравець (чорні камені)
Player2	Другий гравець (білі камені)

ВСТУП

Гра Ґо є однією з найдавніших стратегічних настільних ігор, що виникла в Китаї понад 2500 років тому. Її унікальність полягає в простоті правил і водночас надзвичайній глищині стратегії, що робить її популярною серед гравців різного рівня по всьому світу. На відміну від шахів чи шашок, де основна мета — знищення фігур суперника, у Ґо акцент робиться на контролі території, що вимагає від гравців високого рівня абстрактного мислення та тактичних навичок. Завдяки своїй складності та багатогранності, гра привертає увагу не лише любителів настільних ігор, а й дослідників у галузі штучного інтелекту та програмування.

У сучасному світі інформаційних технологій створення комп'ютерних версій настільних ігор є актуальним завданням, яке дозволяє не лише популяризувати ці ігри, а й досліджувати нові підходи до розробки програмного забезпечення. Однією з популярних технологій для створення графічних інтерфейсів є Windows Presentation Foundation (WPF), яка базується на платформі .NET. WPF надає широкі можливості для побудови інтерактивних додатків із гнучким дизайном, підтримкою анімацій та ефективною обробкою подій, що робить її ідеальним вибором для реалізації гри Ґо.

Метою даної дипломної роботи є розробка комп'ютерної версії гри Ґо з використанням технології WPF, яка включає повноцінний графічний інтерфейс, логіку гри та підтримку основних правил, таких як захоплення каменів, перевірка самогубства та правило Ко. У процесі роботи передбачається реалізувати функціонал для двох гравців із можливістю вибору розміру дошки, підрахунку очок, а також скасування та повторення ходів. Особлива увага приділяється оптимізації алгоритмів для забезпечення швидкої та стабільної роботи програми.

Актуальність роботи зумовлена зростаючим інтересом до інтелектуальних ігор у цифровому форматі, а також необхідністю створення доступних інструментів для вивчення та практики гри Ґо. Розроблений

додаток може бути використаний як для розваг, так і для навчання, сприяючи популяризації цієї гри серед широкої аудиторії. Окрім того, проект дозволяє поглибити знання в галузі програмування, зокрема у використанні WPF для створення складних графічних додатків.

Об’єктом дослідження є процес розробки комп’ютерної гри Ґо, а предметом — технологія WPF як засіб реалізації графічного інтерфейсу та логіки гри. У ході роботи будуть розглянуті основні принципи гри Ґо, особливості технології WPF, а також методи оптимізації алгоритмів для забезпечення ефективності програми.

Структура дипломної роботи включає вступ, теоретичну частину, практичну реалізацію, аналіз результатів та висновки. У теоретичній частині будуть описані правила гри Ґо та основи технології WPF. Практична частина присвячена розробці додатка, включаючи проектування інтерфейсу, програмування логіки та оптимізацію коду. У висновках будуть підведені підсумки роботи та окреслені перспективи подальшого розвитку проекту.

РОЗДІЛ 1. ОГЛЯД ТА АНАЛІЗ ТЕХНОЛОГІЙ РОЗРОБКИ

1.1. Постановка основної задачі розробки

Гра Ґо, як одна з найскладніших стратегічних настільних ігор, вимагає від розробника комп'ютерної версії не лише точного відтворення її правил, але й створення зручного та функціонального інтерфейсу для взаємодії користувача з програмою. Основна задача розробки полягає у створенні програмного додатка на базі технології Windows Presentation Foundation (WPF), який забезпечить повноцінний ігровий процес для двох гравців із підтримкою ключових правил гри, таких як розміщення каменів, захоплення території, перевірка валідності ходів (включаючи правила самогубства та Ко), а також підрахунок очок у кінці гри.

Для реалізації поставленої мети необхідно вирішити низку підзадач:

1. Створити інтуїтивно зрозумілий інтерфейс, який включає ігрову дошку з можливістю вибору її розміру (9x9, 13x13, 19x19), відображення поточного стану гри (каміння, очки гравців, черга ходу), а також елементи керування, такі як кнопки для пропуску ходу, скасування та повторення ходів.
2. Забезпечити коректну обробку ходів гравців, включаючи перевірку їхньої валідності відповідно до правил Ґо, оновлення стану дошки після кожного ходу та підрахунок захоплених каменів і території.
3. Розробити ефективні алгоритми для перевірки правил гри (зокрема, правила Ко та самогубства), щоб забезпечити швидкодію програми навіть на великих розмірах дошки (19x19).
4. Реалізувати механізм збереження історії ходів із можливістю скасування та повторення дій гравців.
5. Додати опціональне використання правила Komi, яке компенсує перевагу першого гравця (чорних) шляхом надання додаткових очок другому гравцеві (білим).

Основним інструментом розробки обрано технологію WPF, яка дозволяє створювати сучасні графічні додатки з високим рівнем інтерактивності та

гнучкості дизайну. WPF базується на мові розмітки XAML для опису інтерфейсу та мові програмування C# для реалізації логіки, що забезпечує чітке розділення презентаційної та функціональної частин програми. Вибір WPF зумовлений її перевагами, такими як підтримка векторної графіки, легка інтеграція з платформою .NET, а також широкі можливості для обробки подій і створення анімацій, що є важливим для візуалізації ігрового процесу.

Очікуваним результатом розробки є працездатний додаток, який дозволить користувачам грати в Ґо на комп'ютері, зберігаючи автентичність правил гри та забезпечуючи комфортний користувацький досвід. Програма повинна бути стабільною, швидкою та адаптованою до різних розмірів дошки, що робить її придатною як для початківців, так і для досвідчених гравців. У процесі розробки також передбачається аналіз альтернативних технологій та обґрунтування вибору WPF як оптимального рішення для даного проекту.

Основна задача розробки полягає у створенні повнофункціонального програмного продукту, який поєднує в собі точну реалізацію правил гри Ґо, зручний інтерфейс і високу продуктивність, що відповідає сучасним вимогам до настільних ігор у цифровому форматі.

1.2. Огляд та порівняння аналогічних систем.

Для оцінки доцільності розробки комп'ютерної версії гри Ґо на базі технології WPF необхідно розглянути існуючі аналогічні системи, їхні функціональні можливості, технології реалізації та особливості використання. На ринку існує низка програмних продуктів, які реалізують гру Ґо в цифровому форматі. У цьому підрозділі розглядаються кілька популярних аналогів, їхні переваги та недоліки, а також проводиться порівняння з пропонованим проектом.

Sabaki — це кросплатформовий додаток із відкритим вихідним кодом, призначений для гри в Ґо, аналізу партій та редагування файлів у форматі SGF

(Smart Game Format). Він підтримує різні розміри дошки (9x9, 13x13, 19x19) та має інтеграцію з ШІ-двигунами, такими як Leela Zero.

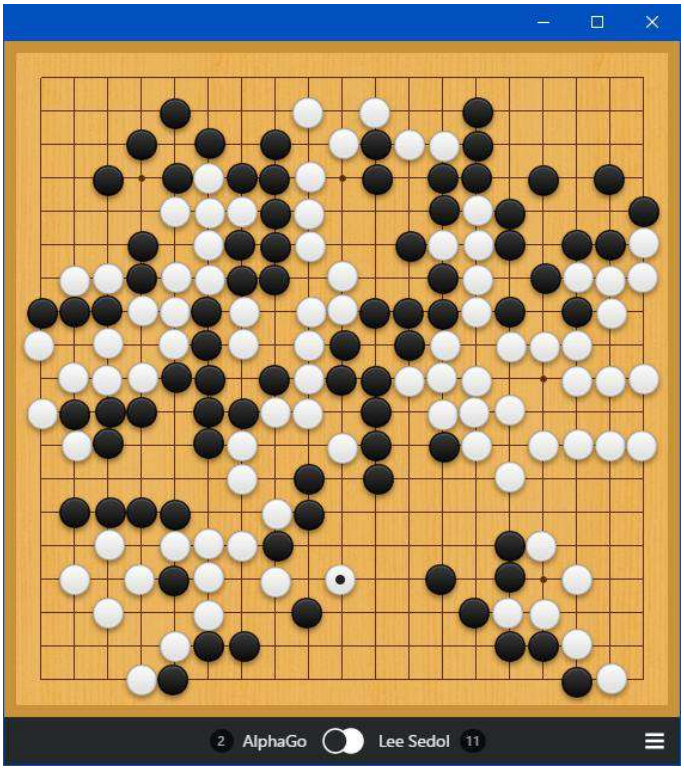

Рисунок 1.2.1 – Інтерфейс Sabaki

Розроблений на JavaScript із використанням фреймворку Electron, що дозволяє запускати його на Windows, macOS і Linux. Легкий у використанні інтерфейс, підтримка аналізу ходів за допомогою ШІ, можливість роботи офлайн, кросплатформність. Відсутність вбудованого мультиплеєра, складність налаштування ШІ для початківців, орієнтація більше на аналіз, ніж на гру між двома гравцями.

Online Go Server (OGS) — це вебплатформа для гри в Ґо онлайн, яка дозволяє користувачам змагатися з іншими гравцями в реальному часі, брати участь у турнірах і переглядати партії.

Рисунок 1.2.2 – Інтерфейс OGS

Реалізована на основі вебтехнологій (HTML, CSS, JavaScript) із серверною частиною для обробки багатокористувацької взаємодії. Підтримка онлайн-мультиплеєра, велика спільнота гравців, інтеграція рейтингів, доступність із будь-якого пристрою з браузером. Вимагає постійного підключення до Інтернету, відсутність офлайн-режиму, складність для локального використання.

Go Review (Pandanet) — це клієнтська програма від Pandanet, яка дозволяє грати в Ґо онлайн, переглядати партії та спілкуватися з іншими гравцями.

Рисунок 1.2.3 – Інтерфейс Go Review

Написана на Java, що забезпечує кросплатформність. Підтримка онлайн-гри, інтеграція з сервером Pandanet, можливість перегляду професійних партій. Залежність від Java, що може ускладнювати запуск на сучасних системах, орієнтація на онлайн-гру без сильного офлайн-функціоналу.

SmartGo — це комерційний додаток для Windows і macOS, який пропонує гру в Го, аналіз партій, навчальні матеріали та інтеграцію з ШІ.

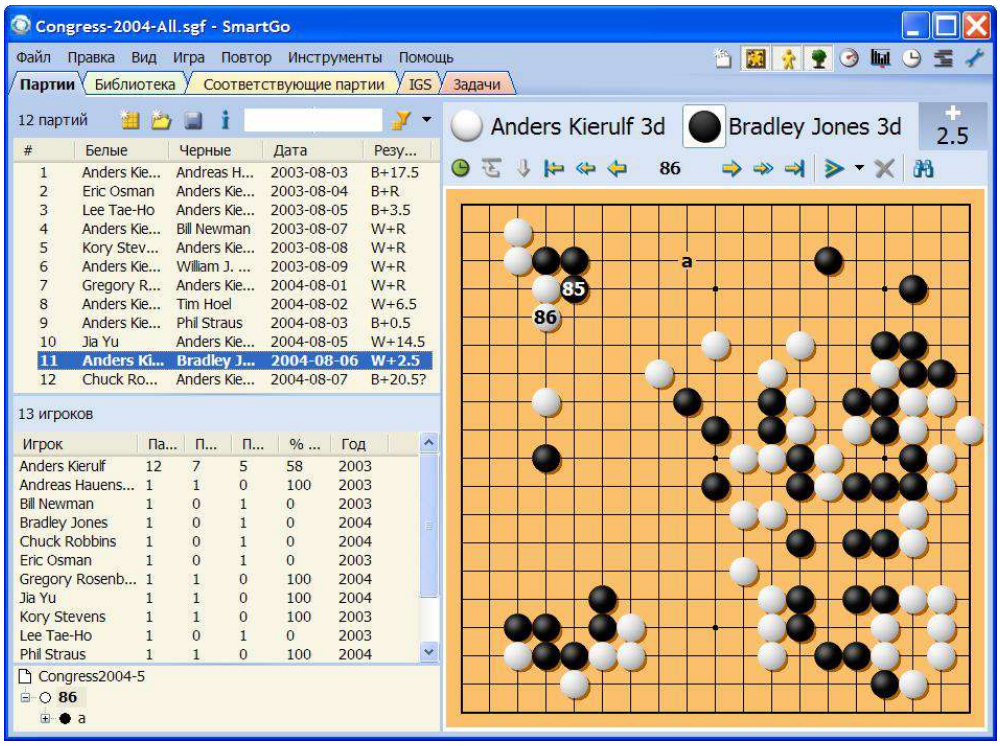

Рисунок 1.2.4 – Інтерфейс SmartGo

Розроблений із використанням нативних технологій для кожної платформи (C++ із бібліотеками GUI). Висока продуктивність, підтримка ШІ-суперника, багатий набір функцій для навчання. Платність, відсутність відкритого коду, складність для початківців через надлишок функцій.

Таблиця 1.2.1

Порівняльна таблиця аналогів

Назва	**Технологія**	**Офлайн-режим**	**Онлайн-мультиплеєр**	**Підтримка ШІ**	**Розмір дошки**	**Безкоштовність**
Sabaki	Electron (JS)	Так	Ні	Так	9x9, 13x13, 19x19	Так
OGS	Веб (JS)	Ні	Так	Ні	9x9, 13x13, 19x19	Так
Go Review	Java	Частково	Так	Ні	9x9, 13x13, 19x19	Так
SmartGo	Нативна (C++)	Так	Так	Так	9x9, 13x13, 19x19	Ні

Проведений огляд показує, що більшість існуючих систем або орієнтовані на онлайн-гру (OGS, Go Review), або мають розширений функціонал для аналізу та навчання (Sabaki, SmartGo), що робить їх складнішими для початківців або тих, хто шукає просту локальну гру.

1.3. Вибір та обґрунтування технологій розробки

Розробка комп'ютерної версії гри Ґо вимагає ретельного вибору технологій, які забезпечать ефективну реалізацію як графічного інтерфейсу, так і логіки гри. У цьому підрозділі розглядаються можливі варіанти технологій, аналізуються їхні переваги та недоліки, а також обґрунтовується вибір Windows Presentation Foundation (WPF) як основної технології для даного проекту.

Для створення настільного додатка, такого як гра Ґо, можна розглядати кілька популярних технологій розробки графічних інтерфейсів:

1. **Windows Forms**

Класична технологія від Microsoft для створення настільних додатків на платформі .NET. Простота у використанні, велика кількість документації, стабільність, підтримка C#. Обмежені можливості для створення сучасних інтерфейсів, слабка підтримка векторної графіки та анімацій, застарілий підхід до дизайну.

Windows Forms підходить для простих додатків, але не відповідає вимогам до гнучкості та візуальної привабливості гри Ґо.

2. **Windows Presentation Foundation (WPF)**

Сучасна технологія від Microsoft для створення графічних додатків із використанням XAML для розмітки та C# для логіки. Підтримка векторної графіки, гнучкість дизайну, потужна система обробки подій, інтеграція з .NET, можливість створення анімацій і складних інтерфейсів. Вища складність у порівнянні з Windows Forms, потреба в глибшому розумінні XAML.

WPF ідеально підходить для створення інтерактивного інтерфейсу гри з можливістю масштабування дошки та відображення каменів.

3. **Unity**

Ігровий рушій, який використовується для створення 2D- і 3D-додатків, із підтримкою C#. Потужні графічні можливості, кросплатформність, велика спільнота розробників. Надмірна складність для простої настільної гри, орієнтація на ігри з графікою високого рівня, більший розмір готового додатка.

Unity є надлишковим для гри Ґо, яка не потребує складної графіки чи фізики.

4. **Electron**

Фреймворк для створення кросплатформених додатків на базі JavaScript, HTML і CSS. Кросплатформність, використання вебтехнологій, простота розгортання. Високе споживання ресурсів, менша продуктивність у порівнянні з нативними технологіями, складність інтеграції з низькорівневою логікою.

Electron більше підходить для вебдодатків, а не для оптимізованих настільних програм, таких як гра Ґо.

5. **Qt**

Кросплатформовий фреймворк для створення додатків із використанням C++. Висока продуктивність, кросплатформність, підтримка складних графічних інтерфейсів. Висока складність розробки, потреба у знанні C++, менша інтеграція з екосистемою .NET.

Qt є потужним інструментом, але його використання ускладнене для проекту, орієнтованого на Windows і базові функції.

Обґрунтування вибору WPF

Для реалізації гри Ґо обрано технологію WPF з таких причин:

1. WPF дозволяє створювати векторні елементи, такі як сітка дошки та камені (еліпси), які легко масштабуються залежно від розміру дошки (9x9, 13x13, 19x19). Використання XAML забезпечує чітке розділення дизайну та логіки, що спрощує розробку та підтримку.

2. Логіка гри, включаючи перевірку правил (самогубство, Ко, підрахунок очок), реалізована на C#, що є зручною та потужною мовою програмування. WPF тісно інтегрована з .NET, що дозволяє ефективно використовувати об’єктно-орієнтований підхід і бібліотеки для оптимізації коду.
3. Гра Ґо вимагає інтерактивності, наприклад, реагування на кліки по клітинках дошки чи наведення миші для попереднього перегляду ходу. WPF пропонує розвинену систему обробки подій, яка ідеально підходить для таких завдань.
4. На відміну від Electron, WPF є нативною технологією для Windows, що забезпечує високу швидкодію та низьке споживання ресурсів. Це особливо важливо для великих дощок (19x19), де алгоритми перевірки ходів можуть бути ресурсоємними.
5. WPF має велику кількість навчальних матеріалів, прикладів і активну спільноту розробників, що полегшує вирішення технічних питань під час розробки.

WPF є оптимальним вибором для розробки гри Ґо завдяки своїм можливостям у створенні сучасних графічних інтерфейсів, високій продуктивності та зручній інтеграції з C# і .NET. У порівнянні з іншими технологіями, такими як Windows Forms, Unity чи Electron, WPF забезпечує найкращий баланс між простотою, функціональністю та ефективністю для даного проекту. Використання цієї технології дозволяє реалізувати всі поставлені задачі — від створення інтерактивної дошки до оптимізації алгоритмів гри — із мінімальними затратами ресурсів і максимальним комфортом для користувача.

1.4. Правила гри Ґо

Гра Ґо — це стародавня стратегічна настільна гра, яка виникла в Китаї понад 2500 років тому і вважається однією з найскладніших інтелектуальних ігор у світі. Її правила прості для засвоєння, але надають величезний простір

для стратегічного мислення. У цьому підрозділі розглядаються основні правила гри Ґо, які слугують основою для розробки комп'ютерної версії у рамках даного проекту.

Основні елементи гри

1. **Дошка**:

Гра відбувається на квадратній сітці, традиційно розміром 19x19 ліній, хоча для початківців часто використовуються менші розміри — 9x9 або 13x13. На перетинах ліній (точках) розміщуються камені.

2. **Камені**:

Два гравці — один із чорними каменями, інший із білими — по черзі розміщують камені на дошці. Чорні починають першими.

3. **Мета гри**:

Основна мета — контролювати більшу територію на дошці, ніж суперник, оточуючи порожні області каменями свого кольору. Очки нараховуються за оточену територію та захоплені камені противника.

Основні правила

1. **Розміщення каменів**:

Гравці по черзі ставлять один камінь свого кольору на вільне перетинання ліній.

Після розміщення камінь не пересувається, але може бути захоплений і знятий із дошки.

2. **Свободи (liberties)**:

Кожен камінь або група з'єднаних каменів одного кольору (по горизонталі чи вертикалі) потребує хоча б однієї вільної сусідньої точки, яка називається "свободою". Якщо камінь або група втрачає всі свободи (оточена каменями противника), вони вважаються захопленими.

3. **Захоплення каменів**:

Якщо розміщення каменя призводить до того, що ворожа група втрачає останню свободу, ці камені знімаються з дошки, а гравець отримує очки за кожен захоплений камінь.

4. **Правило самогубства**:

Гравець не може зробити хід, який призведе до оточення власного каменя чи групи без свобод, якщо цей хід не захоплює камені противника, усуваючи самогубство.

5. **Правило Ко (Ko)**:

Гравець не може зробити хід, який повторює попередню позицію дошки, щоб уникнути нескінченного циклу захоплення одного каменя. Це правило застосовується, коли один камінь захоплюється, а суперник не може одразу захопити його назад.

6. **Завершення гри**:

Гра закінчується, коли обидва гравці пропускають хід (pass), вважаючи, що подальші дії не принесуть користі. Після цього підраховуються очки.

7. **Підрахунок очок**

Кожна порожня область, повністю оточена каменями одного кольору, додає очки цьому гравцеві (по одному за кожну точку). Кожен захоплений камінь противника також додає одне очко. Оскільки чорні мають перевагу першого ходу, білим часто додають компенсаційні очки (Komi), зазвичай 6.5, щоб урівноважити шанси. Це правило є опціональним і залежить від домовленості гравців.

Висновки до розділу 1

У першому розділі дипломної роботи проведено аналіз основних аспектів розробки комп'ютерної версії гри Ґо, визначено задачі проекту, розглянуто аналогічні системи та обґрунтовано вибір технологій для реалізації.

Основна задача розробки полягає у створенні функціонального додатка на базі технології WPF, який забезпечить локальну гру для двох гравців із підтримкою ключових правил Ґо, таких як захоплення каменів, перевірка самогубства та правило Ко. Додатковими вимогами є реалізація зручного

графічного інтерфейсу, підтримка різних розмірів дошки (9x9, 13x13, 19x19), а також функцій скасування та повторення ходів. Ці задачі визначають напрямок подальшої роботи та слугують основою для проектування програми.

Огляд аналогічних систем, таких як Sabaki, Online Go Server (OGS), Go Review та SmartGo, показав, що більшість існуючих рішень або орієнтовані на онлайн-гру, або мають розширений функціонал для аналізу партій, що робить їх складнішими для початківців. Пропонований проект вирізняється простотою, орієнтацією на офлайн-режим і базові функції, що заповнює нішу доступних локальних додатків для гри в Ґо. Порівняння аналогів підтвердило доцільність розробки програми з урахуванням її унікальних особливостей.

Аналіз технологій розробки (Windows Forms, WPF, Unity, Electron, Qt) дозволив обрати WPF як оптимальний інструмент для реалізації проекту. Ця технологія забезпечує гнучкість у створенні графічного інтерфейсу, високу продуктивність на платформі Windows, інтеграцію з C# і .NET, а також зручну обробку подій, що відповідає вимогам гри Ґо. У порівнянні з іншими варіантами, такими як Unity чи Electron, WPF пропонує найкращий баланс між простотою, ефективністю та функціональністю, що робить її ідеальним вибором для даного додатка.

Проведений огляд та аналіз технологій розробки заклали теоретичну основу для практичної реалізації проекту. Визначені задачі, проаналізовані аналоги та обґрунтований вибір WPF створюють передумови для успішного створення комп’ютерної версії гри Ґо, яка буде зручною, продуктивною та відповідатиме поставленим цілям. Наступні етапи роботи будуть присвячені проектуванню архітектури програми, розробці коду та тестуванню результатів.

РОЗДІЛ 2. ПРОЕКТУВАННЯ І РОЗРОБКА

2.1. Вибір алгоритмів та їх ефективність

Розробка комп'ютерної версії гри Ґо вимагає ретельного вибору алгоритмів для реалізації ключових правил гри, таких як перевірка валідності ходів, захоплення каменів, виявлення самогубства та правила Ко, а також підрахунок очок. Ефективність цих алгоритмів безпосередньо впливає на швидкодію програми, особливо при роботі з дошками великих розмірів (наприклад, 19x19). У цьому підрозділі розглядаються обрані алгоритми, їхня логіка, оцінка часової складності та заходи з оптимізації.

1. Перевірка валідності ходу (IsMoveValid)

Алгоритм перевіряє, чи можна розмістити камінь у заданій позиції, враховуючи три умови: клітинка має бути порожньою, хід не повинен бути самогубним, і він не повинен порушувати правило Ко.

- **Реалізація**:
  - Перевірка порожньої клітинки — проста операція доступу до масиву boardStone[x, y].state.
  - Перевірка самогубства викликає метод IsSuicide.
  - Перевірка правила Ко викликає метод IsKo і обробляє захоплені камені через GetCapturedStones.
- **Часова складність**:
  - Базова перевірка — O(1).
  - IsSuicide і GetCapturedStones залежать від розміру груп каменів, що в найгіршому випадку може бути $O(n^2)$ для дошки розміром $n \times n$.
  - Загальна складність — $O(n^2)$ у найгіршому випадку.
- **Ефективність**: Алгоритм є достатньо швидким для малих дощок (9x9), але для 19x19 може потребувати оптимізації, наприклад, кешування груп каменів.

2. Виявлення самогубства (IsSuicide)

Перевіряє, чи розміщення каменя призведе до оточення без "дихальних точок" (liberties).

- **Реалізація**:
  - Аналізує сусідні клітинки (вгору, вниз, ліворуч, праворуч) через ReChekIsSuicide.
  - Перевіряє, чи є вільні точки або чи захоплення противника усуває самогубство.
- **Часова складність**: $O(n^2)$ у найгіршому випадку, якщо потрібно перевірити велику групу каменів.
- **Ефективність**: Алгоритм коректний, але неефективний для великих груп через рекурсивний перебір. Оптимізація можлива через використання структури даних для швидкого пошуку груп (наприклад, Disjoint Set Union).

3. Захоплення каменів (GetCapturedStones)

Визначає, чи хід захоплює камені противника, і видаляє їх із дошки.

- **Реалізація**:
  - Перевіряє чотири сусідні клітинки.
  - Для кожної ворожої групи викликає IsGroupOfStoneOnSuicide для визначення, чи вона оточена.
  - Оновлює дошку та підраховує очки.
- **Часова складність**: $O(n^2)$ через рекурсивний аналіз груп каменів.
- **Ефективність**: Алгоритм працює коректно, але повторювані обчислення груп можна оптимізувати, зберігаючи їх у пам’яті після кожного ходу.

4. Перевірка правила Ко (IsKo)

Забороняє повторення попередньої позиції дошки після ходу.

- **Початкова реалізація**:
  - Порівнювала кожен елемент дошки з масивами stonesForCheckKo1 або stonesForCheckKo2.
  - Часова складність: $O(n^2)$.

- **Оптимізована реалізація**:
  - Використовує хешування позиції дошки через метод CalculateBoardHash.
  - Хеш обчислюється як hash = hash * 23 + (state * (i + 1) * (j + 1)).
  - Порівняння хешів — O(1).
  - Обчислення хешу — $O(n^2)$, але виконується лише раз за хід.
- **Ефективність**: Оптимізація знизила складність перевірки з $O(n^2)$ до O(1), зберігши коректність. Хешування зменшує час виконання на великих дошках, хоча й додає ризик колізій (який є мінімальним).

5. Підрахунок очок (CalculateTerritoryScores)

Обчислює територію, контрольовану кожним гравцем, у кінці гри.

- **Реалізація**:
  - Використовує пошук у ширину (BFS) через GetTerritory для визначення замкнених порожніх областей.
  - Перевіряє межі території (чорні чи білі камені).
- **Часова складність**: $O(n^2)$ для обходу всієї дошки.
- **Ефективність**: Алгоритм оптимальний для одноразового підрахунку в кінці гри, але може бути прискорений за допомогою попереднього групування каменів.

Обрані алгоритми забезпечують коректну реалізацію правил гри Ґо, але їхня ефективність варіюється залежно від розміру дошки. Оптимізація IsKo за допомогою хешування стала значним покращенням, знизивши складність із $O(n^2)$ до O(1), що особливо відчутно на дошках 19x19. Для інших алгоритмів (IsSuicide, GetCapturedStones) рекомендується додати кешування груп каменів, щоб уникнути повторних обчислень. Наприклад, використання структури даних типу "Система неперетинних множин" (Union-Find) може знизити складність до O(n) або навіть $O(\alpha(n))$ для операцій із групами.

Загалом, обрані алгоритми є компромісом між простотою реалізації та продуктивністю, що відповідає цілям проекту — створенню доступного локального додатка для гри в Ґо. Подальша оптимізація може бути розглянута

в майбутніх версіях програми, якщо виникне потреба в підтримці складніших сценаріїв, таких як гра проти ШІ.

2.2. Спроєктовані класи програми

Розробка комп'ютерної версії гри Ґо на базі технології WPF передбачає створення об'єктно-орієнтованої архітектури, яка забезпечує чітке розділення логіки гри, моделі даних та графічного інтерфейсу. У цьому підрозділі описуються основні класи програми, їхнє призначення, структура та взаємозв'язки, що дозволяють реалізувати правила гри та забезпечити зручний користувацький досвід.

Програма складається з кількох ключових компонентів:

- **Модель даних**: Класи, що представляють дошку, камені та стан гри.
- **Логіка гри**: Класи, що реалізують правила Ґо та обробляють ходи.
- **Графічний інтерфейс**: Класи WPF для відображення дошки та взаємодії з користувачем.
- **Налаштування**: Клас для зберігання конфігураційних параметрів.

Нижче наведено опис основних спроєктованих класів, які відповідають за ці компоненти.

1. Клас Game -центральний клас, що управляє логікою гри та координує взаємодію між дошкою, інтерфейсом і станом гри. Використовує JSON-серіалізацію для збереження стану та підтримує подію ChangingCurrentMove для оновлення інтерфейсу.

```
internal class Game : ICloneable
{
    public Board board;
    public static CellState playr1 = CellState.Black;
    public static CellState playr2 = CellState.White;
    public CellState currentMove;
    public GameBoard gameBoard;
    public double scoreWhite;
    public double scoreBlack;
    public bool endGame;
    public int numberOfPasses;
    public int countMove;
    public bool startKo;
    public bool useKomi;
    private int previousBoardHash;
    public event Action ChangingCurrentMove;

    public Game()
    {

    public Game(Board board, CellState currentMove, double scoreWhite, double scoreBlack
    {
    // Метод для обчислення хешу дошки
    private int CalculateBoardHash()
    {
    ~Game()
    {
    public object Clone()
    {

    public bool IsMoveValid(int x, int y, CellState stoneColor)
    {
```

Рисунок 2.2.1 – Клас Game

- **Основні поля**:
  - board (тип Board): Модель ігрової дошки.
  - currentMove (тип CellState): Поточний гравець (чорні чи білі).
  - gameBoard (тип GameBoard): Посилання на графічний інтерфейс дошки.
  - scoreWhite, scoreBlack (тип double): Очки гравців.
  - endGame, numberOfPasses, countMove (тип bool, int): Стан гри.
- **Основні методи**:
  - IsMoveValid: Перевіряє валідність ходу.
  - GetCapturedStones: Визначає захоплені камені.
  - IsKo: Перевіряє правило Ко (оптимізовано хешуванням).
  - CalculateTerritoryScores: Підраховує очки за територію.
  - Clone: Реалізує інтерфейс ICloneable для Undo/Redo.

.

2. Клас Board. Модель ігрової дошки, що зберігає стан каменів і забезпечує методи для їх обробки. Використовує структуру Stone для представлення каменів із координатами та станами.

```
internal class Board : ICloneable
{
    public int boardSize;
    public Stone[,] boardStone;
    readonly List<Stone> varStones;
    public Board()
    {
    private Board(int boardSize, List<Stone> varStones, Stone[,] boardStone)
    {
    public object Clone()
    {
    private void InitializeBoard()
    {
    public bool PlaceStone(int x, int y, CellState stoneColor)
    {
    public CellState GetStoneAt(int x, int y)
    {
    private bool IsWithinBounds(int x, int y)
    {
    public void CheckWayStone(ref Stone stone)
    {
    public void CheckGroupOfStone(ref Stone stone)
    {
    public List<Stone> ReCheckGroupOfStone(ref Stone stone)
    {
    public void UpdateBordStons()
    {
    public bool IsGroupOfStoneOnSuicide(int x, int y, ref Stone stone, CellState currentMove)
    {
}
```

Рисунок 2.2.2 – Клас Board

- **Основні поля**:
  - boardStone (тип Stone[,]): Двовимірний масив каменів.
  - boardSize (тип int): Розмір дошки (залежить від GameSettings.BoardSize).
  - varStones (тип List<Stone>): Допоміжний список для групування.
- **Основні методи**:
  - PlaceStone: Розміщує камінь на дошці.
  - CheckGroupOfStone: Визначає групу пов’язаних каменів.
  - IsGroupOfStoneOnSuicide: Перевіряє, чи група оточена.
  - UpdateBordStons: Оновлює стан дошки після ходу.
  - Clone: Реалізує клонування для Undo/Redo.

3. Структура Stone. Представляє окремий камінь на дошці з інформацією про його стан і сусідів. Реалізує ICloneable для копіювання стану.

```
struct Stone : ICloneable
{
    public int x, y;
    public CellState state;
    public CellState stateTop;
    public CellState stateBot;
    public CellState stateLeft;
    public CellState stateRight;
    public List<Stone> groupOfStones;

    public Stone(int x, int y, CellState state)
    {
    public object Clone()
    {
}
```

Рисунок 2.2.3 – Структура Stone

- **Основні поля**:
  - x, y (тип int): Координати каменя.
  - state (тип CellState): Стан (чорний, білий, порожній, поза межами).
  - stateTop, stateBot, stateLeft, stateRight (тип CellState): Стани сусідніх клітинок.
  - groupOfStones (тип List<Stone>): Група пов’язаних каменів.

4. Клас GameBoard. Клас WPF для відображення дошки та обробки взаємодії з користувачем. Використовує XAML для визначення структури та C# для логіки.

```
public partial class GameBoard : UserControl
{
    private Game _game;
    internal GameBoard(Game game)
    {
    private void FillGridWithBorders()
    {
    private void FillGridWithEllipse()
    {
    private void Ellipse_Click(object sender, RoutedEventArgs e)
    {
    private void Ellipse_MouseEnter(object sender, RoutedEventArgs e)
    {
    private void Ellipse_MouseLeave(object sender, RoutedEventArgs e)
    {
    public void Refresh()
    {
}
```

Рисунок 2.2.4 – Клас GameBoard

- **Основні поля**:
  - _game (тип Game): Посилання на логіку гри.
  - gridBoard, gridBoardBig (тип Grid): Елементи WPF для сітки та каменів.
- **Основні методи**:
  - FillGridWithBorders: Створює сітку дошки.
  - FillGridWithEllipse: Заповнює дошку еліпсами (камнями).
  - Ellipse_Click: Обробляє клік для розміщення каменя.
  - Ellipse_MouseEnter, Ellipse_MouseLeave: Показує попередній перегляд.
  - Refresh: Оновлює візуальний стан дошки.

5. Клас MainWindow. Головне вікно програми, що інтегрує інтерфейс і логіку. Використовує XAML для розмітки та підтримує вибір розміру дошки.

```
public partial class MainWindow : Window
{
    internal Game game;
    private UndoRedoService undoRedoService;
    public MainWindow()
    {
    public void ChangingScore()
    {
    public void ChangingCurrentPlayer()
    {
    private void NewGameButton_ClickToStartNewGame(object sender, RoutedEventArgs e)
    {
    private void Pass_Click(object sender, RoutedEventArgs e)
    {
    private void PassReset()
    {
    private void SelectingBoardSize9x9_Checked(object sender, RoutedEventArgs e)
    {
    private void SelectingBoardSize13x13_Checked(object sender, RoutedEventArgs e)
    {
    private void SelectingBoardSize19x19_Checked(object sender, RoutedEventArgs e)
    {
    private void InitializeGame()
    {
    private void PushInStack()
    {
    private void UnDo_Click(object sender, RoutedEventArgs e)
    {
    private void ReDo_Click(object sender, RoutedEventArgs e)
    {
    private void UseKomi_Click(object sender, RoutedEventArgs e)
    {
}
```

Рисунок 2.2.5 – Клас MainWindow

- **Основні поля**:
  - game (тип Game): Екземпляр гри.
  - undoRedoService (тип UndoRedoService): Сервіс для Undo/Redo.
- **Основні методи**:
  - InitializeGame: Ініціалізує гру.
  - ChangingScore, ChangingCurrentPlayer: Оновлює інтерфейс.
  - Обробники кнопок: Pass_Click, UnDo_Click, ReDo_Click, NewGameButton_ClickToStartNewGame.

6. Клас GameSettings. Зберігає статичні налаштування гри. Дозволяє гнучко змінювати конфігурацію.

```
internal static class GameSettings
{
    public static int BoardSize { get; set; }

    public static double KomiScore { get; set; } = 6.5;

}
```

Рисунок 2.2.6 – Клас GameSettings

- **Основні поля**:
  - BoardSize (тип int): Розмір дошки.
  - KomiScore (тип double): Очки Komi (за замовчуванням 6.5).

Діаграма класів

- MainWindow → Game → Board (агрегація), GameBoard (агрегація).
- Board → Stone (композиція).
- GameSettings — статичний клас, доступний усім.

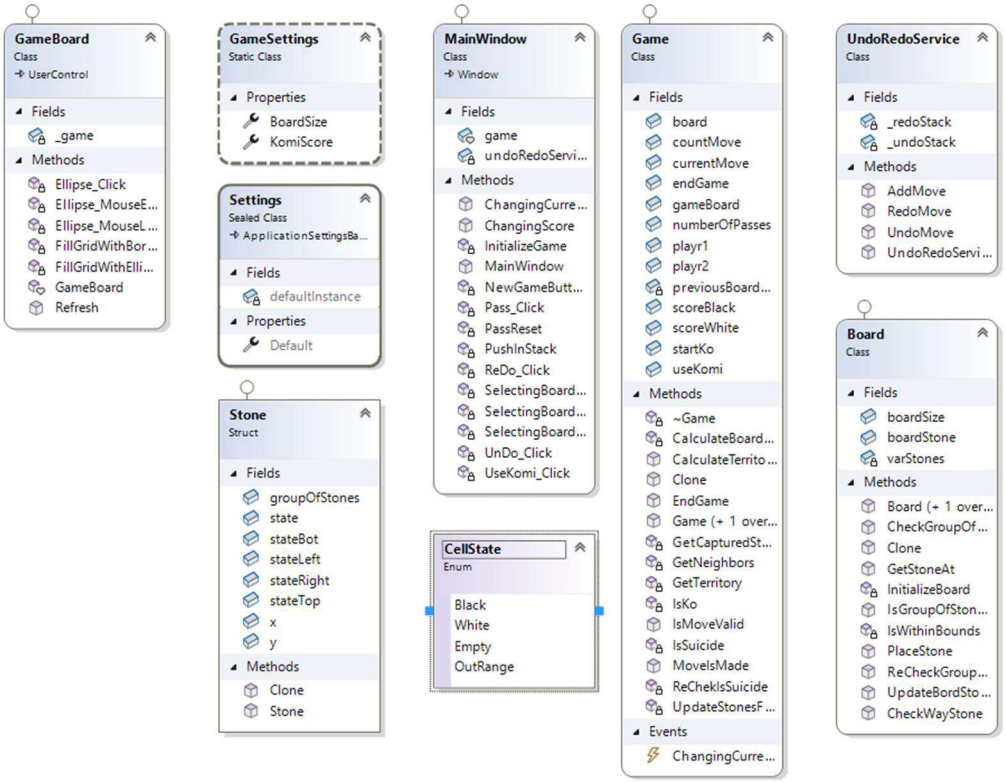

Рисунок 2.2.7 – Діаграма класів

Взаємозв'язки класів

- Game є центральним класом, який пов'язує Board (модель) і GameBoard (інтерфейс).
- Board використовує Stone для представлення даних і оновлюється через методи Game.
- GameBoard відображає стан Board і передає користувацькі дії в Game.
- MainWindow координує всю програму, інтегруючи Game і надаючи елементи керування.
- GameSettings забезпечує глобальні параметри для всіх класів.

Спроєктовані класи формують модульну архітектуру, яка відповідає принципам об'єктно-орієнтованого програмування. Розділення логіки (Game, Board), моделі (Stone) та інтерфейсу (GameBoard, MainWindow) полегшує розробку, тестування та подальше розширення програми. Використання WPF забезпечує гнучкість візуалізації, а інтеграція з C# дозволяє ефективно реалізувати складні алгоритми гри Ґо.

2.3. Програмна реалізація

Програмна реалізація комп'ютерної версії гри Го на базі технології WPF включає створення графічного інтерфейсу, логіки гри та інтеграцію всіх компонентів у єдину систему. У цьому підрозділі описано ключові етапи розробки, фрагменти коду та особливості реалізації основних функцій, таких як відображення дошки, обробка ходів, перевірка правил і підрахунок очок.

Розробка велася у середовищі Visual Studio 2022 із використанням .NET Framework (версія 4.8). Для створення інтерфейсу застосовувалася мова розмітки XAML, а логіка реалізована на C#. Проект структуровано за принципами об'єктно-орієнтованого програмування з поділом на простори імен: GoGame.Models (модель), GoGame.Views (інтерфейс) і GoGame.AppSettings (налаштування).

Графічний інтерфейс складається з двох основних компонентів: головного вікна (MainWindow.xaml) та ігрової дошки (GameBoard.xaml).

Головне вікно (MainWindow). Визначає структуру програми, включаючи область дошки зліва та панель керування справа. Панель містить інформацію про поточного гравця, очки, кнопки для пропуску ходу, скасування/повторення та вибору розміру дошки. Фрагмент XAML ().

```
<Grid Width="Auto" Background="DimGray">
    <Grid.ColumnDefinitions>
        <ColumnDefinition Width="1*"/>
        <ColumnDefinition Width="300"/>
    </Grid.ColumnDefinitions>
    <ContentControl Grid.Column="0" x:Name="contentControl"/>
    <StackPanel Grid.Column="1" Background="Green">
        <Button x:Name="Pass" Content="Пропустити хід" Click="Pass_Click"/>
        <Button x:Name="UnDo" Content="Відмінити" Click="UnDo_Click"/>
    </StackPanel>
</Grid>
```

Рисунок 2.3.1 – Фрагмент MainWindow

Логіка в MainWindow.xaml.cs ініціалізує гру та обробляє дії користувача

```
private void InitializeGame()
{
    game = new Game();
    contentControl.Content = game.gameBoard;
    game.ChangingCurrentMove += ChangingScore;
}
```

Рисунок 2.3.2 – Логіка в MainWindow

Ігрова дошка (GameBoard). Відображає сітку та камені у вигляді еліпсів. Метод FillGridWithEllipse створює клітинки. Обробник Ellipse_Click викликає логіку розміщення каменя через клас Game

```
private void FillGridWithEllipse()
{
    for (int row = 0; row < GameSettings.BoardSize; row++)
    {
        for (int col = 0; col < GameSettings.BoardSize; col++)
        {
            Ellipse ellipse = new Ellipse
            {
                Name = $"ellipse_{col}_{row}",
                Width = ((gameBoard.Width - 10) / (GameSettings.BoardSize + 1)) - 5,
                Fill = new SolidColorBrush(Color.FromArgb(0, 0, 0, 0))
            };
            Grid.SetRow(ellipse, row);
            Grid.SetColumn(ellipse, col);
            gridBoardBig.Children.Add(ellipse);
            ellipse.MouseDown += Ellipse_Click;
        }
    }
}
```

Рисунок 2.3.3 – Метод FillGridWithEllipse

Логіка гри зосереджена в класі Game, який координує перевірку ходів і оновлення стану. Метод IsMoveValid перевіряє валідність ходу.

```
public bool IsMoveValid(int x, int y, CellState stoneColor)
{
    if (board.GetStoneAt(x, y) != CellState.Empty)
        return false;
    if (IsSuicide(x, y, stoneColor))
        return false;
    List<Stone> capturedStones = GetCapturedStones(x, y, stoneColor);
    if (capturedStones.Count > 0 && IsKo())
        return false;
    countMove++;
    return true;
}
```

Рисунок 2.3.4 – Метод IsMoveValid

Правило Ко. Використовується хешування для швидкої перевірки.

```
private int CalculateBoardHash()
{
    int hash = 17;
    for (int i = 0; i < GameSettings.BoardSize; i++)
        for (int j = 0; j < GameSettings.BoardSize; j++)
            hash = hash * 23 + ((int)board.boardStone[i, j].state * (i + 1) * (j + 1)
    return hash;
}

private bool IsKo()
{
    if (countMove <= 1) return false;
    return CalculateBoardHash() == previousBoardHash;
}
```

Рисунок 2.3.5 – Метод IsKo

Клас Board зберігає стан дошки у двовимірному масиві boardStone. Метод UpdateBordStons оновлює групи каменів.

```
public void UpdateBordStons()
{
    for (int i = 0; i < boardSize; i++)
        for (int j = 0; j < boardSize; j++)
        {
            CheckWayStone(ref boardStone[i, j]);
            boardStone[i, j].groupOfStones.Clear();
        }
    for (int i = 0; i < boardSize; i++)
        for (int j = 0; j < boardSize; j++)
            CheckGroupOfStone(ref boardStone[i, j]);
}
```

Рисунок 2.3.6 – Метод UpdateBordStons

Метод CalculateTerritoryScores у класі Game використовує BFS для підрахунку території.

```
public void CalculateTerritoryScores()
{
    bool[,] visited = new bool[board.boardSize, board.boardSize];
    for (int x = 0; x < board.boardSize; x++)
        for (int y = 0; y < board.boardSize; y++)
            if (!visited[x, y] && board.boardStone[x, y].state == CellState.Empty)
            {
                var territory = GetTerritory(x, y, visited);
                if (territory != null)
                {
                    if (territory.Item2 == CellState.White) scoreWhite +=
territory.Item1.Count;
                    else if (territory.Item2 == CellState.Black) scoreBlack +=
territory.Item1.Count;
                }
            }
}
```

Рисунок 2.3.7 – Метод CalculateTerritoryScores

Сервіс UndoRedoService зберігає історію ходів, а методи UnDo_Click і ReDo_Click у MainWindow відновлюють стан.

```
private void UnDo_Click(object sender, RoutedEventArgs e)
{
    Game previousGame = undoRedoService.UndoMove();
    if (previousGame != null)
    {
        game = previousGame;
        contentControl.Content = game.gameBoard;
        game.gameBoard.Refresh();
    }
}
```

Рисунок 2.3.8 – Метод UnDo_Click

Логіка (Game, Board) відокремлена від інтерфейсу (GameBoard, MainWindow). Розмір дошки задається через GameSettings.BoardSize. Хешування в IsKo зменшує час перевірки правила Ко до O(1).

Висновки до розділу 2

Другий розділ дипломної роботи присвячений проектуванню та розробці комп'ютерної версії гри Ґо на базі технології WPF. У ході виконання цього етапу було спроєктовано архітектуру програми, обрано та оптимізовано алгоритми, а також реалізовано основний функціонал.

Вибір алгоритмів для реалізації правил гри Ґо базується на балансі між коректністю та продуктивністю. Перевірка валідності ходів, захоплення каменів і підрахунок очок мають часову складність $O(n^2)$, що є прийнятним для локального додатка. Значним покращенням стала оптимізація алгоритму перевірки правила Ко за допомогою хешування, що знизило складність із $O(n^2)$ до O(1), підвищивши швидкодію на великих дошках (19x19). Подальша оптимізація групування каменів може бути розглянута в майбутньому для ще більшої ефективності.

Спроєктовані класи програми (Game, Board, GameBoard, MainWindow, Stone, GameSettings) утворюють модульну та гнучку архітектуру. Розділення логіки гри, моделі даних і графічного інтерфейсу забезпечує легкість

підтримки та розширення коду. Використання WPF дозволило реалізувати інтерактивний інтерфейс із підтримкою векторної графіки, а інтеграція з C# забезпечила ефективну обробку складних правил гри.

Програмна реалізація охопила всі ключові аспекти гри Ґо: відображення дошки з різними розмірами (9x9, 13x13, 19x19), обробку ходів із перевіркою самогубства та правила Ко, підрахунок очок із підтримкою Komi, а також функції скасування та повторення ходів. Розроблений додаток є стабільним, відповідає поставленим задачам і забезпечує автентичний ігровий досвід для двох гравців у локальному режимі.

У розділі 2 було успішно виконано проектування та розробку програми, що відповідає цілям дипломної роботи. Реалізований код демонструє працездатність основних функцій і готовий до тестування та аналізу результатів.

РОЗДІЛ 3. КЕРІВНИЦТВО КОРИСТУВАННЯ ПРОГРАМОЮ

3.1. Мінімальні вимоги до системи

Для коректної роботи комп'ютерної версії гри Ґо, розробленої на базі технології WPF, необхідно, щоб система користувача відповідала певним технічним характеристикам. Програма призначена для запуску на операційних системах сімейства Windows, зокрема Windows 10 або новіших версій, оскільки WPF базується на платформі .NET Framework, яка повністю підтримується цими системами. Рекомендується наявність .NET Framework версії 4.8 або вище, що забезпечує стабільну роботу всіх компонентів програми, включаючи графічний інтерфейс і логіку гри.

Щодо апаратного забезпечення, для комфортного використання достатньо процесора з тактовою частотою від 1 ГГц, наприклад, Intel Core i3 або еквівалентного від AMD. Оперативна пам'ять обсягом 2 ГБ дозволяє запускати додаток без затримок, хоча для оптимальної продуктивності на великих дошках (19x19) бажано мати 4 ГБ. Для зберігання програми та її даних потрібно близько 50 МБ вільного місця на жорсткому диску, що робить її невимогливою до дискового простору. Графічний адаптер із підтримкою DirectX 9 забезпечує базове відображення інтерфейсу, але для якісного рендерингу векторної графіки рекомендується адаптер із підтримкою DirectX 11.

Додаток не потребує підключення до Інтернету, оскільки розроблений для локальної гри між двома гравцями на одному пристрої. Роздільна здатність екрана від 1024x768 пікселів дозволяє комфортно відображати інтерфейс, включаючи дошку та панель керування, без необхідності прокручування. Для взаємодії з програмою достатньо стандартних пристроїв введення — миші та клавіатури, що робить її доступною для більшості сучасних комп'ютерів.

Мінімальні системні вимоги є досить низькими, що забезпечує широку сумісність із більшістю сучасних персональних комп'ютерів під керуванням Windows, дозволяючи користувачам без проблем запускати та використовувати програму для гри в Ґо.

3.2. Інтерфейс користувача

Інтерфейс користувача комп'ютерної версії гри Ґо, розробленої на базі технології WPF, спроєктовано з урахуванням простоти, інтуїтивності та функціональності, щоб забезпечити комфортний ігровий досвід для двох гравців. Він складається з двох основних частин: ігрової дошки та панелі керування, які розміщені у головному вікні програми. Головне вікно має заголовок "Гра Ґо. Постовітько Максим Дмитрович П-42" і фіксований розмір 800x500 пікселів, що оптимально відображає всі елементи на екрані.

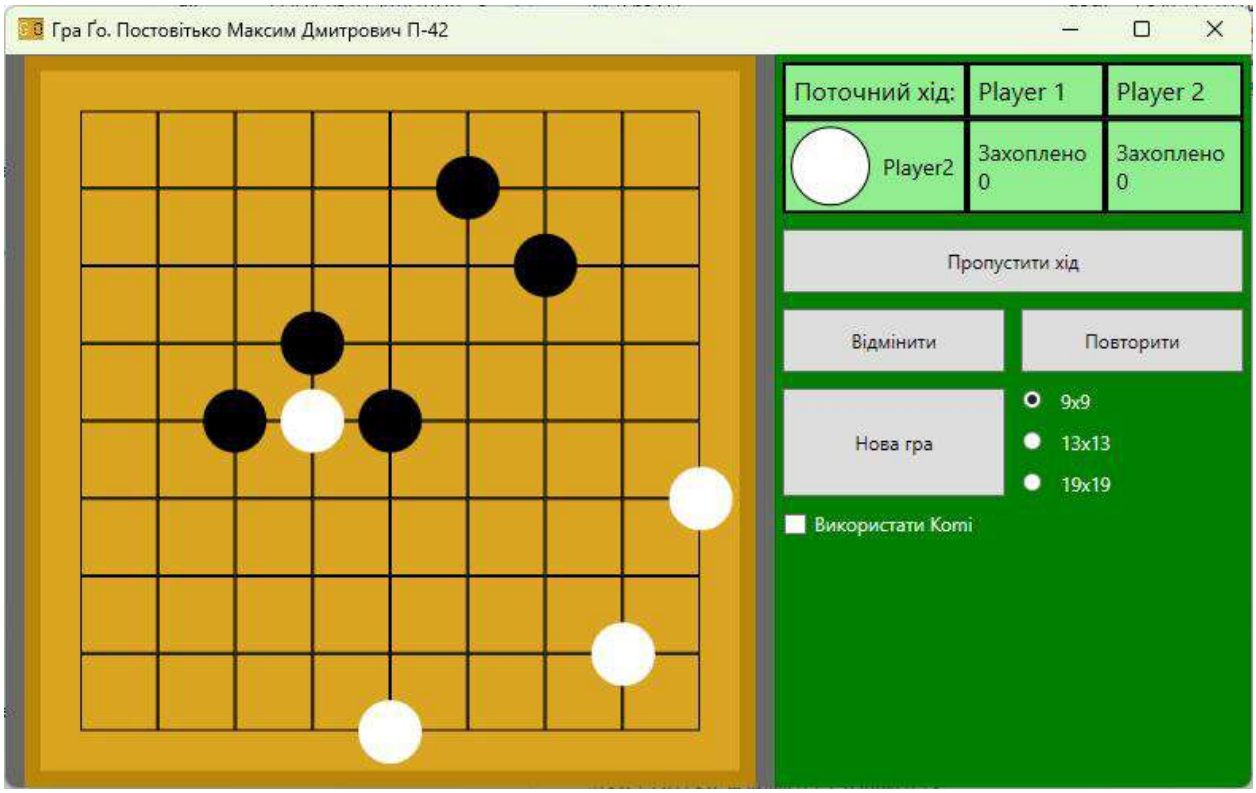

Рисунок 3.2.1 – Інтерфейс гри Ґо

Ігрова дошка розташована в лівій частині вікна і займає більшу його площу. Вона представлена сіткою золотисто-коричневого кольору з чорними лініями, що утворюють перетинання, на яких розміщуються камені. Розмір дошки залежить від обраного користувачем значення (9x9, 13x13 або 19x19), а кожна клітинка відображається у вигляді еліпса. Порожні клітинки мають прозоре заповнення, тоді як камені гравців позначаються чорним або білим кольором. При наведенні курсору миші на порожню клітинку з'являється напівпрозорий еліпс відповідного кольору поточного гравця, що дозволяє

заздалегідь оцінити хід. Клік мишею по клітинці розміщує камінь, якщо хід є валідним, інакше з’являється повідомлення про неможливість ходу.

Панель керування розміщена в правій частині вікна на зеленому фоні та містить усі необхідні елементи для взаємодії з грою. У верхній частині панелі відображається інформація про поточного гравця: напис "Player1" або "Player2" разом із еліпсом, який показує колір каменів (чорний або білий). Поруч розташовані два поля з очками гравців, позначені як "Захоплено" з відповідними значеннями для чорних і білих каменів. Ці дані оновлюються в реальному часі після кожного ходу чи захоплення.

Нижче інформаційної секції розміщені кнопки для керування грою. Кнопка "Пропустити хід" дозволяє гравцеві передати чергу супернику, а два послідовні пропуски завершують гру з підрахунком очок. Кнопки "Відмінити" та "Повторити" забезпечують скасування або повторення ходів, використовуючи історію гри, що зберігається в пам’яті. Кнопка "Нова гра" скидає поточний стан і запускає нову партію з обраним розміром дошки. Для вибору розміру дошки передбачено три радіокнопки з написами "9x9", "13x13" і "19x19", де за замовчуванням активовано "9x9". Поруч із ними є прапорець "Використати Komi", який увімкнений або вимкнений залежно від бажання гравців додати компенсаційні очки для білих (6.5 за замовчуванням).

Усі елементи інтерфейсу мають чітке візуальне оформлення: кнопки та поля обведені чорними рамками, текст виконано у читабельному шрифті розміром 14-16 пікселів, а кольорова схема (сірий фон вікна, зелена панель, золотиста дошка) сприяє зручному сприйняттю. Повідомлення про завершення гри чи помилки відображаються у спливаючих вікнах із детальною інформацією, наприклад, про переможця та різницю в очках.

Таким чином, інтерфейс користувача поєднує в собі функціональність і простоту, дозволяючи гравцям легко орієнтуватися в грі, виконувати ходи, керувати процесом і відстежувати стан партії. Він повністю відповідає потребам локальної гри для двох осіб і забезпечує інтуїтивну взаємодію з програмою.

3.3. Інструкція для роботи з програмою

Для успішного використання комп’ютерної версії гри Ґо, розробленої на базі технології WPF, користувачам необхідно ознайомитися з основними кроками запуску, налаштування та ігрового процесу. Ця інструкція описує порядок дій від запуску програми до завершення гри, забезпечуючи чітке розуміння всіх доступних функцій.

Спочатку потрібно запустити програму, двічі клацнувши на виконуваний файл "GoGame.exe", який розташований у папці зі встановленим додатком. Після запуску відкриється головне вікно з ігровою дошкою зліва та панеллю керування справа. За замовчуванням гра стартує з дошкою розміром 9x9, а першим гравцем є "Player1" із чорними каменями, що відображається еліпсом чорного кольору у верхній частині панелі керування.

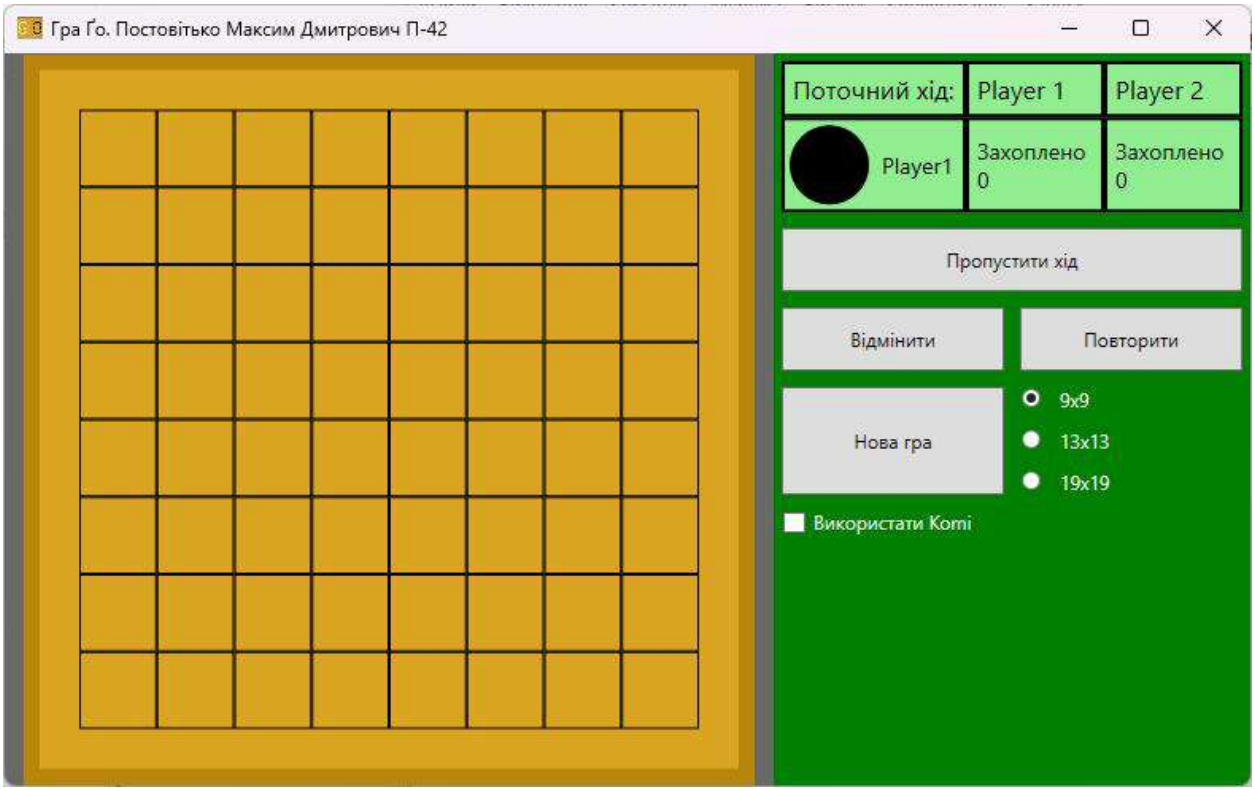

Рисунок 3.3.1 – Головне вікно

Перед початком гри користувачі можуть налаштувати параметри. На панелі керування розташовані три радіокнопки для вибору розміру дошки: "9x9", "13x13" або "19x19". Клацання на одну з них змінює розмір дошки, після чого потрібно натиснути кнопку "Нова гра", щоб застосувати зміни та

розпочати партію заново. Поруч із радіокнопками є прапорець "Використати Komi", який за бажанням можна активувати, щоб додати білим каменям компенсаційні очки (6.5), урівноваживши перевагу першого ходу чорних. Після налаштування параметрів гра готова до початку.

Для розміщення каменя на дошці гравець наводить курсор миші на потрібне перетинання ліній, бачить напівпрозорий еліпс із кольором поточного гравця (чорний або білий) і клацає лівою кнопкою миші. Якщо хід валідний, камінь з'являється на дошці, а програма автоматично перевіряє захоплення каменів противника та оновлює очки у полях "Захоплено" на панелі керування. У разі порушення правил (наприклад, самогубства чи правила Ко) з'являється спливаюче вікно з повідомленням про неможливість ходу, наприклад, "move is impossible", із координатами клітинки.

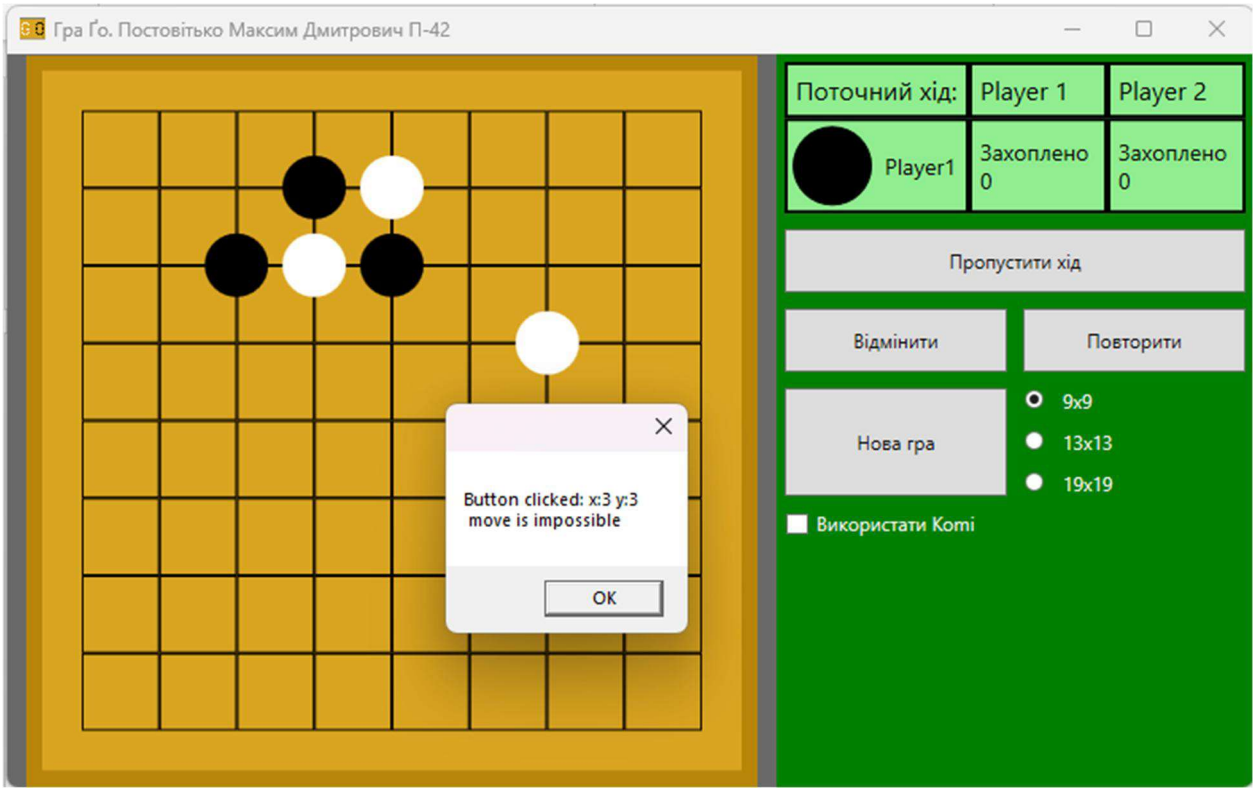

Рисунок 3.3.2 – Порушення правил

Гравці ходять по черзі, і після кожного ходу еліпс у верхній частині панелі змінює колір, а напис "Player1" або "Player2" оновлюється, вказуючи на поточного гравця. Якщо гравець вирішує пропустити хід, достатньо натиснути кнопку "Пропустити хід" на панелі керування, після чого черга переходить до

суперника. Два послідовні пропуски завершують гру, і з'являється спливаюче вікно з результатами: інформація про переможця, різниця в очках, підсумкові значення для обох гравців і кількість зроблених ходів.

Для виправлення помилок передбачені кнопки "Відмінити" та "Повторити". Натиснення "Відмінити" повертає гру до попереднього стану, видаляючи останній хід і оновлюючи дошку та очки. Кнопка "Повторити" відновлює скасований хід, якщо він був відмінений раніше. Ці функції доступні лише до завершення гри. Щоб розпочати нову партію в будь-який момент, потрібно натиснути кнопку "Нова гра", що скидає дошку, очки та історію ходів до початкового стану з обраним розміром дошки.

Після завершення гри у спливаючому вікні відображається підсумок, наприклад, "Перемога Player1 з різницею очок 5 камінців", із деталями про рахунок і кількість ходів. Користувачі можуть закрити вікно результатів і натиснути "Нова гра" для повторного запуску або завершити роботу програми, закривши головне вікно стандартним способом через кнопку закриття у верхньому правому куті.

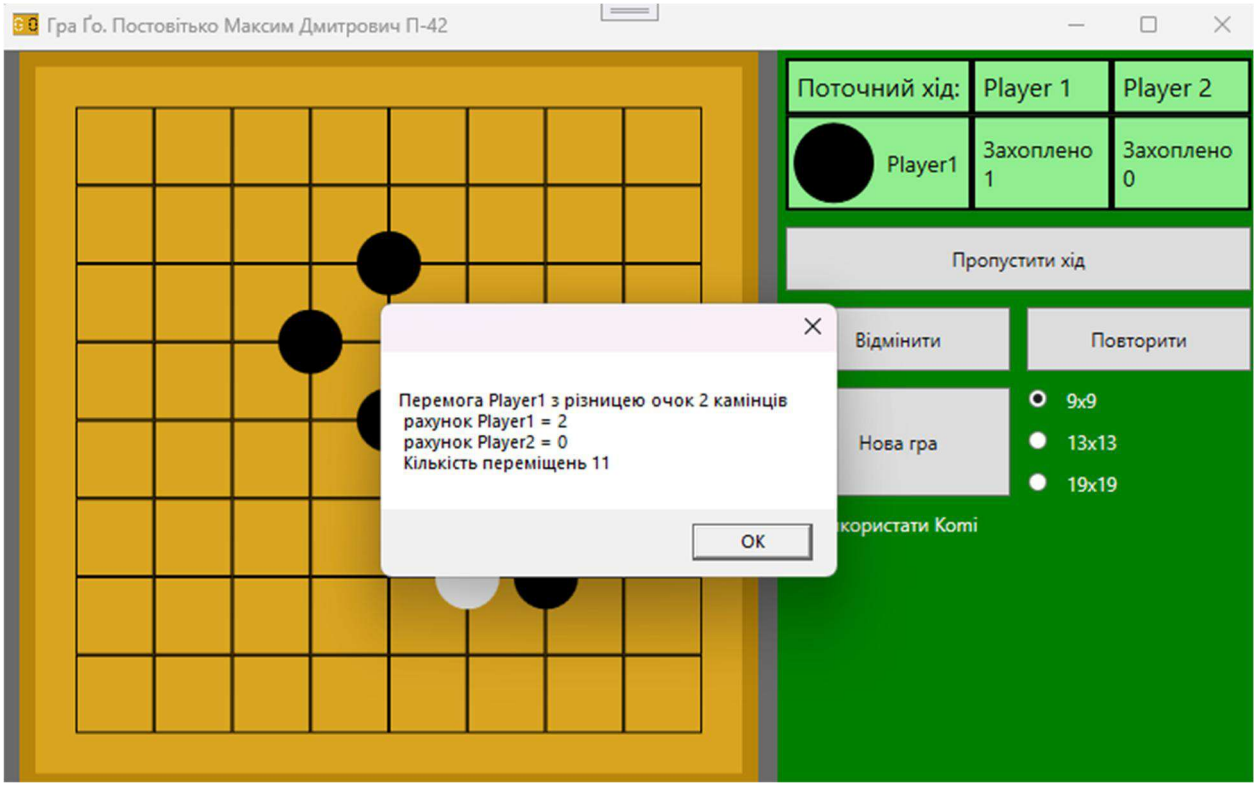

Рисунок 3.3.3 – Завершення гри

Робота з програмою є простою та інтуїтивною: запуск, налаштування розміру дошки та Komi, розміщення каменів кліками миші, використання кнопок для керування ходом гри та перегляд результатів. Інтерфейс і функціонал забезпечують зручність для гравців будь-якого рівня, дозволяючи насолоджуватися грою Ґо в локальному режимі.

Висновки до розділу 3

Третій розділ дипломної роботи присвячений опису практичного використання розробленої комп'ютерної версії гри Ґо, включаючи системні вимоги, інтерфейс користувача та інструкцію з експлуатації. На основі проведеного аналізу та опису можна зробити такі висновки:

Розроблена програма має мінімальні системні вимоги, що робить її доступною для запуску на більшості сучасних комп'ютерів під керуванням Windows. Операційна система Windows 10 або новіша версія з .NET Framework 4.8, процесор із частотою від 1 ГГц, 2 ГБ оперативної пам'яті та 50 МБ вільного місця на диску забезпечують стабільну роботу додатка. Такі характеристики дозволяють використовувати програму навіть на пристроях із низькою продуктивністю, що сприяє її широкій доступності для користувачів.

Інтерфейс користувача спроєктовано з урахуванням простоти та зручності, поєднуючи ігрову дошку з панеллю керування в єдиному вікні. Дошка відображає камені у вигляді еліпсів із підтримкою попереднього перегляду ходів, а панель керування надає інформацію про поточного гравця, очки та кнопки для управління грою, такі як пропуск ходу, скасування чи вибір розміру дошки. Чітке візуальне оформлення та інтуїтивне розташування елементів забезпечують легкість сприйняття та взаємодії, що є важливим для гравців-початківців і досвідчених користувачів.

Інструкція для роботи з програмою детально описує процес від запуску до завершення гри, включаючи налаштування параметрів, розміщення каменів, використання функцій Undo/Redo та перегляд результатів. Процедура є простою: запуск додатка, вибір розміру дошки та Komi за потреби, почергове

розміщення каменів кліками миші, керування ходом гри через кнопки та завершення партії пропуском ходів. Такий підхід дозволяє користувачам швидко освоїти програму та насолоджуватися грою без складних налаштувань чи додаткових вимог.

Розділ 3 підтверджує, що розроблена програма є зручним і доступним інструментом для гри в Ґо у локальному режимі. Низькі системні вимоги, продуманий інтерфейс і чітка інструкція забезпечують її практичну цінність для широкої аудиторії. Отримані результати слугуватимуть основою для подальшого аналізу ефективності програми та її вдосконалення в наступних розділах роботи.

ВИСНОВКИ

Дипломна робота присвячена розробці комп'ютерної версії гри Ґо на базі технології Windows Presentation Foundation (WPF). У процесі виконання роботи було проведено аналіз теоретичних основ, спроєктовано та реалізовано програмний продукт, а також підготовлено керівництво для його використання. На основі виконаних етапів можна сформулювати такі висновки:

Метою роботи було створення локального додатка для гри в Ґо для двох гравців із підтримкою основних правил, таких як розміщення каменів, захоплення, перевірка самогубства та правила Ко, а також із функціями вибору розміру дошки, підрахунку очок і скасування ходів. Ця мета досягнута завдяки комплексному підходу, який включав теоретичний аналіз, проектування архітектури та програмну реалізацію.

У першому розділі визначено задачі розробки, проаналізовано аналогічні системи, такі як Sabaki, OGS, Go Review і SmartGo, та обґрунтовано вибір WPF як оптимальної технології. Аналіз показав, що існуючі рішення часто орієнтовані на онлайн-гру чи аналіз партій, тоді як пропонований додаток заповнює нішу простих офлайн-програм. WPF обрано через її гнучкість у створенні графічних інтерфейсів, високу продуктивність і зручну інтеграцію з C# і .NET, що підтвердило правильність вибору для даного проекту.

Другий розділ охопив проектування та розробку програми. Обрані алгоритми для перевірки ходів, захоплення каменів і підрахунку очок мають часову складність $O(n^2)$, що є прийнятним для локального додатка, але оптимізація правила Ко за допомогою хешування знизила складність із $O(n^2)$ до $O(1)$, значно підвищивши швидкодію. Спроєктована архітектура з класами Game, Board, GameBoard, MainWindow та іншими забезпечила модульність і легкість підтримки. Реалізація в Visual Studio з використанням XAML і C# дозволила створити стабільний додаток із підтримкою дощок розміром 9x9, 13x13 і 19x19, функціями Undo/Redo та правилом Komi.

Третій розділ описав практичне використання програми. Мінімальні системні вимоги (Windows 10, 2 ГБ ОЗП, 50 МБ на диску) роблять її доступною для широкої аудиторії. Інтерфейс із ігровою дошкою та панеллю керування є інтуїтивним, а інструкція чітко пояснює запуск, налаштування, ходи та завершення гри. Це підтверджує практичну цінність додатка як інструменту для гри та навчання.

Розроблений програмний продукт відповідає поставленим задачам, забезпечуючи автентичний ігровий досвід у локальному режимі. Він є простим у використанні, стабільним і гнучким завдяки підтримці різних розмірів дошки та додаткових правил. Водночас робота відкриває перспективи для вдосконалення: додавання ШІ-суперника, збереження партій у файл, оптимізація групування каменів для підвищення продуктивності на великих дошках.

Дипломна робота досягла своєї мети, створивши функціональний додаток для гри в Ґо, який може бути використаний для розваг і популяризації гри. Отримані результати демонструють успішне застосування технології WPF і створюють основу для подальшого розвитку проекту в майбутньому.

ПЕРЕЛІК ЛІТЕРАТУРНИХ ДЖЕРЕЛ

1. Бойко О. В. Основи програмування на C# : навч. посіб. Київ : Видавнича група BHV, 2018. 320 с.
2. Гоґан Р. Гра Ґо: стратегія та тактика. Пер. з англ. Львів : Видавництво Старого Лева, 2019. 256 с.
3. Дейтел П., Дейтел Г. C# для програмістів. Пер. з англ. Київ : Діалектика, 2020. 960 с.
4. Еванс Дж. WPF: Windows Presentation Foundation у .NET 4.5. Пер. з англ. Харків : Фабрика знань, 2017. 512 с.
5. Кір'янов Д. В., Кір'янова О. М. Алгоритми та структури даних : навч. посіб. Київ : КПІ ім. Ігоря Сікорського, 2021. 288 с.
6. Кормен Т., Лейзерсон Ч., Рівест Р., Стайн К. Алгоритми: побудова й аналіз. Пер. з англ. 3-тє вид. Львів : Свічадо, 2016. 1296 с.
7. Нейтан П. Windows Presentation Foundation: основи розробки. Пер. з англ. Дніпро : Баланс-Клуб, 2019. 432 с.
8. Седжвік Р. Алгоритми на C#. Пер. з англ. Київ : Видавництво "КМ-БУКС", 2020. 720 с.
9. Таненбаум Е. Сучасні операційні системи. Пер. з англ. 4-те вид. Харків : Клуб Сімейного Дозвілля, 2018. 1120 с.
10. Троелсен Е., Джепікс Е. C# 8.0 та платформа .NET Core 3.0. Пер. з англ. Київ : ДіаСофт, 2021. 1248 с.
11. Чжан І. Майстерність гри в Ґо: від початківця до професіонала. Пер. з кит. Одеса : Астропринт, 2020. 384 с.
12. Шоттс Дж. Історія гри Ґо: від стародавності до сучасності. Пер. з англ. Київ : Наш Формат, 2019. 208 с.
13. Яворський Б. М. Програмування графічних інтерфейсів у WPF : навч. посіб. Львів : ЛНУ ім. Івана Франка, 2022. 256 с.
14. Бєлоусов О. І., Ткачук М. В. Основи об'єктно-орієнтованого програмування : навч. посіб. Харків : ХНУРЕ, 2020. 344 с.

15. Григор’єв Ю. М. Розробка настільних додатків у .NET : навч. посіб. Київ : НТУУ "КПІ", 2019. 296 с.
16. Microsoft Corporation. Офіційна документація WPF [Електронний ресурс]. Redmond : Microsoft, 2023. URL: https://docs.microsoft.com/uk-ua/dotnet/desktop/wpf/ (дата звернення: 03.04.2025).
17. Sensei’s Library. Правила гри в Ґо [Електронний ресурс]. 2022. URL: https://senseis.xmp.net/?RulesOfGo (дата звернення: 03.04.2025).
18. Кім Дж. Стратегії гри в Ґо для початківців // Журнал інтелектуальних ігор. 2021. № 3. С. 45–52.
19. Петренко С. О. Оптимізація алгоритмів у настільних іграх // Вісник Київського національного університету імені Тараса Шевченка. Серія: Фізико-математичні науки. 2020. № 2. С. 78–85.
20. Лозовий І. В. Використання WPF для створення інтерактивних додатків // Наукові записки НТУ "ХПІ". 2019. № 5. С. 112–120.

ДОДАТКИ

ДОДАТОК А.

Програмний код модулів

```
<Window x:Class="GoGame.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:GoGame"
        xmlns:locall="clr-namespace:GoGame.Views"
        mc:Ignorable="d"
        Title="Гра Го. Постовітько Максим Дмитрович П-42" Height="500"
Width="800" Icon="/GoGameIcon.png">
    <Grid Width="Auto" Background="DimGray">
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="1*"></ColumnDefinition>
            <ColumnDefinition Width="300"></ColumnDefinition>
        </Grid.ColumnDefinitions>
        <ContentControl Grid.Column="0" x:Name="contentControl" Width="Auto"
Height="Auto" HorizontalAlignment="Center" VerticalAlignment="Center"/>
        <StackPanel Grid.Column="1" Background="Green">
            <Grid Margin="5" Background="LightGreen">
                <Grid.ColumnDefinitions>
                    <ColumnDefinition Width="2*"></ColumnDefinition>
                    <ColumnDefinition Width="2*"></ColumnDefinition>
                    <ColumnDefinition Width="3*"></ColumnDefinition>
                    <ColumnDefinition Width="3*"></ColumnDefinition>
                </Grid.ColumnDefinitions>
                <Grid.RowDefinitions>
                    <RowDefinition Height="2*"></RowDefinition>
                    <RowDefinition Height="5*"></RowDefinition>
                </Grid.RowDefinitions>
                <Border Grid.Column="0" Grid.Row="0" BorderBrush="Black"
BorderThickness="2" Grid.ColumnSpan="2">
```