МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ ЕКОНОМІКИ І ТЕХНОЛОГІЙ

ННІ/факультет	Інформаційних технологій
Кафедра	Інформатики і прикладного програмного забезпечення
Спеціальність	Інженерія програмного забезпечення
Форма навчання	Денна

КВАЛІФІКАЦІЙНА БАКАЛАВРСЬКА РОБОТА

Омельяненко Юлії Владиславівни
(прізвище, ім'я, по батькові здобувача)

на тему **Розробка настільної гри "Го"**
(повна назва теми)

за матеріалами праць провідних спеціалістів з розробки ПЗ та проектування БД
(повна назва бази дослідження)

науковий керівник д.т.н. професор _____ Зеленський О. С.
(наук. ступінь, вчене звання) *(підпис)* *(прізвище, ініціали)*

Робота допущена до захисту в ЕК
Протокол засідання кафедри
від 11.06.2025 № 12
Завідувач кафедри _____
(підпис)
д.т.н., професор Зеленський О.С.
Наук. ступень, вчене звання *Ініціали, прізвище*

Кривий Ріг – 2025

ЗГОДА здобувачки вищої освіти

Державного університету економіки і технологій про перевірку кваліфікаційної роботи на прояви академічного плагіату та розміщення в Репозитарії Університету

Я, Омельяненко Юлія Владиславівна, підтримую політику Державного університету економіки і технологій з академічної доброчесності і відкритого доступу.

Засвідчую, що кваліфікаційна бакалаврська робота «Розробка настільної гри "Го"» виконана самостійно та не містить академічного плагіату. Я не надавала і не одержувала недозволену допомогу під час підготовки цієї роботи. Робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.

Із чинним Положенням про запобігання та виявлення академічного плагіату в роботах здобувачів вищої освіти Державного університету економіки і технологій ознайомлена. Чітко усвідомлюю, що в разі виявлення у кваліфікаційній роботі порушення норм академічної доброчесності робота не допускається до захисту або оцінюється незадовільно.

Також я поінформована, що відповідно до «Положення про Репозитарій (електронну базу даних) Державного університету економіки і технологій» зазначена робота буде розміщена в Електронному архіві Університету (Репозитарії ДУЕТ). З умовами такого розміщення ознайомлена.

Дата підпис ініціали, прізвище (власноруч)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ ЕКОНОМІКИ І ТЕХНОЛОГІЙ

ННІ/факультет	Інформаційних технологій
Кафедра	Інформатики і прикладного програмного забезпечення
Спеціальність	Інженерія програмного забезпечення
Форма навчання	Денна

«ЗАТВЕРДЖУЮ»
Завідувач кафедри _____ Зеленський О.С.
(підпис) (Прізвище, ініціали)
«__11__»__червня__ 2025_ року

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ БАКАЛАВРСЬКУ РОБОТУ

1. Тема роботи «Розробка настільної гри "Го"»_____

Керівник роботи д.т.н., професор Зеленський О. С._____
затверджені наказом закладу вищої освіти від «4 » квітня 2025 р. № 222-ст

2. Строк подання здобувачем роботи до «09» червня 2025 р.

3. Зміст кваліфікаційної роботи, об’єкт, предмет та мета дослідження:

Розділ 1. Постановка задачі

Розділ 2. Проектування задачі

Розділ 3. Проектування бази даних для проекту

Розділ 4. Розробка гри

Об’єкт дослідження: Гра "Го"

Предмет дослідження: розробка алгоритму партій гри

Мета кваліфікаційної роботи: створення настільної гри "Го"

5. Дата видачі завдання «4 » квітня 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів МДР	Строк виконання етапів роботи	Відмітка керівника про виконання етапів (дата, підпис)
1	Підготовка розділу 1	04.04.2025-13.04.2025	
2	Підготовка розділу 2	14.04.2025-26.04.2025	
3.	Підготовка розділу 3	27.04.2025-15.05.2025	
4	Підготовка розділу 4	16.05.2025-08.06.2025	
5	Реєстрація завершеної кваліфікаційної роботи	09.06.2025	Реєстраційний №___ «09»червня 2025 р.
6	Отримання відгуку від наукового керівника	10.06.2025	
7	Подання кваліфікаційної роботи на перегляд завідувачу кафедри	11.06.2025	
8	Отримання зовнішньої рецензії	12.06.2025	
9	Попередній захист кваліфікаційної роботи на кафедрі	13.06.2025	
10	Підготовка до захисту в ЕК	16.06.2025-21.06.2025	

Завдання підготував науковий керівник _____ Зеленський О.С.

(підпис) (прізвище та ініціали)

Завдання одержав _____ Омельяненко Ю.В.

(підпис) (прізвище та ініціали)

АНОТАЦІЯ

на бакалаврську кваліфікаційну роботу

«Розробка настільної гри "Го"»

Омельяненко Юлії Владиславівни

Кваліфікаційна бакалаврська робота на здобуття освітньо-кваліфікаційного рівня бакалавра зі спеціальності 121 «Інженерія програмного забезпечення» – Державний університет економіки і технологій – Кривий Ріг, 2025.

Основним завданням даної бакалаврської дипломної роботи було створення програмного застосунку для гри Go з можливістю збереження, завантаження ігор.

У роботі було вдосконалено теоретико-практичні підходи до створення десктопних ігор із використанням об’єктно-орієнтованого програмування. Здійснено проектування архітектури застосунку, реалізовано взаємодію з базою даних SQL Lite, а також забезпечено зручність використання інтерфейсу. Для реалізації програмного забезпечення використано мову програмування C# у середовищі .NET, а також бібліотеку Windows Forms для побудови інтерфейсу користувача. З боку бази даних реалізовано зберігання інформації про гравців, хід гри та завершені партії. Здійснено підключення до СУБД SQL Lite через відповідні запити, які забезпечують роботу ігрового алгоритму.

Запропоновано рішення для реалізації гри з логікою ходу, пасу, збереженням і завантаженням, що може бути основою для подальшого розвитку.

Ключові слова: настільна гра Go, C#, .NET, Windows Forms, база даних, SQL Lite, збереження гри, інтерфейс користувача.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

Скорочення	Розшифрування
SQL Lite	Компактна вбудована СУБД
СУБД	Система управління базми даних
OpenGL	Специфікація, що визначає платформонезалежний програмний інтерфейс для написання додатків, що використовують двовимірну та тривимірну комп'ютерну графіку
WinForms	Інтерфейс програмування програм, що відповідає за графічний інтерфейс користувача і є частиною Microsoft .NET Framework
Алгоритм	Набір інструкцій, які описують порядок дій виконавця, щоб досягти результату розв'язання задачі за скінченну кількість дій
Структура даних	Спосіб організації даних в більш складні типи даних

ЗМІСТ

ВСТУП

У сучасному світі ігрова індустрія займає значне місце серед програмного забезпечення. Розробка інтелектуальних настільних ігор для комп'ютерів є не лише засобом розваги, а й способом тренування логічного мислення, стратегічного планування та прийняття рішень. Особливої популярності набули класичні ігри, серед яких особливе місце займає гра Go — одна з найдавніших стратегічних ігор у світі. Незважаючи на простоту правил, вона має велику кількість можливих варіацій та вимагає високої концентрації уваги від гравців. Усе це робить Go цікавою для програмної реалізації з освітнім та практичним потенціалом.

Розробка настільного застосунку для гри Go є актуальною, оскільки вона дозволяє перенести традиційну логічну гру в цифрове середовище, що відкриває можливості для збереження та повторного перегляду ігор для проведення аналізу. Крім того, така розробка дає змогу користувачам грати з іншими людьми, не покидаючи свого пристрою. Створення програмного забезпечення для Go також сприяє збереженню культурної спадщини та популяризації інтелектуальних ігор серед широкої аудиторії.

Метою дипломної роботи є розробка програмного забезпечення для гри Go з можливістю збереження, завантаження та перегляду ігрових партій, реалізацією базової логіки гри та інтеграцією з базою даних для зберігання історії ігор та інформації про гравців.

Об'єктом дослідження є процес програмної реалізації логічних стратегічних ігор у середовищі Windows.

Предметом дослідження є методи та засоби реалізації ігрової логіки, обробки даних, збереження і відновлення стану гри в контексті розробки комп'ютерної версії гри Go.

Для досягнення поставленої мети необхідно вирішити такі завдання:

1. Проаналізувати правила гри Go та визначити вимоги до програмної реалізації.

2. Спроектувати архітектуру додатку та визначити структуру основних модулів.
3. Розробити графічний інтерфейс користувача для взаємодії з грою.
4. Реалізувати основну логіку гри, включаючи постановку каменів, передачу ходу та завершення гри.
5. Спроектувати базу даних для зберігання інформації про ігри та гравців.
6. Створити базу даних для зберігання інформації про ігри та гравців.
7. Реалізувати механізми збереження та завантаження стану гри з бази даних.

РОЗДІЛ 1
ПОСТАНОВКА ЗАДАЧІ

1.1. Характеристика задачі

У рамках дипломної роботи буде реалізовано програму, яка імітує настільну гру Го для гри на персональному комп’ютері. Основна цільова аудиторія — українські користувачі, зацікавлені у стратегічних настільних іграх. Розробка здійснюватиметься з використанням мови програмування C# та фреймворку Windows Forms (далі WinForms). WinForms — це графічний інтерфейс користувача (GUI) для платформи .NET, який дозволяє створювати класичні настільні додатки для Windows. Цей фреймворк забезпечує прості засоби для створення вікон, кнопок, панелей, обробки подій [3].

Ось основні характеристики WinForms:

- Підтримка широкого набору елементів керування.
- Наявність конструктора форм.
- Інтерфейс реалізується через класи та події
- Підтримка спадкування
- Підтримка сторонніх бібліотек, зокрема OpenGl
- Підтримка в ОС Windows

Як було зазначено раніше, гра орієнтована на запуск у середовищі Windows. Microsoft Windows є найпоширенішою операційною системою серед користувачів персональних комп’ютерів.

Ця операційна система створює сприятливі умови для розробки та стабільного функціонування десктопних застосунків, реалізованих на базі технології WinForms. Завдяки оптимальному розподілу системних ресурсів та управлінню обчислювальними процесами забезпечується робота програмного забезпечення навіть на комп’ютерах із невисокими технічними характеристиками.

Графічна підсистема Windows надає інструменти для реалізації подвійної буферизації, що дозволяє уникнути візуальних артефактів при оновленні інтерфейсу й гарантує плавне відтворення графіки. Система управління пам’яттю, такі як механізм автоматичного збирання сміття Garbage Collector, дозволяє мінімізувати ймовірність витоків пам’яті та спрощує обробку об'єктів у динамічному середовищі виконання.

Платформа також забезпечує розвинену модель багатопотокового виконання, що дає змогу відокремлювати обчислювальну логіку від графічного інтерфейсу, підвищуючи чутливість програми до дій користувача.

Підтримка уніфікованого кодування Unicode забезпечує коректне відображення текстової інформації різними мовами, включаючи українську.

Windows також характеризується високим рівнем сумісності між різними версіями (Windows 7, 8, 10, 11). Тому при розробці можна уникнути потреби у додатковій розробці адаптації до кожної версії ОС.

Доступ до файлової системи, засобів обробки системного часу, подій введення та інших функціональних компонентів здійснюється через інтегровані API-платформи .NET.

У межах даної дипломної роботи реалізація гри Го має низку свідомо заданих обмежень, обумовлених як технічними чинниками, так і часовими рамками виконання проекту. Ці обмеження не впливають на основну функціональність гри, але визначають межі її використання та потенційні напрями майбутнього розширення.

Реалізація мережевої взаємодії між клієнтами — зокрема, через TCP/IP або з використанням веб-сервісів — не входить до поточної архітектури програми. Він функціонує в режимі локального двокористувацького сеансу за одним комп’ютером. Включення мережевої складової потребуватиме впровадження додаткових компонентів, таких як система аутентифікації, обробка затримок мережі, синхронізація ігрового стану між клієнтами та сервером.

У межах реалізованого проекту не передбачено розробки ігрової партії проти комп’ютера. Усі партії здійснюються у режимі взаємодії між двома реальними гравцями за одним пристроєм. Причиною відмови від функціоналу є висока обчислювальна складність побудови ефективної стратегії гри у Го, навіть для базового рівня. Більшість відомих підходів до реалізації ШІ для цієї гри базуються на використанні дерев рішень великої глибини, стохастичних симуляцій або нейронних мереж, що потребують значного обсягу оперативної пам’яті, високої тактової частоти процесора та значного часу на обчислення кожного ходу. Поставлені задачі передбачають розробку додатку, який буде доступний для слабких пристроїв, впровадження ресурсоємного компонента у вигляді ШІ суперечить вимогам до продуктивності та мінімальних системних вимог.

Програма функціонує автономно, без підключення до зовнішніх серверів або баз даних. Уся ігрова логіка, а також збереження/завантаження партій реалізовані локально. Програма позбавляється від залежності до інтернет-з’єднання, за допомогою такого підходу, і дає можливість охопити більше аудиторії.

У перспективі така архітектура може бути розширена шляхом додавання серверної частини або інтеграції з існуючими ігровими платформами, однак у межах цієї роботи реалізація подібних можливостей не розглядається як пріоритет.

1.2. Правила гри в Го

Го — це стратегічна настільна гра, що виникла в Давньому Китаї понад 2500 років тому [1]. Вона належить до класу абстрактних логічних ігор із повною інформацією та двома учасниками. Незважаючи на простоту базових правил, гра потребує глибокого стратегічного мислення, через велику кількість можливих позицій і стратегічних варіантів.

Пропоную розглянути правила гри в Го на прикладі мобільної гри від компанії Popoko.

Почнемо з ігрової дошки. У класичному варіанті ігрове поле має вигляд квадратної сітки розміром 19 на 19 ліній, утворюючи 361 перехрестя. Але таке поле надто велике, потребує багато часу на гру та деякий досвід гри. Тому для новачків було впроваджено ще можливий розмір дошки 4 на 4, 5 на 5, 7 на 7, 9 на 9, 15 на 15 та 13 на 13 (Рис. 1.1.).

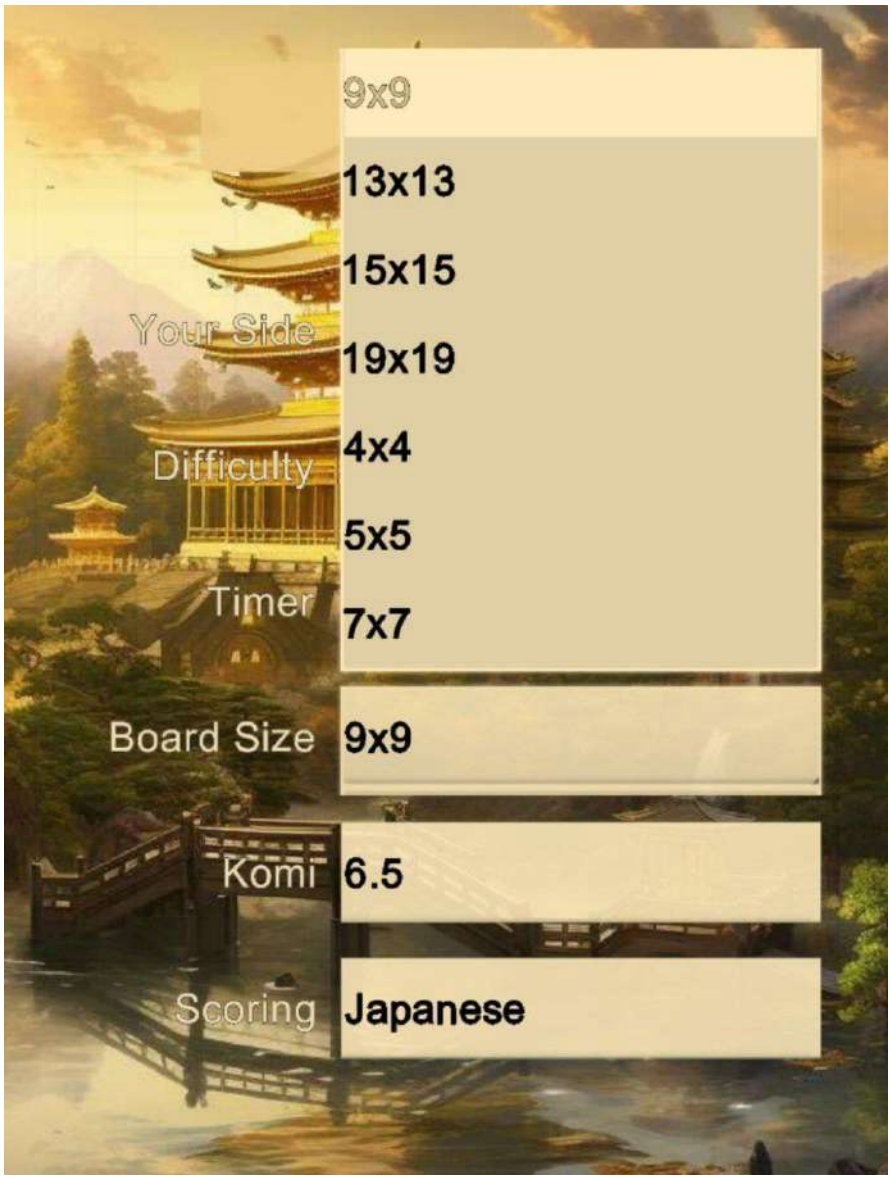

Рис. 1.1. Доступні розміри дошки

Основна мета гри — контролювати більшу частину ігрового поля, ніж суперник, шляхом оточення території та захоплення каменів опонента. Територія визначається як вільні перехрестя, повністю оточені каменями одного кольору.

Гравці ходять по черзі, розміщуючи один камінь за хід на будь-яке вільне перехрестя. Першим грає гравець з чорними каменями (Рис. 1.2.)

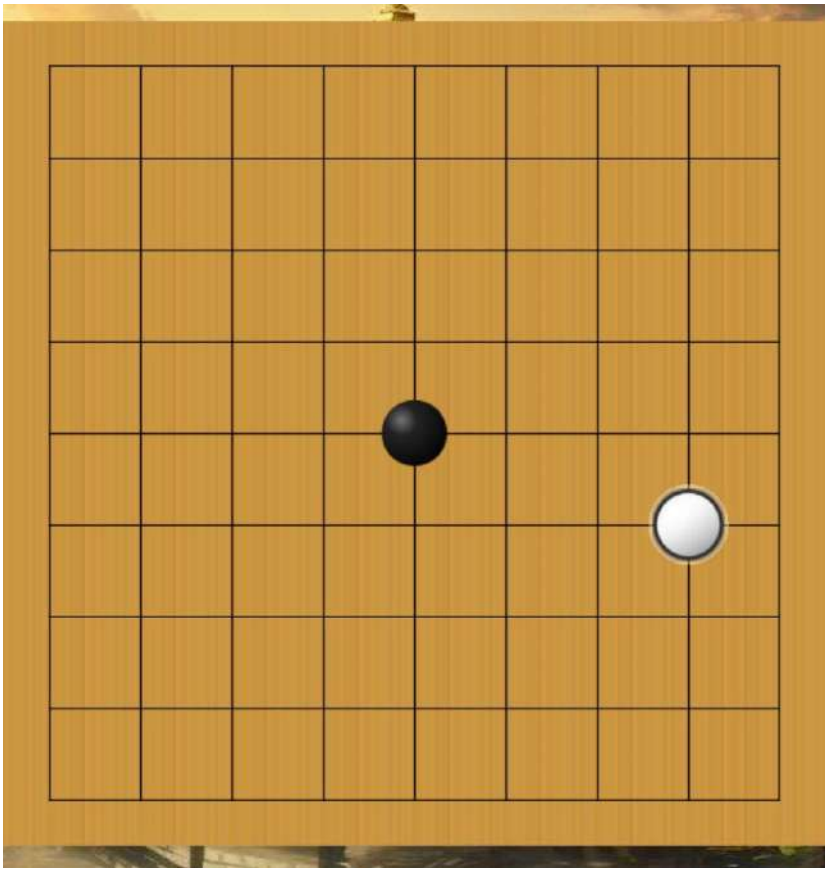

Рис. 1.2. Перший хід гри

Камінь або група з'єднаних каменів одного кольору вважається захопленим, якщо всі його дихання (дихання — суміжні вільні перехрестя) оточені каменями супротивника. Захоплені камені знімаються з дошки, і кожен з них додається до рахунку гравця, що його захопив (Рис. 1.3. та Рис. 1.4.).

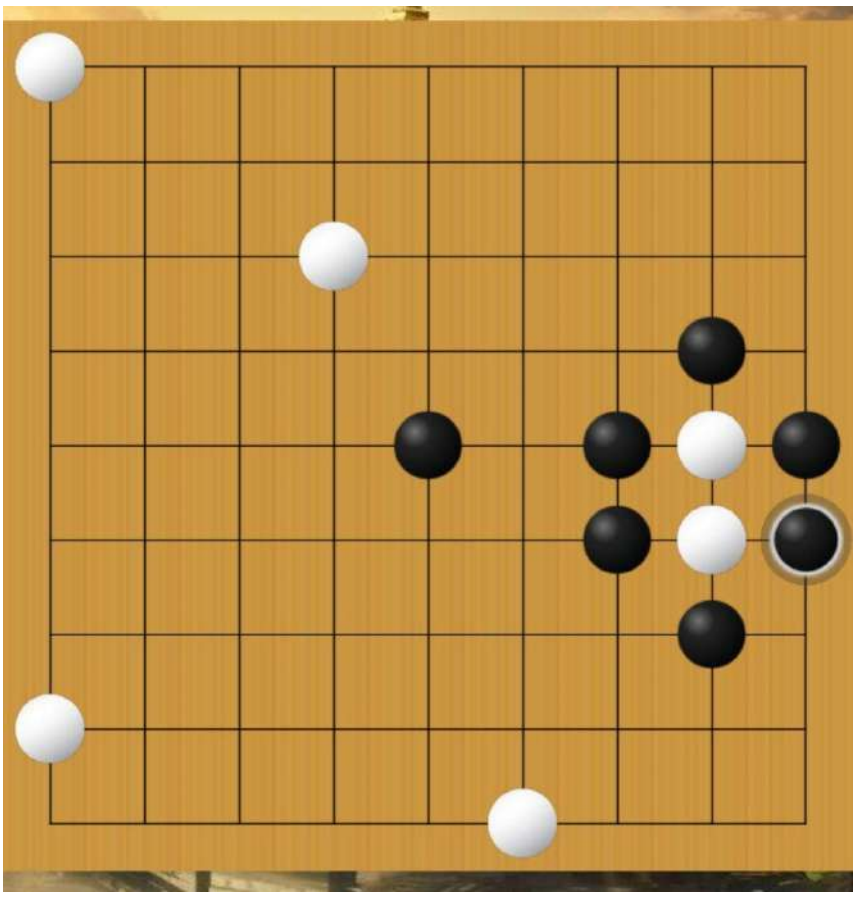

Рис. 1.3. Захоплення каменю супротивника

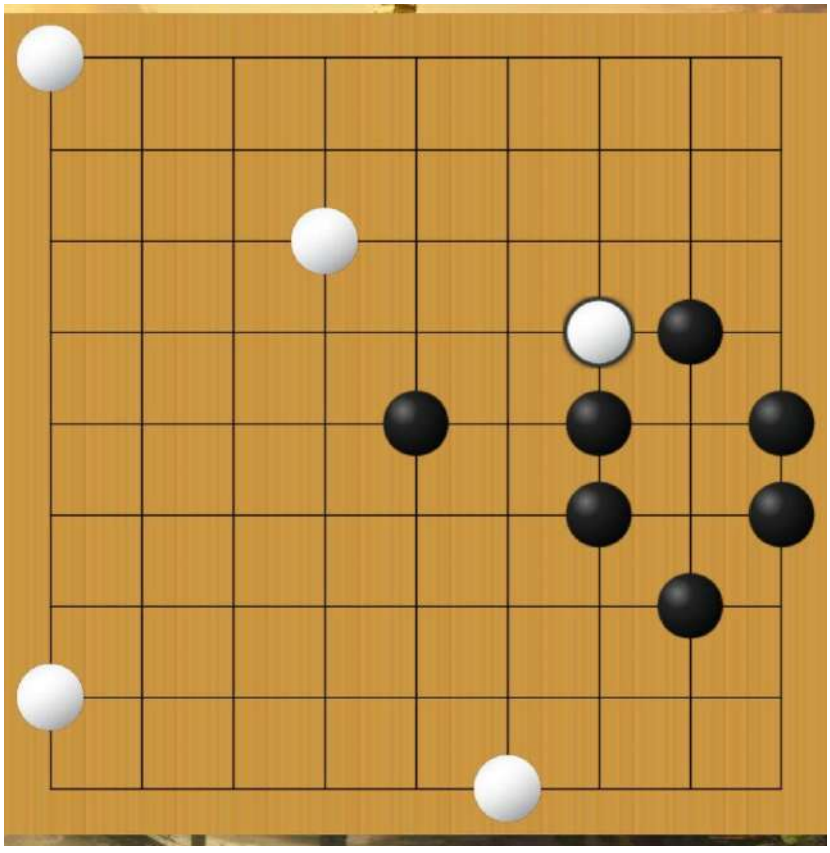

Рис. 1.4. Каміння прибираються з поля

Гравець не має права робити хід, у результаті якого власний камінь чи група каменів буде одразу повністю оточені без залишку дихання, за винятком випадків, коли такий хід призводить до захоплення каменів супротивника, що відкриває нове дихання. Таке правило називається «заборона на самогубство», оскільки камінь по суті ставиться і одразу ж прибирається з поля (Рис. 1.5.).

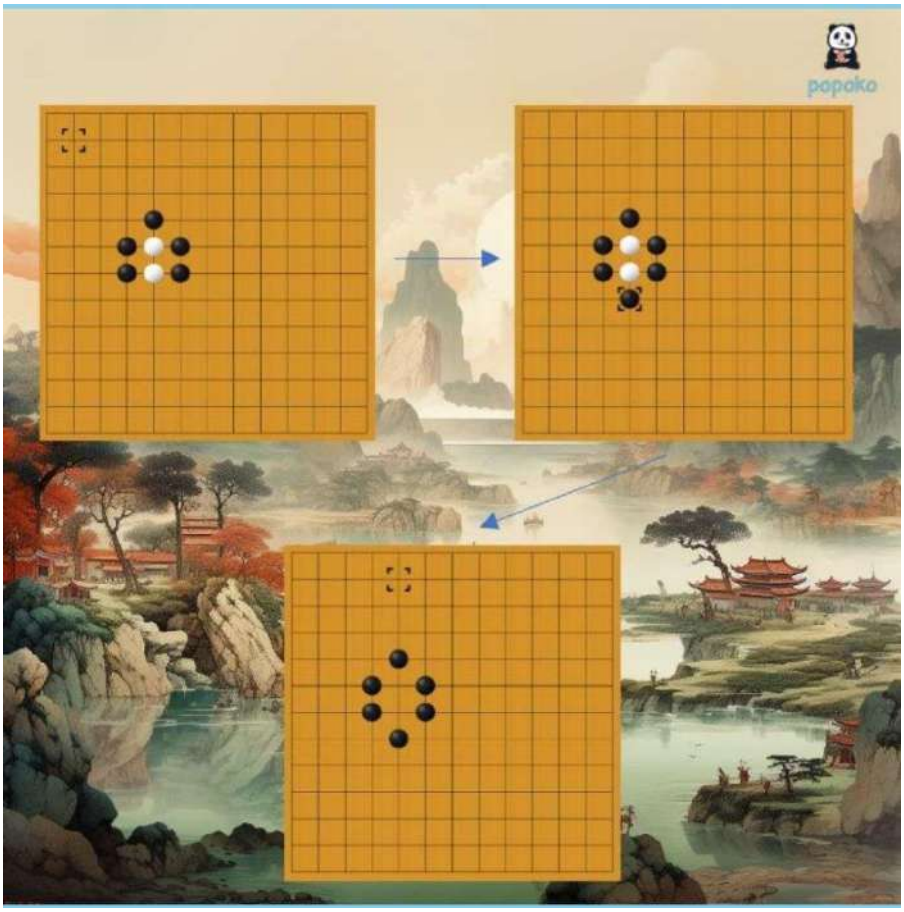

Рис. 1.5. Заборона на самогубство

За правилами гри забороняється хід, що відновлює попередню позицію, яка була на полі за один хід до цього (Рис. 1.6.). Це запобігає нескінченним циклам захоплень і негайного повернення каменів. Таке правило має назву правило «ко» (від яп. "вічність").

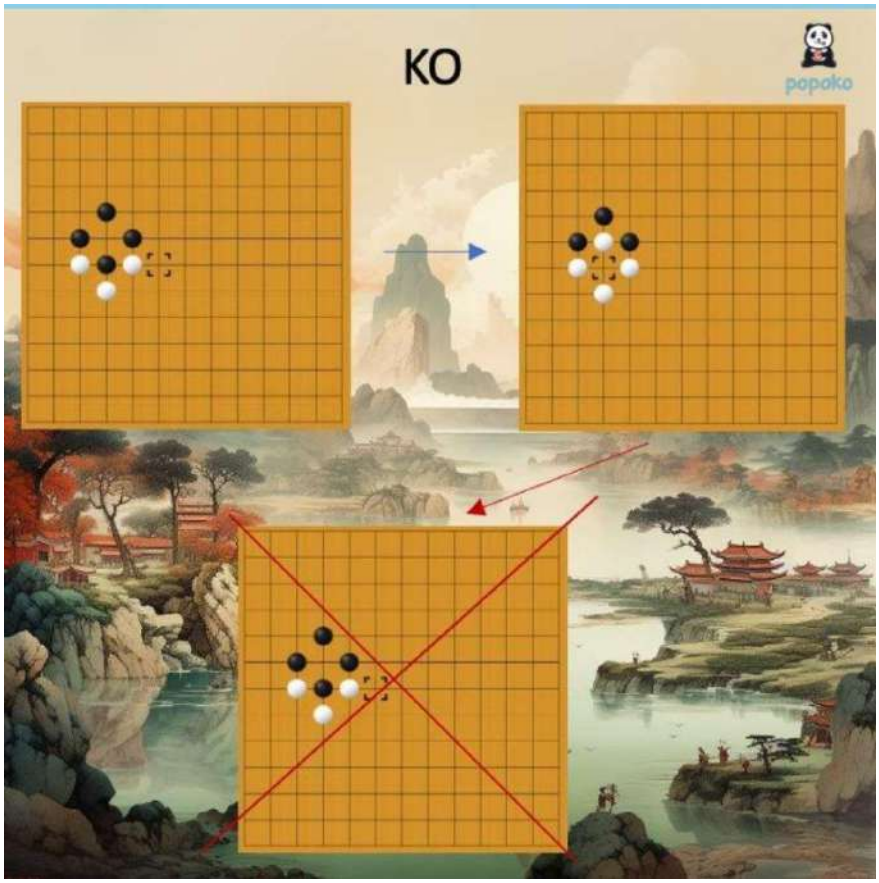

Рис. 1.6. Правило «ко»

За правилами гри гравці ходять по черзі, але передбачена можливість пасування та пропуску ходу. Коли гравці спасували поспіль один за одним, гра завершується і визначається переможець. Очки рахуються за наступною методологією:

- рахується кількість перехресть, що формують контрольовану територію;
- кількість захоплених каменів супротивника;
- компенсаційні очки (комі), які зазвичай надаються білому гравцеві (грає другим) для врівноваження переваги чорного першого ходу.

Стандартне значення комі — 6,5 очок. Останнім часом від цього правила все більше гравців відмовляється, адже вважають що перший хід не може суттєво вплинути на результати гри.

Перемагає той гравець у якого в сумі більше контрольованих перехресть. Хочу запропонувати ознайомитися з усіма термінами гри у табл. 1.1 Ці терміни будуть використовуватися протягом усього опису дипломної роботи.

Таблиця 1.1

Терміни гри Го

Термін	Пояснення
Камінь	Ігровий елемент чорного або білого кольору
Дихання	Вільне суміжне перехрестя, необхідне для "життя" каменя або групи
Група каменів	Набір з’єднаних за лініями каменів одного кольору
Захоплення	Зняття каменів супротивника, які повністю втратили дихання
Комі	Компенсаційні очки, що надаються білому гравцеві
Ко	Спеціальне правило, що забороняє безкінечне відтворення позиції
Територія	Область поля, повністю оточена каменями одного кольору
Пас	Добровільна відмова від ходу

Хотілося б також звернути увагу на можливості, які надає мобільна гра, але вони не пов’язані з правилами гри. У головному меню (Рис. 1.7.) можна створити гру, зберегти гру або ж завантажити існуючу гру.

Рис. 1.7. Запропоновані можливості гри

Це однозначно дуже корисна можливість, яка розширює гнучкість програми. Завдяки можливості збереження гри користувач може перервати сесію в будь-який момент без втрати прогресу, що є актуальним для Го, де окремі партії можуть тривати досить тривалий час. А ще завантаження збережених ігор дозволяє гравцям повертатися до минулих сесій для повторного перегляду та аналізу. Оскільки гра Го про стратегію, то гравці охоче будуть користуватися можливістю проаналізувати старі партії.

Ідея інтеграції такої можливості розширює аудиторію та надає унікальності грі. Тож було прийнято рішення теж інтегрувати базу даних та додати можливість збереження партій.

1.3. Постановка вимог до програми

Як було зазначено раніше, програмна реалізація гри Го створюється з використанням мови програмування C# у середовищі .NET Framework, із застосуванням технологій WinForms для побудови графічного інтерфейсу

користувача та OpenGL для реалізації графічного виводу дошки та елементів гри.

Серед популярних альтернативних засобів для створення GUI-додатків існують також середовища WPF, Qt, JavaFX, та повноцінні кросплатформені фреймворки на зразок Unity, Godot, або Electron. Проте кожен із них має як переваги, так і недоліки.

WPF (Windows Presentation Foundation) — це більш сучасна технологія у межах екосистеми .NET, яка має різноманітні інструменти для розробки адаптивних і стилізованих інтерфейсів [12]. Проте, вона вимагає більше ресурсів на рендеринг, і є надлишковою для проекту, де інтерфейс є функціональним, а не декоративним. До того ж, WPF тісно пов’язаний з DirectX, що обмежує можливості апаратного прискорення при використанні OpenGL.

Із популярних платформ для побудови графіки існує також така платформа, як Unity. Вона підтримує C# і має методи реалізації для візуалізації. Однак вона орієнтована переважно на тривимірні ігри та мультимедійні застосунки, що не відноситься до поставлених задач. Також Unity має суттєво вищі системні вимоги, що суперечить цілі забезпечення стабільної роботи на малопотужних комп’ютерах.

JavaFX і Qt — кросплатформенні інструменти, що потребують інших мов програмування (Java, C++), які є менш бажаним у контексті проекту, орієнтованого на Windows-середовище з використанням C#.

WinForms був обраний, як засіб розробки, через його простоту, передбачувану поведінку, продуктивність для простих інтерфейсів, слабких пристроїв та хорошою інтеграцією з OpenGL. Платформа дозволяє в режимі конструктора створити зручний у використанні інтерфейс із базовими елементами управління. WinForms підтримує системні ресурси Windows, має зрозумілу модель подій і не вимагає складної конфігурації для запуску.

Застосування OpenGL для створення графічного інтерфейсу гри дозволяє без проблем мати високу швидкість рендерингу, незалежної від

віконних компонентів Windows, і забезпечувати потенційну можливість розширення до анімацій.

Мова програмування C# поєднує у собі синтаксичну простоту з об’єктно-орієнтованою моделлю, має розвинуту екосистему бібліотек, вбудовану підтримку багатопотоковості, обробки подій та роботи з файлами. Вона також є оптимальним вибором для Windows-платформи завдяки повній інтеграції з .NET Framework.

Після завершення розробки програма буде компілюватися у виконуваний файл формату .exe, стандартний файл для застосунків у середовищі Windows. Файл .exe містить повноцінний скомпільований код разом з усіма необхідними ресурсами, що дозволяє запускати програму без потреби у встановленні додаткового програмного забезпечення або залежностей. Оскільки C# є компільованою мовою, можливість скомпілювати .exe є ще однією перевагою серед інших мов програмування.

Для реалізації проекту було обрано сучасні програмні засоби, які забезпечують зручність розробки, керування залежностями та налагодження коду. Основним інструментом середовища розробки виступає Visual Studio — інтегрована середа, яка надає повний набір функцій для роботи з мовою C# та платформою .NET. Visual Studio забезпечує автоматичну компіляцію, покрокове налагодження, візуальне конструювання інтерфейсу у WinForms, а також підтримку систем контролю версій Git.

Для керування зовнішніми бібліотеками та модулями застосовується система пакетного менеджменту NuGet. Вона дозволяє швидко підключати сторонні компоненти. Під час розробки алгоритму, вона буде використана для завантаження обгортки для роботи з OpenGL. Ця система також автоматично слідкує за сумісністю версій та залежностями, а це значно підвищує зручність розробки, забезпечує можливість у використанні додаткових інструментів і спрощує підтримку та оновлення проекту.

На комп’ютері користувача повинна бути встановлена операційна система Windows 7, Windows 8, Windows 10, Windows 11. Програма

розроблена з урахуванням архітектури Windows і використовує її внутрішні механізми управління вікнами, ресурсами та пам’яттю. Системи на базі Linux або macOS не підтримуються без застосування спеціалізованих емуляторів або віртуального середовища.

На пристрої повинно бути встановлено .NET Framework версії 4.7 або вище, для виконання C#-додатків, доступу до стандартних бібліотек та підтримку роботи з формами, графікою й потоками. У разі відсутності цієї платформи гра просто не запуститься на комп’ютері.

Для коректної роботи рендерингу дошки гри за допомогою OpenGL, потрібна підтримка драйвера відеокарти з реалізацією OpenGL не нижче версії 2.1, щоб були гарантії сумісності з базовим API для виводу двовимірної графіки без апаратних конфліктів.

Під час розробки гри необхідно враховувати всі особливості правил Го. Необхідно реалізувати можливість вибору розміру ігрової дошки: 9×9, 13×13 або 19×19. Ігрова сітка, дошка, камені та фон мають рендеритися за допомогою OpenGL відповідно до вибраного параметра. Передбачається додавання текстур дошки та рендерингу тексту для покращення візуальної складової інтерфейсу. Графіка реалізується у тривимірному просторі (3D), із урахуванням перспективи, тіней і освітлення.

Гравці виконують ходи по черзі, першими грають чорні камені. Час на гру не обмежується — тривалість партії визначається самими учасниками. Відповідно до правил гри, у разі, якщо камінь втрачає усі дихання, він знімається з поля. Реалізовано обробку ситуацій відповідно до правила самогубства та правила ко. Тобто забороняється хід у позицію без дихань та забороняється відновлення попередньої конфігурації дошки за один хід.

У вікні гри має бути передбачено можливість відображення статистики партії, а саме: поточної кількості захоплених територій кожним із гравців, а також історії зроблених ходів. Це буде реалізовано через доступ по кліку на відповідну кнопку.

Передбачено можливість пропуску ходу. У разі двох пасів поспіль партія завершується. Після завершення виконується підрахунок захоплених територій — переможцем вважається той гравець, який контролює більшу кількість перетинів ліній сітки.

Також заплановано реалізувати функціонал збереження та завантаження гри. Збереження здійснюється у базі даних після заповнення відповідної форми. Зберігається ім’я гравця, що грає білими каменями, ім’я гравця, що грає чорними, розмір ігрового поля та повна історія ходів.

Окрема форма повинна надавати можливість переглядати список збережених ігор, та доступ до функції завантаження обраної партії, очищення історії збережених ігор і створення нової гри.

Висновки до розділу 1

У даному розділі було сформульовано основну задачу проекту та проведено аналіз. Розглянуто правила гри Го, які впливають на архітектуру програмної реалізації, проаналізовано приклад реалізації гри від компанії Popoko як орієнтир для визначення функціональних можливостей майбутнього застосунку, визначено вимоги до обчислювального середовища користувача, обґрунтовано вибір програмних засобів для розробки, а також встановлено технічні та функціональні вимоги до самої гри.

РОЗДІЛ 2
ПРОЕКТУВАННЯ ЗАДАЧІ

Метою проекту є створення програмного забезпечення для моделювання гри Го, що включає графічне відображення дошки та каменів, логіку гри, облік очок, обробку ходів, підтримку взаємодії з базою даних для збереження ігор та гравців. Програма має забезпечити інтуїтивний інтерфейс для користувачів, підтримку взаємодії у режимі гри між двома гравцями за однією машиною, з можливістю збереження та відновлення гри.

Програма побудована за модульним принципом, що дозволяє ізолювати окремі функціональні частини: графічне відображення, логіка гри, обробка подій, збереження/завантаження даних, управління станом гри. Для реалізації поставленої мети задача розбивається на кілька взаємопов’язаних підзадач, кожна з яких виконує визначену функцію у загальній архітектурі програми.

При запуску програми користувач має можливість або розпочати нову гру, або завантажити раніше збережену гру з бази даних, щоб продовжити партію, яка була перервана раніше. Якщо користувач обирає нову гру, він може вказати розмір дошки (9×9, 13×13 або 19×19). Стан дошки ініціалізується порожнім, а перший хід за гравцем у якого чорні камені.

У процесі гри відбувається обробка ходів гравців. Програма приймає координати клітинки, на яку гравець хоче поставити камінь, і перевіряє коректність цього ходу відповідно до правил гри Го. Перевіряються такі умови: клітинка повинна бути вільною, хід не повинен бути самогубним або порушувати правило ко. Якщо хід допустимий, камінь ставиться на дошку, після чого перевіряється захоплення каменів супротивника. Якщо група каменів не має свободи дихання, вона вилучається з дошки, а захоплені камені рахуються.

Графічне зображення гри постійно оновлюється, щоб відображати актуальний стан дошки, каменів та інформацію про поточного гравця і кількість захоплених каменів. Після кожного ходу відбувається фіксація стану

гри, зміна активного гравця, а також перевірка умов завершення партії. Гра завершується, коли обидва гравці поспіль пропускають хід або коли дошка повністю заповнена. У кінці гри здійснюється підрахунок очок за територію та захоплені камені, і визначається переможець.

Програма підтримує збереження стану гри у базу даних. У будь-який момент поточна гра зберігається і пізніше її можна відновити її з точністю до останнього ходу та позиції каменів.

Взаємодія користувача з грою забезпечується через обробку подій: кліки по дошці для вибору клітинки, меню для збереження, завантаження, початку нової гри або виходу з програми. Всі ці дії супроводжуються відповідним оновленням інтерфейсу.

Структурувати алгоритм можна наступним чином:

1. Ініціалізація гри
    a. Вибір завантаження раніше збереженої гри з бази даних через форму.
    b. Якщо вибрана нова гра — користувач обирає розмір дошки.
    c. Встановлюється початковий стан дошки або відновлюється зі збережених даних.
    d. Активний гравець визначається автоматично на основі збереженого стану або задається у процесі гри.
2. Обробка ходів гравців
    a. Приймаються координати клітинки, куди гравець хоче поставити камінь.
    b. Перевіряється коректність ходу за правилами.
    c. Оновлюється стан дошки, розміщується камінь.
    d. Перевіряється захоплення каменів суперника та вилучаються відповідні групи.
    e. Обліковується кількість захоплених каменів.
3. Відображення поточного стану гри
    a. Оновлюється графічне відображення дошки та каменів.

b. Відображається інформація про поточного гравця та кількість захоплених каменів.
c. Відображається історія ходів.

4. Завершення ходу та передача ходу іншому гравцю
    a. Фіксується поточний стан гри.
    b. Змінюється активний гравець.
    c. Перевіряються умови закінчення гри.
    d. Якщо гра завершена — підраховуються очки і оголошується результат.
5. Збереження та відновлення гри
    a. Зберігається поточний стан гри у локальну базу даних.
    b. Відновлюється збережена гра з бази даних із усіма параметрами.
6. Інтерактивність та обробка подій користувача
    a. Обробляються кліки миші по ігровій дошці.
    b. Обробляються команди меню збереження, завантаження, початок нової гри.
    c. Оновлюється інтерфейс відповідно до дій користувача.

Опис схеми алгоритму основного циклу гри (Рис. 2.1.):

1. Початок гри
    a. Надати користувачу вибір завантажити існуючу гру або розпочати нову.
    b. Якщо вибрано завантаження — відновити стан гри з бази даних.
    c. Якщо нова гра — ініціалізувати дошку за вибраним розміром.
    d. Визначити активного гравця.
2. Основний цикл гри
    a. Очікувати хід активного гравця.
    b. Отримати координати вибраної клітинки.
    c. Перевірити легітимність ходу.
    d. Якщо хід некоректний — чекати новий хід.
    e. Якщо коректний — розмістити камінь на дошці.

f. Видалити захоплені камені суперника.

g. Оновити інтерфейс.

h. Перевірити умови завершення гри.

i. Якщо гра завершена — підрахувати очки і оголосити результат.

j. Інакше перейти до наступного гравця.

3. Підрахунок очок

a. Порахувати територію та захоплені камені.

b. Визначити переможця.

4. Збереження та відновлення гри

a. При команді збереження зберегти стан.

b. При завантаженні відновити стан і продовжити гру.

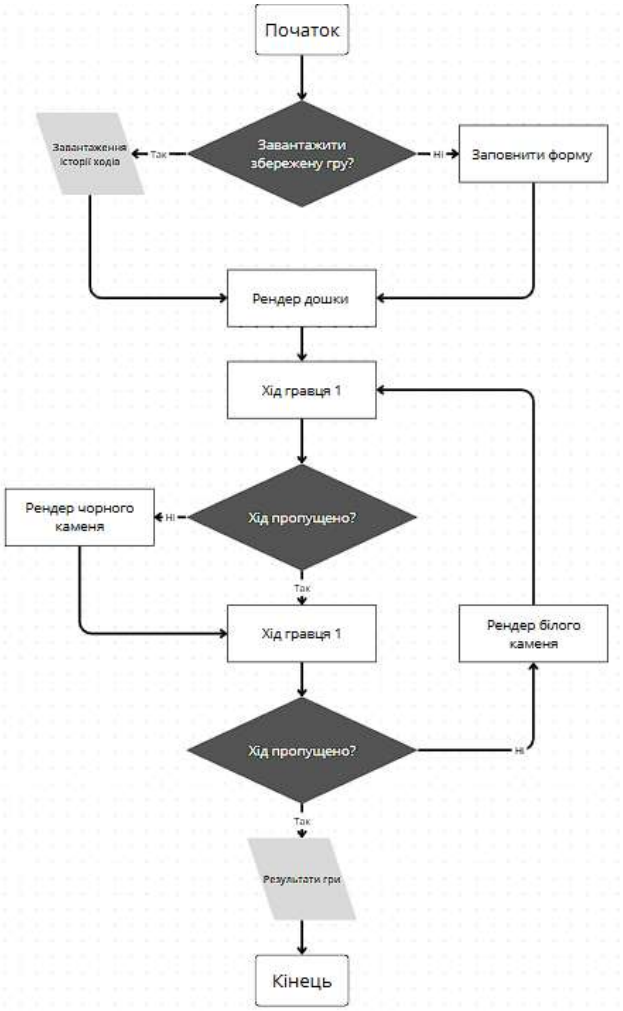

Рис. 2.1. Алгоритм програми

Розподіл логіки гри на окремі частини робить процес гри зрозумілим і зручним. Кожен етап має свою чітку роль, що дозволяє програмі працювати

послідовно та без збоїв. Завдяки цьому користувач бачить логічний хід гри, а програма вчасно реагує на його дії.

Висновки до розділу 2

У цьому розділі було здійснено проектування алгоритму програмного забезпечення для моделювання гри Го. Визначено основні етапи функціонування програми — від ініціалізації гри та обробки ходів гравців до оновлення інтерфейсу, збереження стану та завершення партії. Було описано структуру логіки гри з урахуванням її специфічних правил, а також передбачено взаємодію з користувачем через графічний інтерфейс і механізми обробки подій. Проектування алгоритму допомогло розмежувати функціональні компоненти програми. Алгоритм було розбито на модулі, щоб спростити подальшу реалізацію та підтримку системи.

РОЗДІЛ 3
ПРОЕКТУВАННЯ БАЗИ ДАНИХ ДЛЯ ПРОЕКТУ

3.1. Вибір системи управління базами даних

Наступним етапом розробки програмного забезпечення для гри Го стало впровадження системи збереження ігрових даних. Ці дані включають інформацію про гравців, розмір ігрової дошки, поточну конфігурацію каменів, історію ходів, кількість захоплених каменів, активного гравця, а також дату й час збереження. Для зберігання цієї інформації потрібно вірно підібрати систему управління базами даних (СУБД), яка дозволить ефективно працювати з невеликими обсягами структурованої інформації у локальному режимі.

У якості СУБД було обрано SQLite — легку реляційну базу даних, що не потребує серверної частини та зберігає всі дані у вигляді одного файлу на диску [9]. Такий підхід є особливо зручним для настільних застосунків або ігор, де очікується автономна робота без підключення до інтернету чи до зовнішніх баз даних. Завдяки простоті розгортання, SQLite дозволяє уникнути зайвої складності, пов’язаної з налаштуванням сервера баз даних, прав доступу чи мережевої взаємодії.

Однією з переваг SQLite є її повністю вбудована архітектура. Це означає, що база не функціонує як окремий процес або служба, а виконується безпосередньо всередині програми. Внаслідок цього всі операції з даними — зчитування, запис, оновлення — відбуваються надзвичайно швидко, що критично важливо для забезпечення плавного ігрового процесу. Особливо це відчутно під час частого збереження та завантаження стану гри, коли затримки навіть у кілька секунд можуть негативно позначитись на зручності використання.

SQLite повністю підтримує мову структурованих запитів SQL, що спрощує реалізацію логіки взаємодії з базою даних у програмному коді.

Створення таблиць, додавання записів, вибірка і фільтрація даних реалізуються через зрозумілий та стандартизований інтерфейс, добре документований і знайомий більшості розробників. Це дає можливість легко оновлювати структуру бази або розширювати функціональність у майбутньому без радикальної зміни архітектури програми. Також варто відзначити, що SQLite є повністю безкоштовною та відкритою технологією, яка не вимагає ліцензійних платежів.

З погляду безпеки, хоча SQLite не має вбудованих механізмів для багатокористувацького захисту чи шифрування, це не є суттєвим недоліком у межах настільної гри, де дані зберігаються на одній машині і використовуються лише однією програмою. За потреби забезпечення додаткового рівня захисту можна використовувати стороннє шифрування файлів бази або обмеження доступу до каталогу збережень на рівні операційної системи.

Для забезпечення збереження та відновлення даних гри Го, у застосунку використовуються SQL-запити — інструкції до бази даних, які дозволяють отримувати, додавати, оновлювати або видаляти дані. Оскільки обрана СУБД — SQLite, необхідно ознайомитися з мовою запитів SQL, та їхньою структурою. Структура запитів залишається загальною для більшості сучасних баз.

Мова SQL поділяється на кілька категорій запитів, залежно від мети їх використання: запити вибірки, модифікації, визначення структури даних та управління доступом. Найчастіше у прикладному програмуванні працюють з так званими DML-запитами (Data Manipulation Language), які забезпечують доступ до існуючих даних і зміну їх стану. У грі Го зберігаються вся інформація про партії, ходи, статус гравців і розміщення каменів, запити цього типу дозволяють динамічно формувати внутрішній стан застосунку на основі даних, що отримуються з бази.

CREATE TABLE створює таблиці, які зберігають основні сутності гри — гравців, ігрові сесії, ходи, статус дошки. Визначення таблиці включає

перелік полів, їх типи (INTEGER, TEXT, REAL тощо), первинні та зовнішні ключі, обмеження цілісності (NOT NULL, UNIQUE, FOREIGN KEY), що гарантують валідність структури даних. Створення правильної схеми — базова умова коректної роботи будь-якої взаємодії з базою.

Команда SELECT є фундаментом будь-якої операції вибірки. Її функціональність розширюється за допомогою додаткових конструкцій — WHERE фільтрує записи за умовами, JOIN дозволяє об’єднувати таблиці, ORDER BY визначає порядок сортування, а LIMIT обмежує обсяг даних, що повертаються. У проекті реалізовано запити, які вибирають останню гру користувача, усі попередні ходи конкретної партії, а також стан дошки після певного ходу, щоб можна було відновити гру до будь-якого моменту або аналізувати ігровий процес.

Використовуються і агрегатні функції у запитах типу SELECT. Наприклад, COUNT використовується для підрахунку кількості зроблених ходів у певній грі, MAX дозволяє визначити найвищий номер ходу для відновлення останнього стану дошки, а функція SUM застосовується для обчислення загальної кількості захоплених каменів. Ці функції обробляють цілі множини рядків, повертаючи єдиний узагальнений результат, який необхідний для аналітики та візуалізації.

У ситуаціях, коли потрібне групування результатів, застосовується конструкція GROUP BY. Вона дозволяє згрупувати всі ходи за іграми або гравцями й обчислити статистику — середню кількість ходів у партії, частоту перемог певного гравця. У поєднанні з HAVING ці групи можуть бути відфільтровані за додатковими критеріями, що відкриває можливості для побудови складніших звітів.

Можливість побудови умовних запитів за допомогою логічних операторів AND, OR, NOT у секції WHERE також розглядається. Це дозволяє описувати вибіркові умови, вибрати всі ігри, які тривають, і при цьому розпочаті після певної дати. Запит можна застосувати для підтримки логіки

фільтрації в інтерфейсі користувача й адаптувати до різних сценаріїв запиту інформації.

Ще один тип запитів — INSERT — відповідає за фіксацію нових даних. У грі після кожного ходу створюється запис, який містить ідентифікатор гри, координати каменя, його колір, номер ходу, а також мітку часу. Такі дані необхідні для повного відтворення послідовності дій. Незважаючи на простоту синтаксису команди, важливо забезпечити узгодженість усіх полів із структурою таблиць, враховуючи типи даних та ключі.

Для оновлення поточного стану використовуються запити UPDATE. Вони дозволяють вносити зміни до вже існуючих записів. Використовуються для оновлення кількості захоплених каменів, статус гри (активна або завершена), або інформацію про переможця. У таких запитах важливо правильно формулювати умови оновлення, аби уникнути зміни нецільових рядків, що може призвести до логічних помилок або втрати даних.

Можливість видалення збережених ігор є однією із поставлених вимог до програми. Для цього треба використати команду DELETE. Вона дозволяє видаляти записи з бази даних відповідно до заданого критерію. У контексті настільного застосунку, де дані зберігаються локально, ця операція дозволяє користувачу самостійно керувати сховищем, очищаючи непотрібні або застарілі сесії гри.

Особливістю роботи із SQLite є те, що вона не потребує попередньої конфігурації, а сама база існує у вигляді одного файлу на диску. Це означає, що усі SQL-запити взаємодіють із внутрішнім представленням даних без участі окремого сервера.

3.2. Структури таблиць бази даних

Проект гри Го передбачає не лише реалізацію внутрішньої логіки та інтерфейсу, а й передбачає можливість збереження ігрових даних. Для цього було створено повноцінну реляційну структуру бази даних, що моделює всі

основні сутності гри, їх взаємозв’язки та можливість розширення функціональності в майбутньому. Архітектура бази даних відповідає підходу до нормалізованого зберігання інформації, що дозволяє уникнути дублювання, підвищити цілісність і забезпечити логіку зв’язків даних.

База складається з п’яти основних таблиць: Players, Games, Moves, Comments та GameTags. Кожна пов’язана з іншими таблицями через зовнішні ключі, які гарантують референційну цілісність. Розглянемо детальніше. Таблиця Games містить зовнішні ключі на гравців (чорного та білого), а всі ходи (Moves), коментарі (Comments) і теги (GameTags) напряму прив’язуються до певної гри, забезпечуючи ієрархічну прив’язку інформації до ігрового сеансу.

Таблиця Players є найпростішою з погляду структури — вона містить лише ідентифікатор гравця та його ім’я. Цього мінімального набору цілком достатньо для зберігання унікальних особистостей учасників партій. Завдяки виділенню гравців в окрему таблицю система може відстежувати участь одного користувача в декількох іграх, вести статистику або надавати фільтрацію за конкретним гравцем.

Центральним вузлом структури є таблиця Games. Вона містить зовнішні ключі на обох гравців, а також часову мітку створення гри. Саме через поле Id цієї таблиці встановлюються всі подальші зв’язки — з ходами, коментарями, тегами. Така структура дозволяє безпосередньо отримати всі пов’язані з грою дані, що робить роботу з нею зручною та логічно цілісною.

Таблиця Moves моделює похідну динаміку гри. Кожен рядок тут відповідає одному ходу: з координатами на дошці (X, Y), кольором каменя, номером ходу та ідентифікатором гри. Завдяки цьому можна відтворити весь процес партії, намалювати каміння на вірних місцях на дошці, зберегти логіку черговості. Окреме поле move_number є особливо корисним для побудови аналітики, порівняння ходів, відновлення гри з проміжного етапу.

Додаткові функціональні можливості реалізуються через таблицю Comments, яка дозволяє прикріплювати текстові примітки до певних ходів

конкретної гри. Це відкриває перспективу створення режиму навчання або аналізу гри з поясненням тактичних рішень. Коментарі прив’язані до номера ходу, щоб можна було легко поєднати текст з візуальним станом дошки.

Окрема таблиця GameTags додає систему категоризації — кожен запис містить ідентифікатор гри та короткий тег. Теги можуть включати ключові слова, стилі гри, оцінки якості партії, теми навчання або назви турнірів. Ця таблиця введена в структуру бази даних, щоб забезпечити можливість гнучкого пошуку, фільтрації та організації великої кількості збережених ігор.

Технічно структура спроектована з урахуванням цілісності даних. Усі зовнішні ключі супроводжуються опцією ON DELETE CASCADE, яка гарантує автоматичне очищення залежних даних при видаленні основного запису. Це спрощує логіку підтримки бази в актуальному стані, не вимагаючи ручної синхронізації між таблицями (табл. 3.1).

Таблиця 3.1

Структура бази даних

Назва таблиці	Поле	Тип даних	Опис поля
Players	Id	INT AUTO_INCREMENT	Унікальний ідентифікатор гравця
	Name	VARCHAR(100)	Ім'я гравця
Games	Id	INT AUTO_INCREMENT	Унікальний ідентифікатор гри
	PlayerBlackId	INT	Ідентифікатор гравця, що грає чорними (зовнішній ключ)
	PlayerWhiteId	INT	Ідентифікатор гравця, що грає білими (зовнішній ключ)
	CreatedAt	DATETIME	Дата і час створення гри

Продовження таблиці 3.1

Moves	Id	INT AUTO_INCREMENT	Унікальний ідентифікатор ходу
	GameId	INT	Ідентифікатор гри (зовнішній ключ)
	move_number	INT	Номер ходу в рамках гри
	X	INT	Координата по осі X на дошці
	Y	INT	Координата по осі Y на дошці
	Color	INT	Колір каменя (наприклад, 0 — чорний, 1 — білий)
Comments	Id	INT AUTO_INCREMENT	Унікальний ідентифікатор коментаря
	GameId	INT	Ідентифікатор гри (зовнішній ключ)
	MoveNumber	INT	Номер ходу, до якого відноситься коментар
	Text	TEXT	Текст коментаря
GameTags	Id	INT AUTO_INCREMENT	Унікальний ідентифікатор тегу
	GameId	INT	Ідентифікатор гри (зовнішній ключ)
	Tag	VARCHAR(50)	Текстове позначення тегу

3.3. Запити необхідні для алгоритму

У процесі реалізації логіки гри Go необхідно сформулювати запити для того, щоб код міг взаємодіяти з базою даних. Запити повинні дозволяти зберігати хід гри, завантажувати попередні ігри, видаляти обрані ігри, обробляти інформацію про гравців, а також забезпечувати коректне продовження або завершення гри. У цьому підрозділі пропоную розглянути основні SQL-запити, які використовуються в додатку, а також їх функціональне призначення у рамках загального алгоритму гри.

Першим побудованим запитом є створення нової гри. За допомогою SQL-операції INSERT INTO Games у таблицю з іграми записуються імена обох гравців, щоб зберегти сесію гри та присвоїти їй унікальний ідентифікатор. У подальшому цей ідентифікатор використовується для асоціювання ходів з відповідною грою.

Другий запит — це збереження окремого ходу гри. Запит INSERT INTO Moves дозволяє фіксувати кожен хід, включаючи його порядковий номер, координати на дошці та колір каменя, щоб повністю відтворити гру при завантаженні та продовженні.

Окремим запитом реалізується збереження повного стану гри. Для цього спочатку використовується DELETE FROM Moves для очищення попередніх ходів конкретної гри, після чого всі нові ходи записуються у базу.

Завантаження всіх ходів певної гри (SELECT MoveNumber, x, y, color FROM Moves) виконується при виборі збереженої гри для перегляду або продовження. Завдяки ньому алгоритм гри отримує повну інформацію про попередній перебіг подій.

Запит для отримання списку всіх ігор (SELECT Id, PlayerBlack, PlayerWhite, CreatedAt FROM Games) дозволяє реалізувати функціонал перегляду та вибору гри користувачем. Також доступний варіант із приєднанням інформації про гравців з іншої таблиці за допомогою JOIN.

Додатково реалізовані запити для обробки гравців SELECT Id FROM Players WHERE Name = @name. Він дозволяє знайти гравця в базі, а якщо його немає — створити нового, використовуючи INSERT INTO Players, щоб передбачити створення унікального гравця в системі, та уникнути дублювання інформації.

Також у програмі передбачена можливість видалення гри (DELETE FROM Games WHERE Id = @id), для керування збереженими даними й очищення непотрібної інформацію.

Усі запити створені для взаємодії з алгоритмом гри: вони не тільки забезпечують збереження прогресу, а й дозволяють реалізувати функції кнопок "Зберегти гру", "Завантажити гру", "Почати нову гру", "Переглянути історію".

Висновки до розділу 3

У розділі було розглянуто вимоги до зберігання інформації, необхідної для функціонування гри. Основну увагу було приділено збереженню ігрового процесу — поточного стану дошки, послідовності ходів та ідентифікації окремих ігор. Було проаналізовано та обрано відповідну систему управління базами даних, яка забезпечує збереження і доступ до даних. Розглянута СУБД дозволяє реалізовувати всі необхідні запити до бази — зчитування, збереження та оновлення інформації про гру. Також були створені SQL-запити для вибірки збережених ігор, запису нових ходів та збереження повного стану гри. Завдяки цьому в коді можна реалізувати функціонал завантаження попередньої гри, а також продовження гри з останнього збереженого моменту.

РОЗДІЛ 4
РОЗРОБКА ГРИ

4.1. Побудова інтерфейсу гри

Інтерфейс гри розроблений із використанням модульної архітектури, де кожен візуальний компонент відповідає окремому класу форми. Пропоную далі розглянути процес схематичної побудови інтерфейсу.

Під час побудови графічного інтерфейсу користувача було враховано як специфіку гри Го, так і потреби гравця в легкому доступі до керування, аналізу і збереження гри. Структура інтерфейсу умовно поділяється на кілька основних блоків, кожен з яких має своє функціональне призначення, логічне розташування та візуальну ієрархію.

Головне вікно — це вікно в якому відбувається ігова партія. Тому вікно повинно містити ігрову дошку, інформаційну панель та доступ до інструментів дій, які необхідно здійснювати додатково під час партії. Центральним елементом усього інтерфейсу є ігрова дошка, яка займає основну частину вікна. Це відображає пріоритет взаємодії гравця саме з полем — у грі Го візуальне спостереження за ситуацією на дошці та точність розміщення каменів є основою. Вікно повинно автоматично адаптувати розміри ігрової області до розмірів форми, зберігаючи оптимальне співвідношення сторін (Рис. 4.1.).

У верхній частині вікна розташовується заголовкова панель. Вона виконує функції виведення поточного статусу гри, зокрема вказує, чий хід. Така інформація має бути завжди доступною, тому панель закріплена зверху і візуально відокремлена від інших елементів — темним фоном і контрастним білим шрифтом. Це не лише покращує читабельність, а й знижує когнітивне навантаження під час гри.

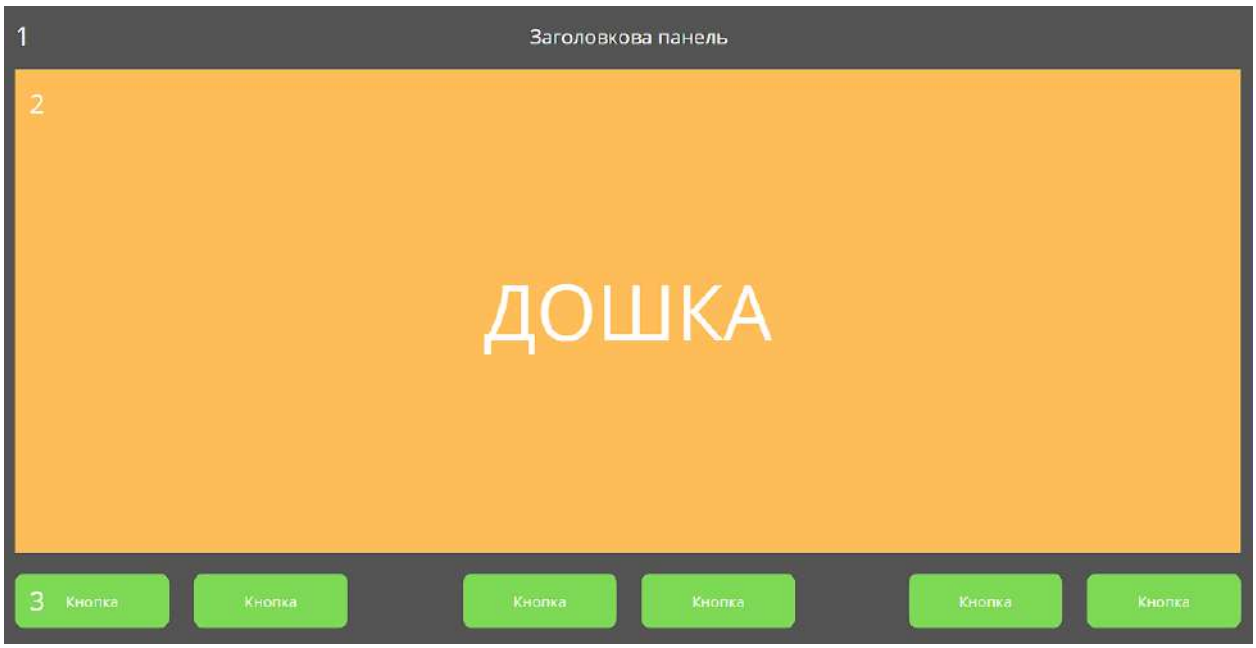

Рис. 4.1. Схема вікна гри:

1 - заголовкова панель

2 - дошка Го

3 - панель з кнопками

Нижня частина інтерфейсу складається з трисекційної панелі керування. Вона організована у вигляді таблиці з трьома стовпцями, що забезпечує візуальний поділ функцій по лозиці їх задач.

Ліва секція призначена для кнопок керування грою, має кнопки для основних дій, які прямо впливають на поточний хід (початок нової гри, "пас"). Вони розміщені першими, щоб зір користувача одразу фіксувався на важливих кнопках гри.

Центральна секція містить інструменти для аналізу гри, такі як перегляд історії ходів та оцінку охвачених територій. Ці функції не використовуються постійно, але повинні бути легко доступними на пізніших етапах гри або для ретроспективного аналізу.

Права секція відведена для дій із файлами: збереження і завантаження партій. Візуально їх оформлено інакше, що допомагає гравцеві швидко орієнтуватися у типі дії, яку він виконує.

Кожна кнопка інтерфейсу має унікальний стиль: округлі краї, контрастний напис, ефекти наведення та натискання, щоб створити сучасний стиль та зменшити ризик помилкового натискання.

Розміщення елементів у головному вікні побудовано вертикально, зверху вниз. Така траекторія відповідає природному шляху руху очей людини при скануванні інформації. Цей маршрут базується на принципах UX-дизайну та досвіді взаємодії з більшістю цифрових інтерфейсів.

У верхній частині вікна — інформаційний блок. Тут розміщується поточний стан гри, пишеться хто ходить, наприклад “Хід: Чорні”. Це перше, на що звертає увагу користувач після відкриття вікна, що дозволяє швидко зрозуміти хто ходить перший.

Центральна зона відведена під дошку — головний елемент гри. Це місце взаємодії, і воно закономірно розміщується в центрі поля зору. Саме сюди користувач буде дивитися протягом більшої частини часу.

Нижню частину займають елементи керування: кнопки дій, інструменти аналізу та управління файлами, щоб гравець міг інтуїтивно “завершувати” цикл погляду, коли після перегляду поля користувач приймає рішення і візуально переходить до кнопок взаємодії.

Додаткові вікна — створення нової гри та вибір збереженої — реалізовані у вигляді окремих форм. Вікно створення гри містить лише необхідні поля: імена чорного та білого гравця, а також кнопку підтвердження (Рис. 4.2.).

Рис. 4.2. Схема вікна нової гри

Форма вибору гри має горизонтальне розміщення списку ігор та вертикальний блок кнопок праворуч. Зліва гравець може ознайомитися зі списком, праворуч — завантажити партію, створити партію, видалити партію (Рис. 4.3.).

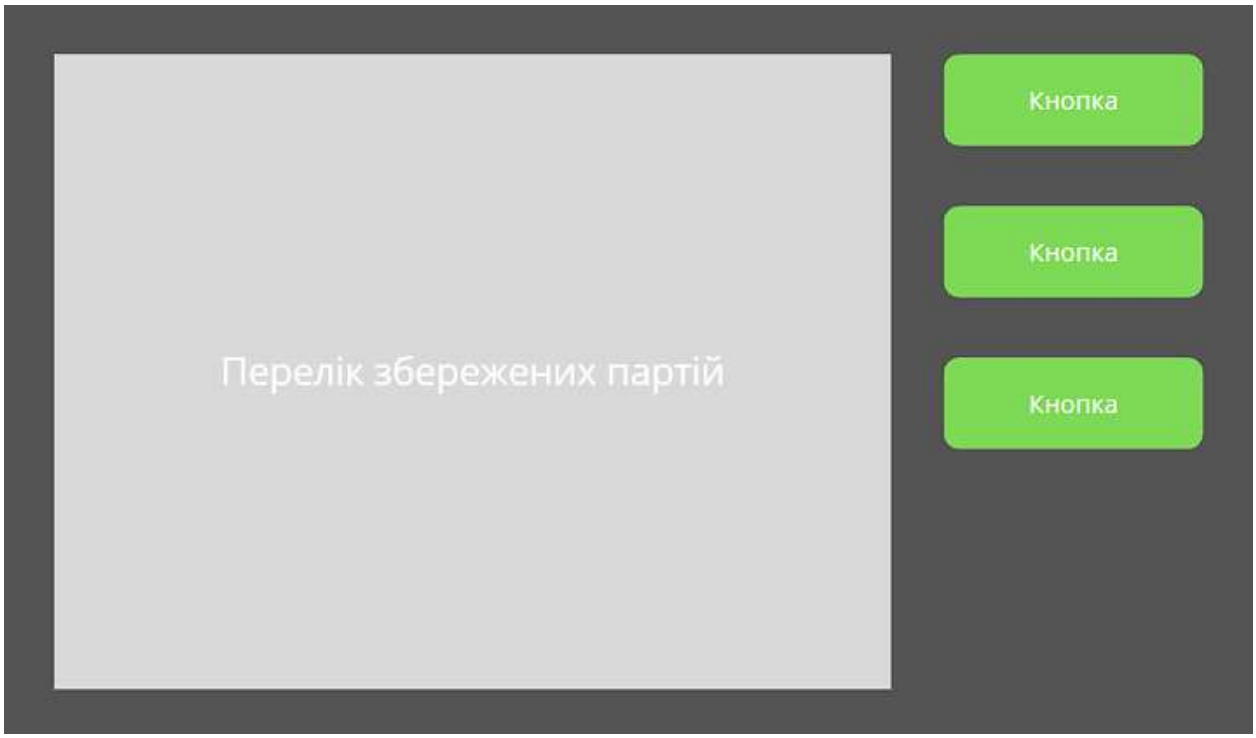

Рис. 4.3. Схема вікна завантаження гри

Діалогове вікно вибору розміру дошки виконує роль початкової конфігурації перед запуском гри. Воно реалізується у вигляді окремої

модальної форми невеликого розміру, що з’являється по центру головного вікна. В середині вікна розміщено комбобокс із фіксованим списком доступних варіантів — 9x9, 13x13 або 19x19. Після вибору розміру користувач підтверджує свій вибір кнопкою «OK», після чого ініціалізується GameEngine із відповідною конфігурацією дошки. В цьому ж місці виконуються підключення до основних подій, що забезпечують динамічне оновлення інтерфейсу під час гри. Це вікно не перевантажене елементами — лише необхідний мінімум, що дозволяє швидко перейти до основного процесу (Рис. 4.4.).

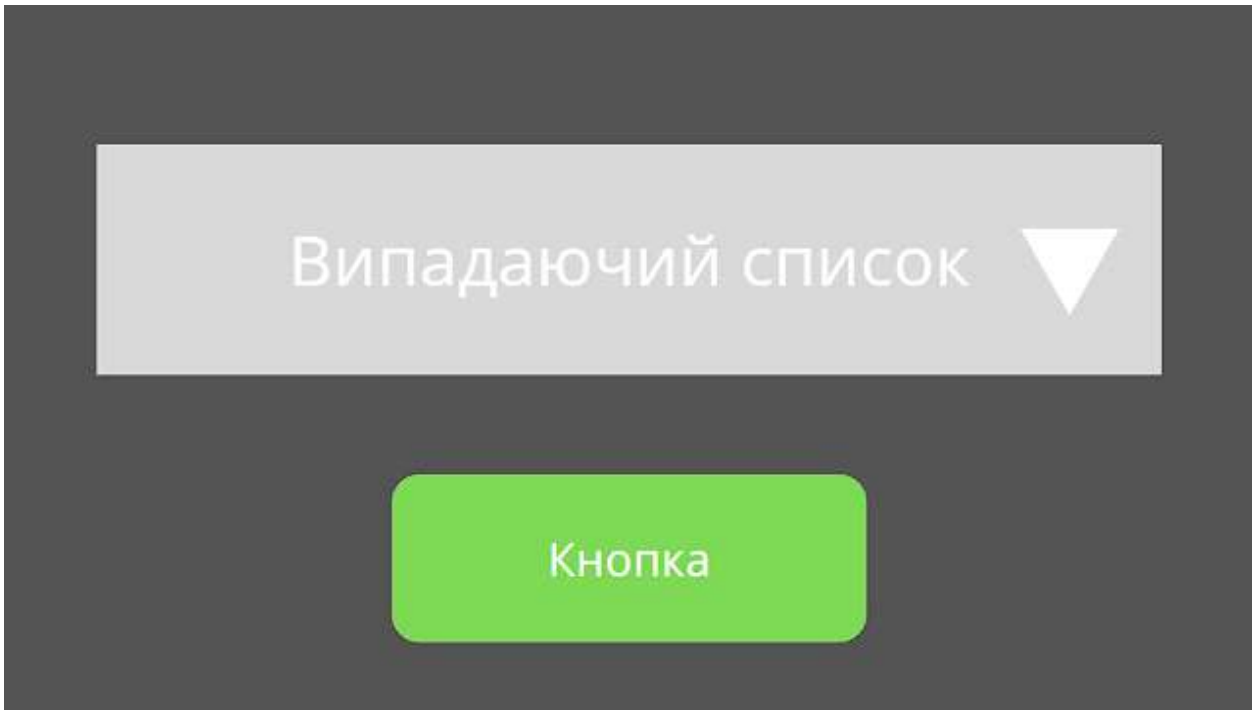

Рис. 4.4. Діалог вибору розміру поля

Кожне з цих рішень підпорядковується загальному принципу: інтерфейс має бути максимально простим, зрозумілим, не перевантаженим графічними елементами, але водночас повноцінним у функціональності. Компонування елементів дозволяє зберігати логічну послідовність дій гравця, а це позитивно впливає на користувацький досвід та зручність роботи з програмою.

Переваги структури з окремими вікнами:

1. Модульність — кожне вікно виконує одну чітку функцію: створення, вибір, перегляд або гра.

2. Користувачі не ламають систему, роблячи помилки, менше плутаються між етапами, бо кожне вікно чітко вказує, що від них необхідно зробити.
3. Фокус — відокремлення елементів допомагає уникати інформаційного перевантаження.
4. Гнучкість дизайну — компоненти можна переробляти окремо, не порушуючи загальну логіку.

Під час проектування інтерфейсу гри Го одним із принципів стала задача розробки функціональної ясності та візуального порядку. Це досягається через розподіл інтерфейсу на окремі вікна відповідно до логічних етапів взаємодії користувача з грою: підготовка, сама гра та аналіз, щоб полегшити користувачеві орієнтацію в інтерфейсі, а й сприяти зниженню когнітивного навантаження, дозволяючи фокусуватися на конкретному завданні в потрібний момент.

Після запуску програми відкривається вікно підготовки гри, де гравець вводить імена та створює нову партію. Це окреме, компактне вікно, яке має поля вводу та кнопку для запуску гри.

Далі, після запуску гри, відкривається головне вікно, де відбувається вся основна ігрова сесія. Це центральна частина інтерфейсу, яка зосереджує увагу на дошці, надає мінімально необхідну інформацію про поточний хід та керування — усе інше вже опрацьовано на попередньому етапі.

Після завершення гри або за потреби в ретроспективному перегляді, користувач може перейти до функціоналу аналізу, переглянути історії ходів чи оцінки територій. Ці дії теж винесені в окремі діалогові вікна, щоб не перевантажувати основне ігрове поле в момент гри.

У процесі розробки інтерфейсу гри Го була поставлена задача впровадження компонента GLControl, який забезпечує відображення ігрового поля за допомогою бібліотеки OpenGL. На відміну від стандартних засобів графічного відображення у Windows Forms, використання GLControl дає змогу працювати із сучасною графічною підсистемою, що значно розширює функціональні можливості і підвищує продуктивність.

GLControl є спеціалізованим елементом управління, який вбудовується у форму так само, як і звичайні елементи типу Panel чи PictureBox. Однак на відміну від них, GLControl взаємодіє з відеокартою безпосередньо через OpenGL. Це дозволяє використовувати апаратне прискорення графіки, що особливо важливо для таких динамічних задач, як інтерактивне малювання дошки, відображення фігур, а в майбутньому — і анімованих ходів або підсвічування територій.

Окрім продуктивності, використання OpenGL відкриває широкі перспективи для візуального вдосконалення інтерфейсу. Завдяки можливостям, які надає OpenGL можливо реалізувати більш складні графічні ефекти — напівпрозорі камені, згладжування ліній сітки, реалістичні тіні або градієнти.

Окрема перевага — масштабованість візуалізації. OpenGL автоматично підтримує зміну розмірів вікна без втрати якості відображення. Це дозволяє дошці залишатись чіткою, незалежно від роздільної здатності екрана. Така перевага особливо корисна в умовах, коли користувач працює з різними типами моніторів або при зміні масштабу вікна.

Також варто зазначити, що використання GLControl дає змогу використовувати більш складну логіку обробки подій миші. У поєднанні з координатною системою OpenGL, це забезпечує точну відповідність між візуальним розташуванням елементів і логікою гри. Це набагато спрощує реалізацію клікабельних зон, підсвічування активного ходу або відображення можливих ходів.

4.2. Розробка гри

Запуск програми відбувається через виконуваний файл [Додаток Б], після чого спливає вікно вибору гри. У цьому вікні відображається список раніше збережених ігор, з якого можна обрати одну для продовження (Рис. 4.5.).

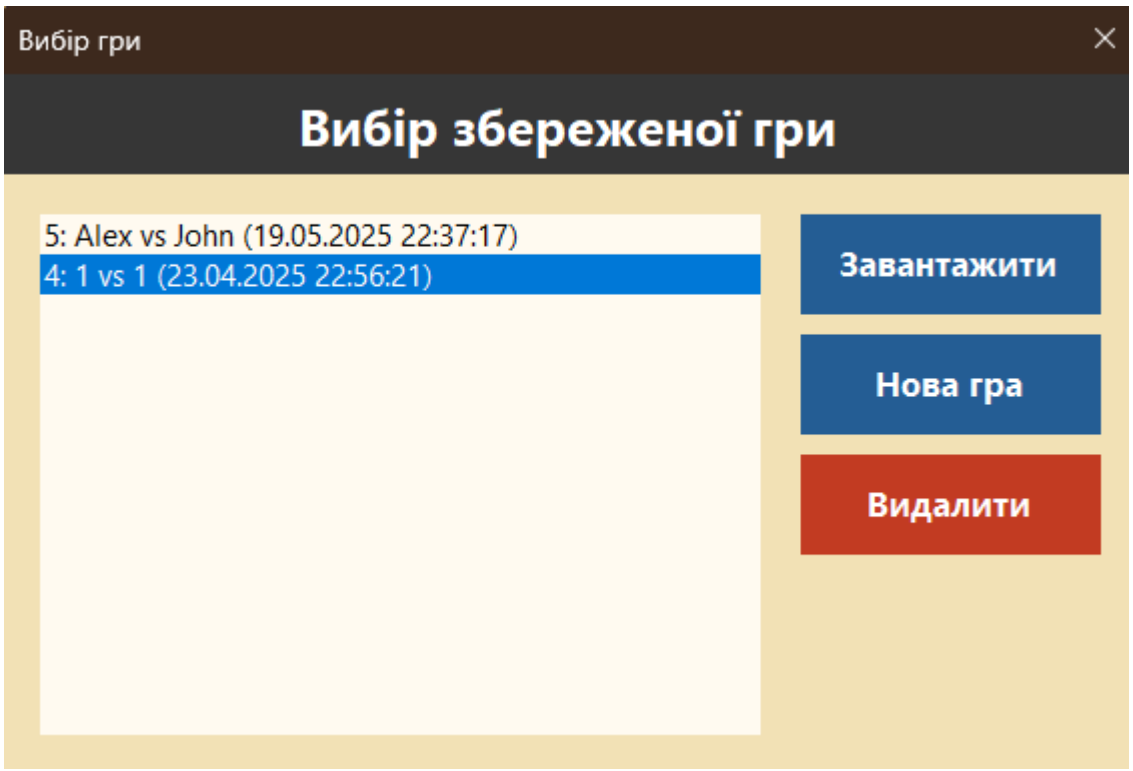

Рис. 4.5. Вибір гри

Список ігор займає основну частину вікна. Праворуч розміщено кнопки для завантаження вибраної гри, створення нової або видалення непотрібних записів.

Форма створення нової гри є компактною формою (рис. 4.6.), у якій реалізовано два текстові поля та одну кнопку. У верхній частині вікна розташований заголовок «Створення нової гри», під ним поле для введення імені чорного гравця, нижче — поле для введення імені білого. Обидва поля мають однакову ширину та вирівняні по горизонталі, забезпечуючи симетричне й чисте візуальне сприйняття. Під полями знаходиться кнопка з підписом «Створити». Після кліку у список ігор форми вибору гри додається нова гра, звідки її і можна завантажити.

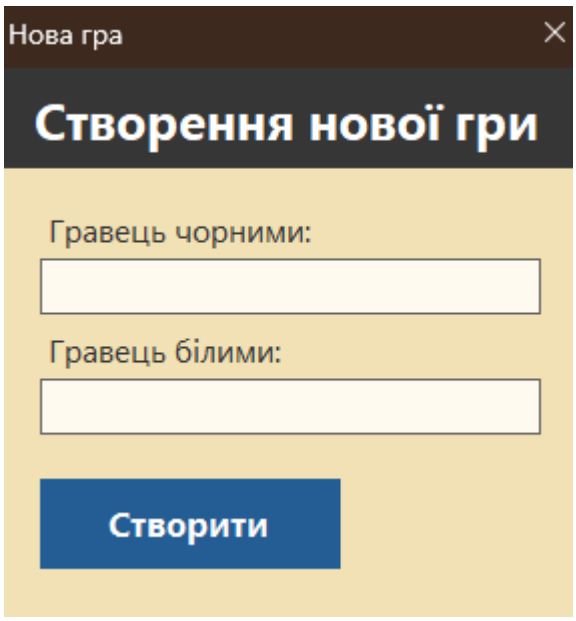

Рис. 4.6. Нова гра

Після створення гри її можна завантажити, обравши у списку та клікнувши кнопку Завантажити. Після чого з’являється діалогове вікно у якому пропонується обрати розмір дошки (Рис. 4.7.).

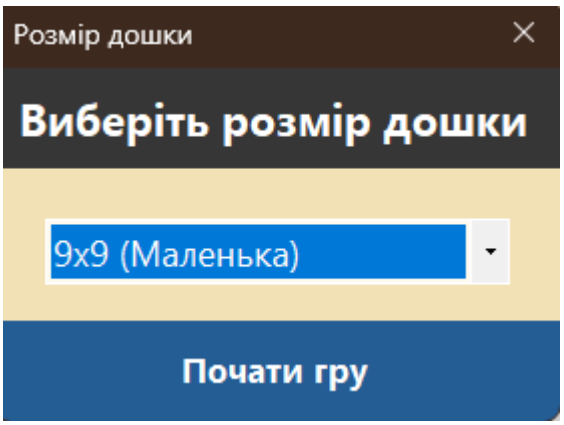

Рис. 4.7. Розмір дошки

Доступні розміри (Рис. 4.8.):

- 9 на 9 (маленька)
- 13 на 13 (середня
- 19 на 19 (стандартна)

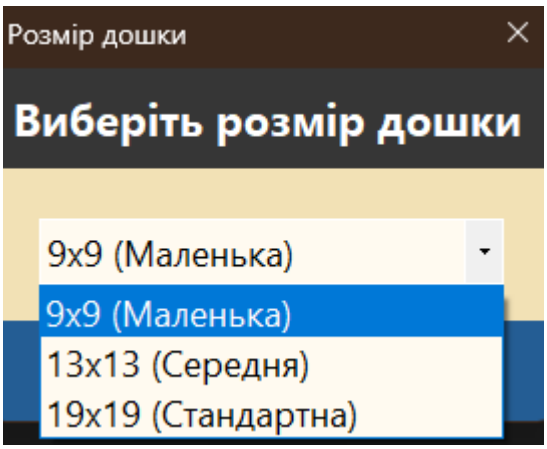

Рис. 4.8. Розмір дошки

Розмір 9 на 9 найчастіше використовується для швидких партій, навчання або гри з новачками. В інтерфейсі така дошка виводиться з більшими проміжками між лініями сітки, що створює візуально просторе і менш навантажене поле. Камені займають значну частину клітинки, тому легше візуально спостерігати за кожним ходом. Завдяки меншій кількості перетинів, дошка рендериться швидше, а координати ходів сприймаються простіше, бо їх меньше (Рис. 4.9.).

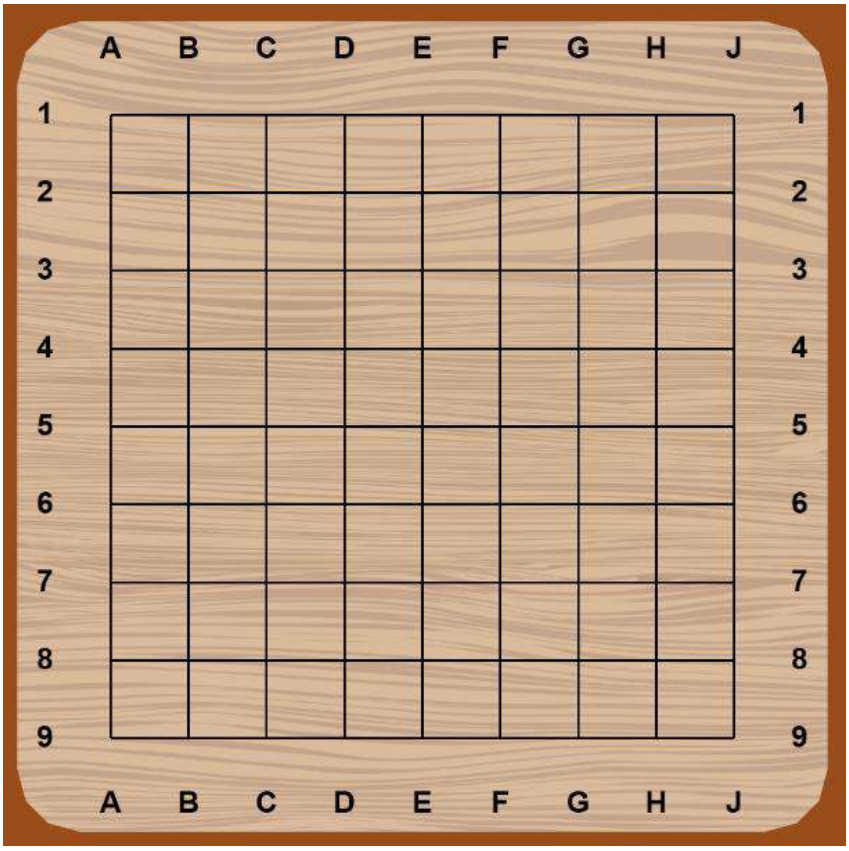

Рис. 4.9. Дошка 9 на 9

Середній розмір 13 на 13, який поєднує в собі баланс між стратегічною грою та візуальною зрозумілістю. Під час відображення цього формату застосовується помірний зум, щоб зберегти достатню видимість ігрових елементів, не втрачаючи при цьому компактності. Камені стають дещо меншими, а сітка — щільнішою (Рис. 4.10.).

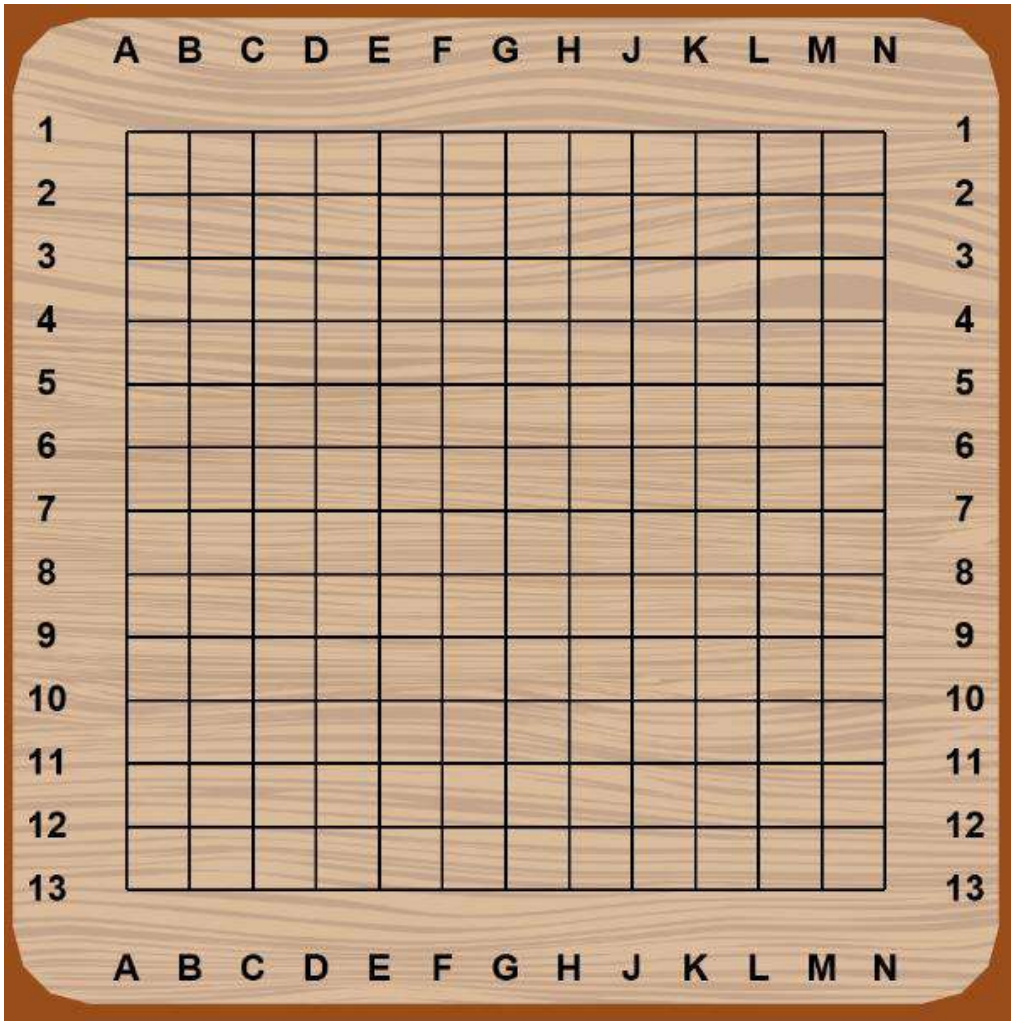

Рис. 4.10. Дошка 13 на 13

19 на 19 стандартний розмір для гри Го, який використовують у змаганнях. Відображення дошки такого формату вимагає максимального ущільнення сітки для збереження пропорцій у межах вікна. Камені відображаються дрібніше, а клітинки — компактніше, що вимагає точнішої роботи з рендерингом. Щоб уникнути візуального "злипання", використовуються точні розрахунки відступів, а також адаптивне позиціонування, щоб дошка залишалася читабельною навіть на меншому екрані (Рис. 4.11.).

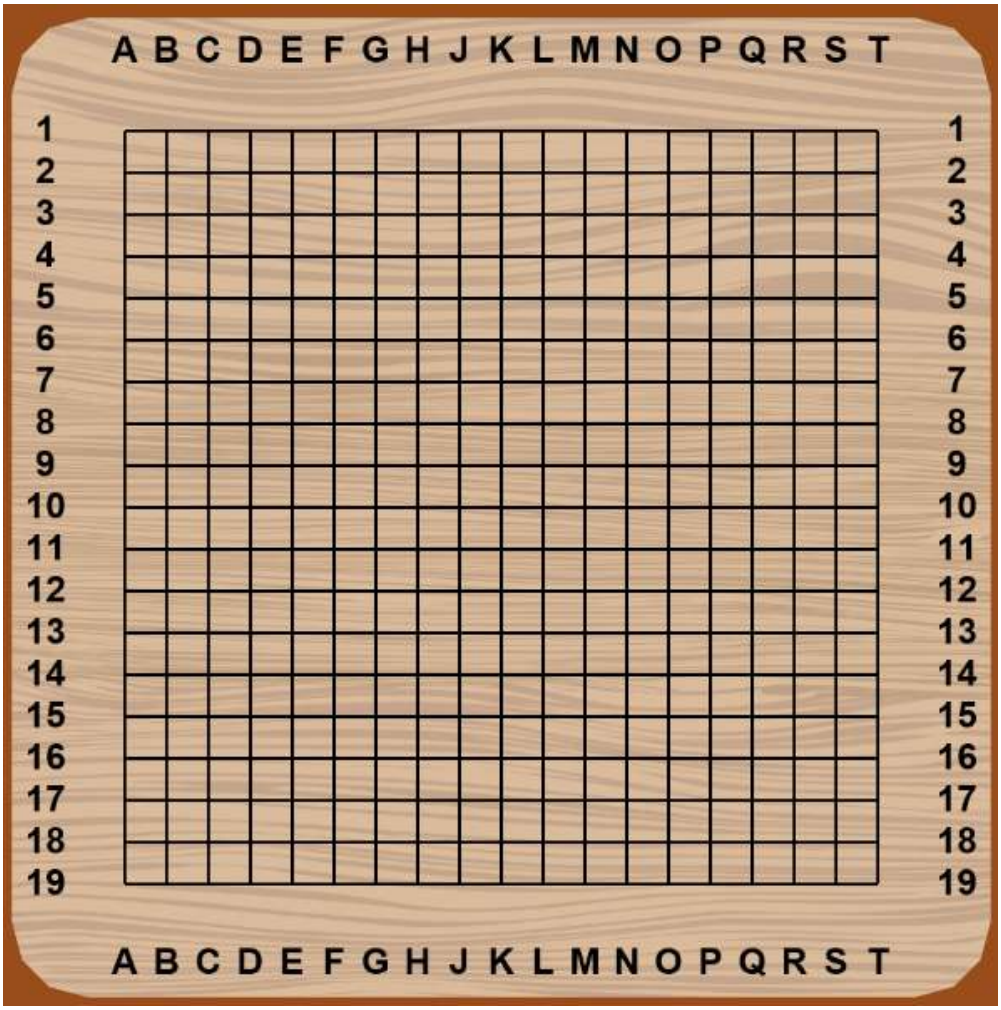

Рис. 4.11. Дошка 19 на 19

Завдяки використанню OpenGL через GLControl, дошка рендериться у тривимірному просторі з текстурами та світлом, що створює ефект справжньої гравіюваної поверхні.

Дошка має закруглені кути, що додає м’якості та реалістичності формі. Це не просто плоский прямокутник — контури згладжені, що візуально нагадує справжню гравіру дерев’яну дошку для гри Го.

Поверхня дошки покрита текстурою світлого дерева — це візуально теплий, натуральний матеріал, який надає глибини і затишку інтерфейсу. Текстура підвантажується окремим зображенням і накладається на геометрію, забезпечуючи ефект фотореалістичності, який підсилюється підсвіткою сцени за допомогою джерела світла.

Для зручності орієнтації на полі, по краях дошки відображаються координати — латинські літери (без I, згідно з традицією) для стовпців і цифри

для рядків. Ці маркери рендеряться окремо і масштабуються відповідно до розміру дошки, розміщуються симетрично з усіх боків.

Саме вікно окрім дошки має елементи інтерфейсу (Рис. 4.12.), які змінюються у процесі гри та з якими користувач взаємодіє під час гри [Додаток А]. У верхній частині вікна розміщується заголовок, який відображає поточного гравця. Текст оновлюється на «Хід чорного» або «Хід білого», автоматично після кожного ходу.

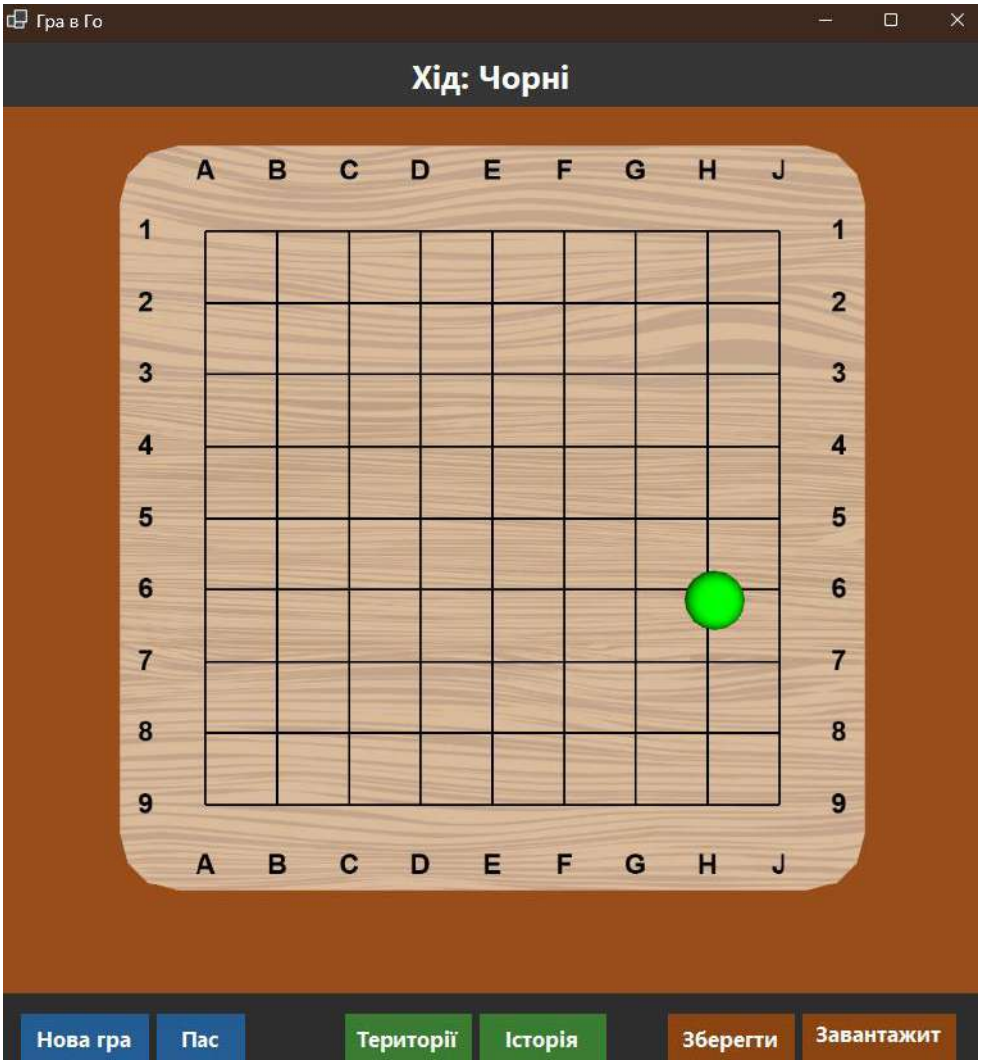

Рис. 4.12. Вікно гри

Центральну частину вікна займає поле гри. В нижній частині форми розміщена панель з кнопками, що дозволяє виконувати ключові дії в грі:

- Нова гра
- Пас
- Території

- Історія
- Зберегти
- Завантажити

Під час розстановки ходу на дошці курсор миші виконує роль попереднього перегляду майбутнього каменя (Рис. 4.13.). У момент, коли гравець наводить мишку на дошку, на цьому місці з’являється зелений напівпрозорий об’ємний камінь. Це візуальне підсвічування виконано у формі тривимірної сфери з м’яким блиском і напівпрозорим матеріалом, що допомагає інтуїтивно зрозуміти, куди буде поставлений камінь без потреби кліку.

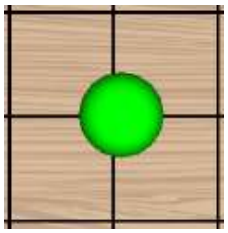

Рис. 4.13. Візуальний ефект наведення мишки

Після підтвердження ходу камінь стає постійною частиною дошки. Камені гравців рендеряться у вигляді об’ємних сфер з відповідними кольорами. Чорні камені мають глибокий матовий відтінок з легкою рефлексією (Рис. 4.14.)

Рис. 4.14. Чорний камінь

Білі камені — яскраві, з легким відблиском, що контрастує з дерев’яною текстурою дошки (Рис. 4.15.).

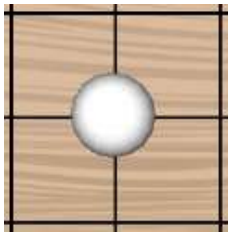

Рис. 4.15. Білий камінь

Крім тривимірної форми та матеріалу, візуальному рендеру каменів налаштовано тінь, що падає від них на поверхню дошки. Кожен камінь створює м’яку округлу тінь, яка враховує джерело освітлення сцени. Зелений камінь під курсором миші також має легку напівпрозору тінь, яка відрізняється за інтенсивністю від звичайних каменів, підкреслюючи його тимчасовий характер. Завдяки тіням дошка виглядає живою, а камені ніби фізично "лежать" на її поверхні, а не просто малюються поверх неї.

Коли камінь або група каменів майже не має дихання, це означає, що залишилася лише одна або дуже мало вільних суміжних точок (Рис. 4.16.).

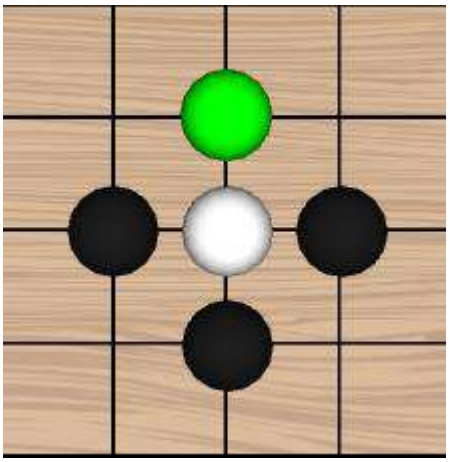

Рис. 4.16. Камінь майже позбавився дихання

В цей момент камінь залишається на полі, але знаходиться в критичному стані. Якщо у каменя зовсім не лишається дихання, він вважається захопленим і автоматично прибирається з дошки. Видалення каменя відбувається негайно після того, як перевіряється відсутність вільних суміжних точок (Рис. 4.17.).

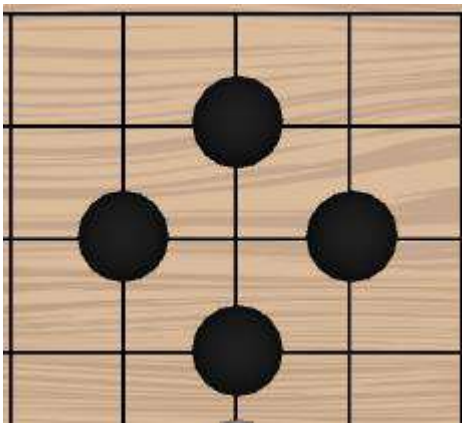

Рис. 4.17. Камінь прибрано з поля

Після того, як камінь прибирається з дошки через відсутність дихання, повторне встановлення каменя у цю клітку заборонено. Ця клітинка вважається зайнятою територією супротивника і зараховується до його рахунку.

Під час гри є можливість переглянути статистику: історію ходів та кількість захоплених територій. Під час гри є дві основні кнопки для перегляду статистики — це кнопка для відображення історії ходів (Рис. 4.18.) і кнопка для перегляду кількості захоплених територій (Рис. 4.19.). Кожна з них викликає своє вікно з інформацією у вигляді стандартного повідомлення MessageBox.

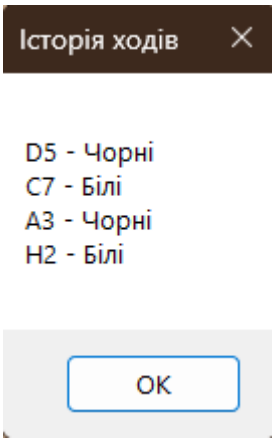

Рис. 4.18. Історія ходів

Коли користувач натискає кнопку для перегляду історії ходів, програма збирає всі зроблені ходи в одну текстову стрічку. Кожен хід представляється у вигляді рядка у списку, де зібрана інформація про кожен зроблений хід з початку гри. Якщо ходів ще немає, то замість цього виводиться повідомлення, що ходи ще не зроблені. Вікно з історією показує текст "Ще не зроблено жодного ходу" (Рис. 4.20.).

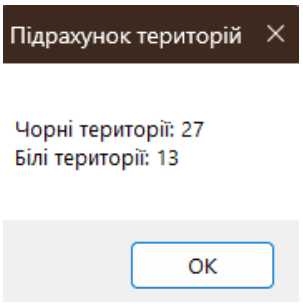

Рис. 4.19. Підрахунок територій

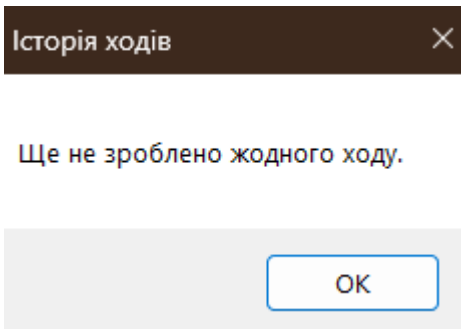

Рис. 4.20. Пуста історія

Натискання кнопки перегляду територій виконує розрахунок очок для обох гравців за допомогою логіки гри. Потім відкривається повідомлення, де вказано кількість територій, захоплених чорними та білими гравцями. Вікно має заголовок "Підрахунок територій".

Обидва вікна, фактично, є стандартними вікнами повідомлень із простим текстом, які не містять додаткових елементів управління чи інтерактивних складових. Вони відкриваються як модальні діалоги, які користувач закриває вручну.

Після того, як гравець робить пас, гра переходить до іншого гравця — змінюється черга ходу. Це означає, що якщо зараз ходять чорні, після пасу хід переходить до білих, і навпаки.

Якщо двічі підряд ніхто не зробив хід, гра автоматично завершується. Кінець гри супроводжується виведенням вікна повідомлення, яке показує рахунок для обох гравців і оголошує переможця. У цьому вікні вказується кількість очок, які набрали чорні та білі камені (Рис. 4.21.).

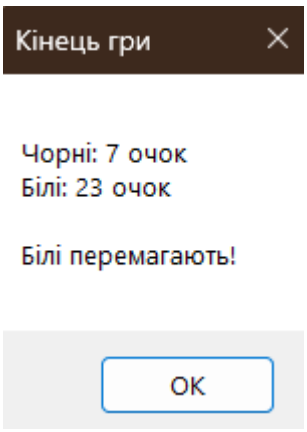

Рис. 4.21. Кінець гри

Одразу після цього відкривається ще одне діалогове вікно з питанням:

"Гра завершена! Почати нову гру?"

Тут користувач може вибрати «Так» або «Ні». Якщо обирає «Так», поточна гра скидається — грається заново, очищується стан гри, камені і дошка оновлюється (Рис. 4.22.).

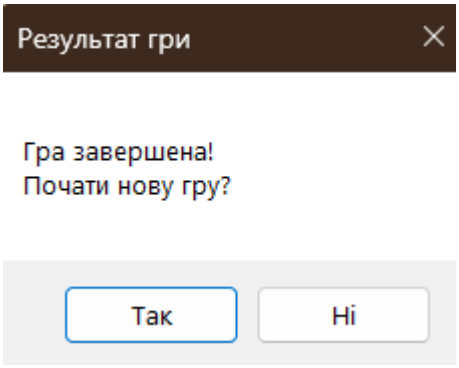

Рис. 4.22. Результати гри

Кнопка "Зберегти" зберігає поточний стан гри. Вона перевіряє, чи існує активна гра та чи ініціалізовано ігровий рушій. Далі зчитується вся дошка: якщо у клітинці є камінь, його координати, колір і номер ходу фіксуються. Ці дані формують повний список ходів, який передається у репозиторій для збереження. У такий спосіб зберігається повний поточний стан гри, щоб згодом його можна було відновити.

Кнопка "Завантажити" відкриває спеціальне вікно, у якому користувач може обрати одну з раніше збережених ігор. Після вибору гра завантажується: розміщення каменів на дошці та стан гри відновлюються так, як вони були на момент збереження. Це дозволяє продовжити гру з того місця, де вона була зупинена.

Палітра кольорів, використана в інтерфейсі гри, базується на принципах візуальної ієрархії, емоційного сприйняття кольору та ергономіки інтерфейсного дизайну. Основний фон у відтінках світло-коричневого кольору асоціюється з природними матеріалами — зокрема деревом, яке традиційно використовується для дощок гри Ґо. Такий теплий нейтральний фон зменшує візуальну втому та не відволікає увагу від ігрового процесу. Водночас він забезпечує достатній контраст для інших елементів інтерфейсу.

Для функціональних кнопок обрано насичений синій колір, що згідно з дослідженнями в області когнітивної психології асоціюється з довірою, впевненістю та дією [2]. Це полегшує сприйняття елементів керування як основних та закликає до взаємодії. Зелений колір, використаний для кнопок аналізу гри, має позитивне емоційне забарвлення й часто асоціюється з безпечними діями та розумовим процесом — що логічно відповідає функціям, пов’язаним із аналізом територій та історії ходів.

Червоний колір застосовано для кнопки видалення, що повністю відповідає загальноприйнятим стандартам дизайну — цей колір слугує попередженням і сигналізує про потенційно незворотну дію. Заголовкова панель оформлена в темно-сірому кольорі, який забезпечує високий контраст з білим текстом, зберігаючи при цьому візуальну стриманість інтерфейсу. Крім того, сірий колір часто використовується для нейтрального фону в професійному програмному забезпеченні, що додає відчуття структурованості та серйозності.

Висновки до розділу 4

В цьому розділі була побудова структури додатку. У процесі розробки було реалізовано розподіл відповідальностей між класами:

1. Класи форм відповідають за користувальницький інтерфейс
2. Класи рендерів відповідають за малювання дошки
3. Класи бд відповідають за збереження ігор
4. Класи ігрової логіки створюють алгоритм відповідний до правил гри

Огляд програми показав, що реалізовані основні можливості гри: постановка каменів, зміна ходу, передбачення заборони на самогубство та притримання правила Ко, обробка пасу, завершення гри, а також функції збереження та відновлення ігрового процесу. Передбачені механізми обробки кінцевого стану гри та повідомлення користувача через діалогові вікна MessageBox, що підвищує зручність взаємодії з додатком.

ВИСНОВКИ

У процесі виконання бакалаврської кваліфікаційної роботи було розроблено програмний застосунок для гри Go – класичної настільної гри, яка потребує глибокого стратегічного мислення та розвиненої аналітики. Реалізація даного програмного забезпечення дозволила не лише ознайомитися з особливостями розробки ігрових систем, а й вирішити комплекс технічних, алгоритмічних та організаційних завдань.

У ході роботи було досліджено теоретичні основи гри Go, вивчено її правила, а також типові підходи до цифрової реалізації ігрових механік. На підставі аналізу були визначені вимоги до майбутньої програми, сформовано її архітектуру та побудовано загальну логічну структуру взаємодії компонентів.

Було приділено увагу проектуванню внутрішнього ігрового алгоритму, що відповідає за логіку розміщення каменів, перевірку правил гри, визначення захоплень та закінчення партії. Була реалізована базова система керування ходами, яка дозволяє чергування між гравцями, здійснення пасу, завершення гри при обопільному пасі та підрахунок очок. Користувачі мають змогу взаємодіяти з графічним інтерфейсом, що спрощує сприйняття ігрового процесу.

Для забезпечення збереження ігор була спроектована та реалізована реляційна база даних. У роботі використовувалась СУБД SQLite. Розроблено таблиці для зберігання інформації про гравців, ігри, а також детальні послідовності ходів. Здійснено реалізацію запитів для створення, оновлення та видалення ігор, збереження повного стану гри, завантаження історії ходів, пошуку існуючих ігор.

Програмний застосунок також передбачає сервіси для збереження поточного стану гри в будь-який момент часу, а також можливість завантаження гри для продовження. Реалізовано обробку помилок та просту логіку повідомлень користувачу про стан гри, включно із завершенням партії.

Створений застосунок має потенціал до подальшого розширення. У перспективі можливе додавання гри з комп’ютерним суперником, підтримки мережевої гри, рейтингової системи для гравців, системи аутентифікації, а також розширення аналітичного функціоналу, додавання статистики перемог, середньої тривалості гри тощо.

Підсумовуючи, можна зазначити, що мету дипломної роботи досягнуто повністю. Усі поставлені завдання виконано: реалізовано робочий застосунок для гри Go з можливістю взаємодії з базою даних, зручною системою збереження і завантаження гри, графічним інтерфейсом та базовими функціями управління партією. Отримані результати можуть бути використані як навчальна основа для студентів-програмістів, як частина більших програмних систем або як самостійний продукт для персонального використання.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Вікіпедія — Вільна енциклопедія [Електронний ресурс]. – Режим доступу: https://uk.wikipedia.org
2. Cherry, Kendra. The Psychology of Color in Marketing and Branding [Електронний ресурс] // Verywell Mind. – 2023. – Режим доступу: https://www.verywellmind.com/color-psychology-2795824
3. WinForms documentation – .NET Desktop Guide [Електронний ресурс]. – Microsoft Learn. – Режим доступу: https://learn.microsoft.com/en-us/dotnet/desktop/winforms/
4. Рамазанова Л. І. Проектування інтерфейсів користувача з урахуванням психологічного сприйняття кольору // Сучасні проблеми дизайну. – 2020. – №2. – С. 34–41.
5. Ерліх, Г. Принципи проектування інтерфейсів користувача. – Київ: Діалектика, 2021. – 312 с.
6. Шевченко В. В. Програмування на C# для початківців. – Харків: Фоліо, 2020. – 240 с.
7. Date, C. J. An Introduction to Database Systems. – 8th ed. – Boston: Pearson, 2003. – 1024 p.
8. Вебсайт MySQL [Електронний ресурс]. – Режим доступу: https://www.mysql.com
9. CodeProject. Working with MySQL in C# [Електронний ресурс]. – Режим доступу: https://www.codeproject.com/Articles/43438/Connect-C-to-MySQL
10. МакКоннелл С. Совершенный код: мастер-класс. – М.: Русская редакция, 2020. – 912 с.
11. Freeman A., Sharp M. Pro ASP.NET Core MVC. – Apress, 2021. – 1040 p.
12. MSDN – Microsoft Developer Network [Електронний ресурс]. – Режим доступу: https://learn.microsoft.com

13. Куртова А. С. Особливості застосування кольору в UI/UX дизайні мобільних застосунків // Вісник дизайну. – 2021. – №3. – С. 12–18.
14. Бардиш К. О. Основи проектування програмних систем. – К.: КНУ, 2019. – 196 с.
15. ISO/IEC 25010:2011 – Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) – System and software quality models.
16. Krug, S. Don't Make Me Think: A Common Sense Approach to Web Usability. – 3rd ed. – New Riders, 2014. – 216 p.
17. Ткаченко І. Засади створення інтерфейсів користувача у Windows Forms // Програмні технології. – 2020. – №1. – С. 45–51.
18. C# Programming Guide – Microsoft Docs [Електронний ресурс]. – Режим доступу: https://learn.microsoft.com/en-us/dotnet/csharp/programming-guide/
19. W3Schools – HTML, CSS, JavaScript tutorials [Електронний ресурс]. – Режим доступу: https://www.w3schools.com
20. Чумаченко І. П. Бази даних та СУБД: навч. посібник. – Харків: ХНУРЕ, 2021. – 276 с.

ДОДАТКИ

Додаток А

Код форми гри

```
internal partial class GameForm : Form
{
    private GameEngine _gameEngine;
    private BoardRenderer _renderer;
    private GLControl _glControl;
    private Label _playerLabel;
    private Button _restartButton;
    private Button _passButton;
    private Button _historyButton;
    private Button _territoryButton;
    private Button _saveGameButton;
    private Button _loadGameButton;

    private MySqlGameRepository _repository;
    private int _currentGameId = -1;
    private int _moveNumber = 0;

    public GameForm()
    {
        InitializeRepository();
        InitializeForm();
        InitializeComponent();
        ShowGameSelectionDialog();
    }

    private void InitializeRepository()
    {
        _repository = new MySqlGameRepository("localhost", "go_game", "root", "root");
    }

    private void InitializeForm()
    {
        Text = "Гра в Го";
        Size = new Size(800, 850);
        StartPosition = FormStartPosition.CenterScreen;
    }

    private void ShowGameSelectionDialog()
    {
        var selectionForm = new GameSelectionForm(_repository);
        if (selectionForm.ShowDialog() == DialogResult.OK)
        {
            _currentGameId = selectionForm.SelectedGameId;
            if (_currentGameId > 0)
            {
                LoadGameFromDatabase();
            }
            else
            {
                ShowBoardSizeDialog();
            }
        }
        else
        {
            ShowBoardSizeDialog();
        }
    }

    private void LoadGameFromDatabase()
    {
        if (_currentGameId <= 0) return;

        var moves = _repository.LoadMoves(_currentGameId);
        if (moves.Count > 0)
        {
```

```
        var firstMove = moves[0];
        int maxCoord = Math.Max(firstMove.X, firstMove.Y);
        int boardSize = maxCoord < 9 ? 9 : maxCoord < 13 ? 13 : 19;

        _gameEngine = new GameEngine(boardSize);
        _gameEngine.PlayerChanged += (sender, args) => UpdatePlayerLabel(args.Player);
        _gameEngine.GameEnded += GameEngine_GameEnded;
        _gameEngine.MoveMade += GameEngine_MoveMade;
        _renderer = new BoardRenderer(_gameEngine.Board);

        foreach (var move in moves)
        {
            _gameEngine.MakeMoveFromDatabase(move.X, move.Y, move.Color);
            _moveNumber = Math.Max(_moveNumber, move.MoveNumber);
        }

        int lastColor = moves[moves.Count - 1].Color;
        UpdatePlayerLabel(_gameEngine.CurrentPlayer);
    }
    else
    {
        ShowBoardSizeDialog();
    }
}

private void ShowBoardSizeDialog()
{
    Color backgroundColor = Color.FromArgb(242, 225, 181);
    Color headerColor = Color.FromArgb(54, 54, 54);
    Color buttonPrimaryColor = Color.FromArgb(36, 93, 148);
    Color comboBoxColor = Color.FromArgb(255, 250, 240);
    Font buttonFont = new Font("Segoe UI", 11, FontStyle.Bold);
    Font comboFont = new Font("Segoe UI", 12);
    Font titleFont = new Font("Segoe UI", 14, FontStyle.Bold);

    var sizeDialog = new Form
    {
        Text = "Розмір дошки",
        Size = new Size(300, 220),
        StartPosition = FormStartPosition.CenterParent,
        FormBorderStyle = FormBorderStyle.FixedDialog,
        MaximizeBox = false,
        MinimizeBox = false,
        BackColor = backgroundColor
    };

    Panel headerPanel = new Panel
    {
        Dock = DockStyle.Top,
        Height = 50,
        BackColor = headerColor
    };

    Label titleLabel = new Label
    {
        Text = "Виберіть розмір дошки",
        Font = titleFont,
        ForeColor = Color.White,
        Dock = DockStyle.Fill,
        TextAlign = ContentAlignment.MiddleCenter
    };
    headerPanel.Controls.Add(titleLabel);
    sizeDialog.Controls.Add(headerPanel);

    Panel contentPanel = new Panel
    {
        Dock = DockStyle.Fill,
        Padding = new Padding(25, 75, 25, 70)
```

```
    };
    sizeDialog.Controls.Add(contentPanel);

    var comboBox = new ComboBox
    {
        Dock = DockStyle.Fill,
        DropDownStyle = ComboBoxStyle.DropDownList,
        Font = comboFont,
        BackColor = comboBoxColor,
        FlatStyle = FlatStyle.Flat,
    };
    comboBox.Items.AddRange(new object[] { "9x9 (Маленька)", "13x13 (Середня)", "19x19 (Стандартна)" });
    comboBox.SelectedIndex = 0;
    contentPanel.Controls.Add(comboBox);

    var okButton = new Button
    {
        Text = "Почати гру",
        Dock = DockStyle.Bottom,
        Height = 50,
        Font = buttonFont,
        ForeColor = Color.White,
        BackColor = buttonPrimaryColor,
        FlatStyle = FlatStyle.Flat,
        UseVisualStyleBackColor = false,
        Cursor = Cursors.Hand
    };
    okButton.FlatAppearance.BorderSize = 0;
    okButton.FlatAppearance.MouseOverBackColor = ControlPaint.Light(buttonPrimaryColor);
    okButton.FlatAppearance.MouseDownBackColor = ControlPaint.Dark(buttonPrimaryColor);

    okButton.Click += (s, e) =>
    {
        int boardSize = comboBox.SelectedIndex == 0 ? 9 : comboBox.SelectedIndex == 1 ? 13 : 19;
        _gameEngine = new GameEngine(boardSize);
        _gameEngine.PlayerChanged += (sender, args) => UpdatePlayerLabel(args.Player);
        _gameEngine.GameEnded += GameEngine_GameEnded;
        _gameEngine.MoveMade += GameEngine_MoveMade;
        _renderer = new BoardRenderer(_gameEngine.Board);
        sizeDialog.Close();
    };

    sizeDialog.Controls.Add(okButton);
    sizeDialog.ShowDialog();
}

private void UpdatePlayerLabel(int player)
{
    if (player == null) _playerLabel.Text = "Хід: Чорні";
    else
        _playerLabel.Text = player == 1 ? "Хід: Чорні" : "Хід: Білі";
}

private void RestartButton_Click(object sender, EventArgs e)
{
    _gameEngine.Reset();
    _moveNumber = 0;
    _currentGameId = -1;
    _glControl.Invalidate();
}

private void PassButton_Click(object sender, EventArgs e)
{
    _gameEngine.Pass();
}

private void SaveGameButton_Click(object sender, EventArgs e)
{
```

```
    if (_currentGameId <= 0)
    {
        var newGameForm = new NewGameForm(_repository);
        if (newGameForm.ShowDialog() == DialogResult.OK)
        {
            _currentGameId = newGameForm.CreatedGameId;
            SaveCurrentGameState();
            MessageBox.Show("Гра збережена успішно!", "Успіх");
        }
    }
    else
    {
        SaveCurrentGameState();
        MessageBox.Show("Гра оновлена успішно!", "Успіх");
    }
}

private void LoadGameButton_Click(object sender, EventArgs e)
{
    var selectionForm = new GameSelectionForm(_repository);
    if (selectionForm.ShowDialog() == DialogResult.OK)
    {
        _currentGameId = selectionForm.SelectedGameId;
        if (_currentGameId > 0)
        {
            LoadGameFromDatabase();
            _glControl.Invalidate();
        }
    }
}

private void SaveCurrentGameState()
{
    if (_currentGameId <= 0 || _gameEngine == null) return;

    var moves = new List<Move>();
    int moveNum = 1;

    for (int x = 0; x < _gameEngine.Board.Size; x++)
    {
        for (int y = 0; y < _gameEngine.Board.Size; y++)
        {
            int stone = _gameEngine.Board.Grid[x, y];
            if (stone != 0)
            {
                moves.Add(new Move(moveNum++, x, y, stone));
            }
        }
    }

    _repository.SaveFullGameState(_currentGameId, moves);
}

private void GlControl_Load(object sender, EventArgs e)
{
    _glControl.MakeCurrent();
    _renderer.InitializeGL();
    _renderer.SetupViewport(_glControl.Width, _glControl.Height);
}

private void GlControl_Paint(object sender, PaintEventArgs e)
{
    if (!_glControl.Context.IsCurrent)
        _glControl.MakeCurrent();

    _renderer.Render();
    _glControl.SwapBuffers();
}
```

```
private void GlControl_Resize(object sender, EventArgs e)
{
   if (_glControl.ClientSize.Width > 0 && _glControl.ClientSize.Height > 0)
   {
      _renderer.SetupViewport(_glControl.Width, _glControl.Height);
      _glControl.Invalidate();
   }
}

private void GlControl_MouseClick(object sender, MouseEventArgs e)
{
   var boardCoordinates = GetBoardCoordinates(e.X, e.Y);
   if (boardCoordinates == null) return;

   int x = boardCoordinates.Value.Item1;
   int y = boardCoordinates.Value.Item2;

   if (_gameEngine.MakeMove(x, y))
   {
      _glControl.Invalidate();
   }
}

private void GameEngine_MoveMade(object sender, MoveEventArgs e)
{
   _moveNumber++;

   if (_currentGameId > 0)
   {
      _repository.SaveMove(_currentGameId, _moveNumber, e.X, e.Y, e.Player);
   }
}

private void GlControl_MouseMove(object sender, MouseEventArgs e)
{
   var point = GetClickBoardIntersection(e.X, e.Y);
   if (point != null)
   {
      float boardX = (point.Value.X + 1.0f) / 2 * (_gameEngine.Board.Size - 1);
      float boardY = (point.Value.Y + 1.0f) / 2 * (_gameEngine.Board.Size - 1);
      _renderer.SetHoverPosition(new Vector2(boardX, boardY));
   }
   else
   {
      _renderer.SetHoverPosition(null);
   }

   _glControl.Invalidate();
}

private (int, int)? GetBoardCoordinates(int mouseX, int mouseY)
{
   var point = GetClickBoardIntersection(mouseX, mouseY);
   if (point == null) return null;

   float worldX = point.Value.X;
   float worldY = point.Value.Y;

   float boardX = (worldX + 1.0f) / 2 * (_gameEngine.Board.Size - 1);
   float boardY = (worldY + 1.0f) / 2 * (_gameEngine.Board.Size - 1);

   int x = (int)Math.Round(boardX);
   int y = (int)Math.Round(boardY);

   if (x >= 0 && x < _gameEngine.Board.Size && y >= 0 && y < _gameEngine.Board.Size)
   {
      return (x, y);
```

Продовження додатку А

```
        }

        return null;
    }

    private Vector3? GetClickBoardIntersection(int mouseX, int mouseY)
    {
        int[] viewport = new int[4];
        GL.GetInteger(GetPName.Viewport, viewport);

        Matrix4 modelview, projection;
        GL.GetFloat(GetPName.ModelviewMatrix, out modelview);
        GL.GetFloat(GetPName.ProjectionMatrix, out projection);

        Vector3 near = BoardRenderer.UnProject(new Vector3(mouseX, mouseY, 0.0f), modelview, projection, viewport) ?? Vector3.Zero;
        Vector3 far = BoardRenderer.UnProject(new Vector3(mouseX, mouseY, 1.0f), modelview, projection, viewport) ?? Vector3.Zero;

        Vector3 dir = (far - near).Normalized();
        if (dir.Z == 0) return null;

        float t = -near.Z / dir.Z;
        if (t < 0) return null;

        return near + dir * t;
    }

    private void HistoryButton_Click(object sender, EventArgs e)
    {
        string moves = string.Join("\n", _gameEngine.MoveHistory);
        MessageBox.Show(moves.Length > 0 ? moves : "Ще не зроблено жодного ходу.", "Історія ходів");
    }

    private void TerritoryButton_Click(object sender, EventArgs e)
    {
        var scores = _gameEngine.CalculateScore();
        MessageBox.Show($"Чорні території: {scores.blackScore}\nБілі території: {scores.whiteScore}", "Підрахунок територій");
    }

    private void GameEngine_GameEnded(object sender, GameScoreEventArgs e)
    {
        string winner = e.BlackScore > e.WhiteScore
            ? "Чорні перемагають!"
            : e.WhiteScore > e.BlackScore
                ? "Білі перемагають!"
                : "Нічия!";

        MessageBox.Show($"Чорні: {e.BlackScore} очок\nБілі: {e.WhiteScore} очок\n\n{winner}", "Кінець гри");

        DialogResult result = MessageBox.Show($"Гра завершена!\nПочати нову гру?", "Результат гри", MessageBoxButtons.YesNo);
        if (result == DialogResult.Yes)
        {
            _gameEngine.Reset();
            _moveNumber = 0;
            _currentGameId = -1;
            _glControl.Invalidate();
        }
    }
}
```

Додаток Б

Код форми вибору гри

```
internal partial class GameSelectionForm : Form
{
    private MySqlGameRepository repository;
    public int SelectedGameId { get; private set; } = -1;

    public GameSelectionForm(MySqlGameRepository repo)
    {
        InitializeComponent();
        repository = repo;
        LoadGames();
    }

    private void LoadGames()
    {
        listBoxGames.Items.Clear();
        var games = repository.GetAllGamesWithPlayers();

        foreach (var g in games)
        {
            listBoxGames.Items.Add($"{g.id}: {g.black} vs {g.white} ({g.createdAt})");
        }
    }

    private void btnLoad_Click(object sender, EventArgs e)
    {
        if (listBoxGames.SelectedIndex >= 0)
        {
            var line = listBoxGames.SelectedItem.ToString();
            SelectedGameId = int.Parse(line.Split(':')[0]);
            DialogResult = DialogResult.OK;
            Close();
        }
    }

    private void btnNewGame_Click(object sender, EventArgs e)
    {
        var form = new NewGameForm(repository);
        if (form.ShowDialog() == DialogResult.OK)
        {
            LoadGames();
        }
    }

    private void btnDelete_Click(object sender, EventArgs e)
    {
        if (listBoxGames.SelectedIndex >= 0)
        {
            var line = listBoxGames.SelectedItem.ToString();
            int gameId = int.Parse(line.Split(':')[0]);

            var confirm = MessageBox.Show("Видалити цю гру?", "Підтвердження", MessageBoxButtons.YesNo);
            if (confirm == DialogResult.Yes)
            {
                repository.DeleteGame(gameId);
                LoadGames();
            }
        }
    }
}
```

Додаток В

Код створення дошки

```
internal class BoardRenderer
{
    private readonly Board _board;
    private Vector2? _hoverPosition;
    private int _woodTextureId;
    private readonly string _letters = "ABCDEFGHJKLMNOPQRST";

    public BoardRenderer(Board board)
    {
        _board = board;
    }

    public void SetHoverPosition(Vector2? position)
    {
        _hoverPosition = position;
    }

    public void InitializeGL()
    {
        GL.ClearColor(0.6f, 0.3f, 0.1f, 1.0f);
        GL.Enable(EnableCap.DepthTest);
        GL.Enable(EnableCap.Lighting);
        GL.Enable(EnableCap.Light0);
        GL.Enable(EnableCap.ColorMaterial);
        GL.ColorMaterial(MaterialFace.Front, ColorMaterialParameter.AmbientAndDiffuse);

        float[] lightPosition = { 0.0f, 0.0f, 5.0f, 1.0f };
        GL.Light(LightName.Light0, LightParameter.Position, lightPosition);

        LoadTexture(new Bitmap("C:\\Users\\Anastasiya\\source\\repos\\GoGame\\GoGame\\wood_texture.png"));
    }

    public void SetupViewport(int width, int height)
    {
        GL.Viewport(0, 0, width, height);
        GL.MatrixMode(MatrixMode.Projection);
        GL.LoadIdentity();

        Matrix4 perspective = Matrix4.CreatePerspectiveFieldOfView(
            OpenTK.MathHelper.PiOver4, width / (float)height, 1.0f, 100.0f);
        GL.LoadMatrix(ref perspective);

        GL.MatrixMode(MatrixMode.Modelview);
        GL.LoadIdentity();
    }

    public void Render()
    {
        GL.Clear(ClearBufferMask.ColorBufferBit | ClearBufferMask.DepthBufferBit);
        GL.LoadIdentity();
        GL.Translate(0, 0, -4);

        DrawBoard();
        DrawStones();
    }

    private void DrawBoard()
    {
        float boardSize = 2.0f;
        float thickness = 0.1f;
        float cellSize = 2.0f / (_board.Size - 1);

        DrawRoundedBoard(boardSize + 0.6f, 0.2f);

        GL.Color3(0f, 0f, 0f);
```

```
    GL.LineWidth(2f);
    GL.Begin(PrimitiveType.Lines);
    for (int i = 0; i < _board.Size; i++)
    {
        float pos = -1.0f + i * cellSize;
        GL.Vertex3(pos, -1.0f, 0.001f);
        GL.Vertex3(pos, 1.0f, 0.001f);
        GL.Vertex3(-1.0f, pos, 0.001f);
        GL.Vertex3(1.0f, pos, 0.001f);
    }
    GL.End();

    if (_hoverPosition.HasValue)
    {
        GL.PushMatrix();
        GL.Translate(
            _hoverPosition.Value.X / (_board.Size - 1) * 2 - 1,
            _hoverPosition.Value.Y / (_board.Size - 1) * 2 - 1,
            0.1f);
        GL.Color3(0.0f, 1.0f, 0.0f);
        DrawSphere(cellSize * 0.4f, 16, 16);
        GL.PopMatrix();
    }

}

private void DrawRoundedBoard(float size, float cornerRadius)
{
    GL.Enable(EnableCap.Texture2D);
    GL.BindTexture(TextureTarget.Texture2D, _woodTextureId);
    GL.Disable(EnableCap.Lighting);
    GL.Color3(1f, 1f, 1f);

    float halfSize = size / 2;
    float halfWidth = halfSize - cornerRadius;
    float halfHeight = halfSize - cornerRadius;

    int segments = 12;

    GL.Begin(PrimitiveType.TriangleFan);
    GL.Normal3(0, 0, 1);
    GL.TexCoord2(0.5f, 0.5f);
    GL.Vertex3(0, 0, 0);

    for (int i = 0; i <= segments / 4; i++)
    {
        float angle = (float)i / segments * 2.0f * (float)Math.PI;
        float x = halfWidth + cornerRadius * (float)Math.Cos(angle);
        float y = halfHeight + cornerRadius * (float)Math.Sin(angle);
        GL.TexCoord2((x + halfSize) / size, (y + halfSize) / size);
        GL.Vertex3(x, y, 0);
    }

    for (int i = segments / 4; i <= segments / 2; i++)
    {
        float angle = (float)i / segments * 2.0f * (float)Math.PI;
        float x = -halfWidth + cornerRadius * (float)Math.Cos(angle);
        float y = halfHeight + cornerRadius * (float)Math.Sin(angle);
        GL.TexCoord2((x + halfSize) / size, (y + halfSize) / size);
        GL.Vertex3(x, y, 0);
    }

    for (int i = segments / 2; i <= 3 * segments / 4; i++)
    {
        float angle = (float)i / segments * 2.0f * (float)Math.PI;
        float x = -halfWidth + cornerRadius * (float)Math.Cos(angle);
        float y = -halfHeight + cornerRadius * (float)Math.Sin(angle);
        GL.TexCoord2((x + halfSize) / size, (y + halfSize) / size);
```

```
            GL.Vertex3(x, y, 0);
        }

        for (int i = 3 * segments / 4; i <= segments; i++)
        {
            float angle = (float)i / segments * 2.0f * (float)Math.PI;
            float x = halfWidth + cornerRadius * (float)Math.Cos(angle);
            float y = -halfHeight + cornerRadius * (float)Math.Sin(angle);
            GL.TexCoord2((x + halfSize) / size, (y + halfSize) / size);
            GL.Vertex3(x, y, 0);
        }

        float closingAngle = 0;
        float cx = halfWidth + cornerRadius * (float)Math.Cos(closingAngle);
        float cy = halfHeight + cornerRadius * (float)Math.Sin(closingAngle);
        GL.TexCoord2((cx + halfSize) / size, (cy + halfSize) / size);
        GL.Vertex3(cx, cy, 0);

        GL.End();
        GL.Disable(EnableCap.Texture2D);
        GL.Enable(EnableCap.Lighting);

        DrawCoordinates();
    }

    private void DrawStones()
    {
        float cellSize = 2.0f / (_board.Size - 1);

        for (int x = 0; x < _board.Size; x++)
        {
            for (int y = 0; y < _board.Size; y++)
            {
                if (_board.Grid[x, y] != 0)
                {
                    float cx = -1.0f + x * cellSize;
                    float cy = -1.0f + y * cellSize;
                    bool isBlack = _board.Grid[x, y] == 1;
                    Draw3DStone(cx, cy, cellSize * 0.4f, isBlack);
                }
            }
        }
    }

    private void Draw3DStone(float cx, float cy, float radius, bool isBlack)
    {
        GL.PushMatrix();
        GL.Translate(cx, cy, 0.05f);
        if (isBlack) GL.Color3(0.1f, 0.1f, 0.1f);
        else GL.Color3(1.0f, 1.0f, 1.0f);

        DrawSphere(radius, 16, 16);
        GL.PopMatrix();
    }

    private void DrawSphere(float radius, int slices, int stacks)
    {
        for (int i = 0; i <= stacks; i++)
        {
            float lat0 = (float)Math.PI * (-0.5f + (float)(i - 1) / stacks);
            float z0 = (float)Math.Sin(lat0);
            float zr0 = (float)Math.Cos(lat0);

            float lat1 = (float)Math.PI * (-0.5f + (float)i / stacks);
            float z1 = (float)Math.Sin(lat1);
            float zr1 = (float)Math.Cos(lat1);

            GL.Begin(PrimitiveType.QuadStrip);
```

```
        for (int j = 0; j <= slices; j++)
        {
            float lng = 2 * (float)Math.PI * (float)(j - 1) / slices;
            float x = (float)Math.Cos(lng);
            float y = (float)Math.Sin(lng);

            GL.Normal3(x * zr0, y * zr0, z0);
            GL.Vertex3(x * zr0 * radius, y * zr0 * radius, z0 * radius);
            GL.Normal3(x * zr1, y * zr1, z1);
            GL.Vertex3(x * zr1 * radius, y * zr1 * radius, z1 * radius);
        }
        GL.End();
    }
}

public static Vector3? UnProject(Vector3 mouse, Matrix4 modelview, Matrix4 projection, int[] viewport)
{
    Matrix4 viewProjectionInverse = Matrix4.Invert(modelview * projection);

    Vector4 ndc = new Vector4(
        (mouse.X - viewport[0]) / viewport[2] * 2.0f - 1.0f,
        1.0f - (mouse.Y - viewport[1]) / viewport[3] * 2.0f,
        2.0f * mouse.Z - 1.0f,
        1.0f
    );

    Vector4 world = Vector4.Transform(ndc, viewProjectionInverse);
    if (world.W == 0.0f) return null;

    return new Vector3(world.X / world.W, world.Y / world.W, world.Z / world.W);
}

public void LoadTexture(Bitmap bitmap)
{
    GL.GenTextures(1, out _woodTextureId);
    GL.BindTexture(TextureTarget.Texture2D, _woodTextureId);

    BitmapData data = bitmap.LockBits(
        new Rectangle(0, 0, bitmap.Width, bitmap.Height),
        ImageLockMode.ReadOnly,
        System.Drawing.Imaging.PixelFormat.Format32bppArgb);

    GL.TexImage2D(TextureTarget.Texture2D, 0, PixelInternalFormat.Rgba,
        data.Width, data.Height, 0, OpenTK.Graphics.OpenGL.PixelFormat.Bgra,
        PixelType.UnsignedByte, data.Scan0);

    bitmap.UnlockBits(data);

    GL.TexParameter(TextureTarget.Texture2D, TextureParameterName.TextureMinFilter, (int)TextureMinFilter.Linear);
    GL.TexParameter(TextureTarget.Texture2D, TextureParameterName.TextureMagFilter, (int)TextureMagFilter.Linear);
}

private void DrawCoordinates()
{
    float cellSize = 2.0f / (_board.Size - 1);
    float scale = 0.2f;
    float offset = 0.01f;

    GL.Disable(EnableCap.Lighting);

    for (int i = 0; i < _board.Size; i++)
    {
        float x = -1.0f + i * cellSize;
        string letter = _letters[i].ToString();

        TextRenderer.DrawTextOnBoard(letter, new Vector2(x - scale / 2 - 0.1f, -1.1f - scale - offset), scale);

        TextRenderer.DrawTextOnBoard(letter, new Vector2(x - scale / 2 - 0.1f, 1.1f + offset), scale);
```

Продовження додатку В

```
        }

        for (int i = 0; i < _board.Size; i++)
        {
            float y = -1.0f + i * cellSize;
            string number = (_board.Size - i).ToString();

            TextRenderer.DrawTextOnBoard(number, new Vector2(-1.2f - scale - offset, y - scale / 2), scale);

            TextRenderer.DrawTextOnBoard(number, new Vector2(1.0f + offset, y - scale / 2), scale);
        }

        GL.Enable(EnableCap.Lighting);
    }
}
```