

Форма № 24

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук і технологій

(повне найменування факультету)

Кафедра програмних засобів

(повне найменування кафедри)

Пояснювальна записка

до дипломного проекту (роботи)

бакалавр

(ступінь вищої освіти)

на тему РОЗРОБКА МОБІЛЬНОГО ДОДАТКА "ШАХИ"

НА ПЛАТФОРМІ UNITY

DEVELOPMENT OF THE CHESS MOBILE APPLICATION

ON THE UNITY PLATFORM

Виконав: студент 4 курсу, групи КНТ-220сп

Спеціальності 122Комп'ютерні науки

(код і найменування спеціальності)

Освітня програма (спеціалізація)

Комп'ютерні науки

ШЕВЦОВ В.Д.

(ПРИЗВИЩЕ та ініціали)

Керівник ШИТІКОВА О.В.

(ПРИЗВИЩЕ та ініціали)

Рецензент ТЕРЕЩЕНКО Е.В.

(ПРИЗВИЩЕ та ініціали)

2023

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет ФКНТ
Кафедра програмних засобів
Ступінь вищої освіти бакалавр
Спеціальність 122 Комп'ютерні науки
(код і найменування)
Освітня програма (спеціалізація) Комп'ютерні науки
(назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ, д.т.н, проф.
Сергій СУББОТІН
“ ” 2023 року

З А В Д А Н Н Я
НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА

ШЕВЦОВА Віктора Дмитровича

(ПРИЗВИЩЕ, ім'я, по батькові)

1. Тема проєкту (роботи) Розробка мобільного додатка "Шахи" на платформі Unity
Development of the chess mobile application on the Unity platform

керівник проєкту (роботи) к.т.н., ШИТИКОВА Олена Вікторівна

(науковий ступінь, вчене звання, ПРИЗВИЩЕ, ім'я, по батькові)

затверджені наказом закладу вищої освіти від « 04 » квітня 2023 року № 87

2. Строк подання студентом проєкту (роботи) 01 червня 2023 року

3. Вихідні дані до проєкту (роботи) рекомендована література, технічне завдання

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1) Аналіз предметної області та постановка завдань дослідження. 2) Матеріали і методи. 3) Реалізація проєкту. 4) Експлуатація, тестування та експериментальне дослідження програми.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, кількість слайдів, плакатів)

Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	ПРИЗВИЩЕ, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-4 Основна частина	ШИТІКОВА О.В., доцент		
Нормоконтроль	БЄЛОВА А.В., асистент		

7. Дата видачі завдання «04» квітня 2023 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Постановка завдання роботи	1 тиждень	Завдання, ТЗ
2	Аналіз предметної області	2 тиждень	Розділ 1
3	Вибір технологій розробки	2 тиждень	Розділ 2
4	Розробка архітектури програми	3 тиждень	Розділ 3
5	Розробка програми	4-5 тиждень	Розділ 3
6	Тестування та експериментальне дослідження програми	6 тиждень	Розділ 4
7	Оформлення пояснювальної записки та документів до неї. Нормоконтроль та рецензування	7 тиждень	Додатки
8	Захист роботи	8 тиждень	

Студент _____ Віктор ШЕВЦОВ
(підпис) (Ім'я ПРИЗВИЩЕ)

Керівник проєкту (роботи) _____ Олена ШИТІКОВА
(підпис) (Ім'я ПРИЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи бакалавра:
107 с., 41 рис., 4 табл., 2 дод., 16 джерел.

ШАХИ, КОМПОНЕНТ, СТРУКТУРА, IDE, IOS, VS.

Об'єкт дослідження – ігри, створені на платформі Unity.

Предмет роботи – програмне забезпечення мобільного додатку «Шахи».

Мета роботи – Розроблення мобільного додатку для гри у шахи.

Матеріали, методи та технічні засоби:

- Unity3D. Unity3D - це популярне інтегроване розвітокове середовище (IDE) та движок для розробки ігор та інтерактивних додатків. Використання Unity3D дозволяє створювати 2D та 3D графіку, реалізовувати фізику, керувати взаємодією об'єктів та багато іншого;
- UnityHub. UnityHub - це інструмент для управління версіями та установки різних версій Unity. Він дозволяє легко перемикатись між різними версіями движка Unity та керувати проектами;
- Visual Studio. Visual Studio - це інтегроване середовище розробки (IDE) від Microsoft, яке широко використовується для програмування різноманітних додатків, включаючи ігри. Використання Visual Studio з Unity дозволяє зручно розробляти, налагоджувати та керувати кодом проекту;
- метод пошуку всіх можливих ходів. Цей метод шукає всі можливі ходи на дошці для кожної фігури. Він використовується для генерації списку доступних ходів, які гравець може зробити в грі;
- алгоритм перевірки можливості ходу. Цей алгоритм перевіряє, чи є певний хід допустимим для заданої фігури. Він враховує правила руху кожної фігури і перевіряє, чи виконуються необхідні умови для здійснення ходу;
- метод зміни кольору: Цей метод змінює колір фігури на протилежний. Він використовується для зміни кольору фігури при виконанні ходу противником.

Результати. Проведено дослідження та аналіз вимог до програмного забезпечення. Визначено структуру даних та необхідні компоненти програми. Розроблено структуру і алгоритм роботи програми на основі встановлених вимог. Реалізовано відповідні компоненти програми. Розроблено можливість грати проти іншої людини. Розроблено програмне забезпечення з використанням Unity3D та Visual Studio, де детально вивчалися, показувалися та обґрунтовувалися вимоги до програми. На основі цих вимог була визначена структура даних та розроблений алгоритм роботи програми. В результаті були реалізовані необхідні компоненти програми, що дозволяють грати проти іншої людини.

Висновки. Розроблено програмне забезпечення, що дозволяє грати з іншою людиною, провести час та розвинути навички логічного мислення.

Галузь використання – програмне забезпечення може використовуватися як додаток для розваги, а також для розвитку логіки, концентрації тощо.

ABSTRACT

Explanatory note to the bachelor's qualification work: 107 p., 41 fig., 4 tab., 2 app., 16 sources.

CHESS, COMPONENT, STRUCTURE, IDE, IOS, VS

Research Object. Games developed on the Unity platform.

Subject of the Work. Software for a mobile application "Chess".

Objective. Development of a mobile application for playing Chess.

Materials, Methods, and Technical Tools:

- Unity3D. Unity3D is a popular integrated development environment (IDE) and engine for game and interactive application development. Using Unity3D allows for creating 2D and 3D graphics, implementing physics, managing object interactions, and more;
- UnityHub. UnityHub is a tool for managing versions and installing different versions of Unity. It enables easy switching between different Unity engine versions and project management;
- Visual Studio. Visual Studio is an integrated development environment (IDE) by Microsoft widely used for programming various applications, including games. Using Visual Studio with Unity provides convenient development, debugging, and code management capabilities;
- All Possible Moves Method. This method searches for all possible moves on the chessboard for each chess piece. It is used to generate a list of available moves that a player can make in the game;
- Move Validity Checking Algorithm. This algorithm checks if a specific move is valid for a given chess piece. It considers the movement rules of each piece and verifies if the necessary conditions for making the move are met;
- Color Change Method. This method changes the color of a chess piece to the opposite color. It is used to change the color of a piece when it is moved by the opponent.

Results. Research and analysis of software requirements have been conducted. The data structure and necessary program components have been determined. The structure and algorithm of the program have been developed based on the established requirements. The corresponding program components have been implemented. The ability to play against another human player has been developed. Software development has been carried out using Unity3D and Visual Studio, where requirements were thoroughly studied, presented, and justified. Based on these requirements, the data structure was defined, and the program's algorithm was developed. As a result, the necessary program components were implemented, enabling gameplay against another human player.

Conclusions. Software has been developed that allows playing against another human player, providing entertainment and enhancing logical thinking skills.

Application Field. The software can be used as an entertainment application as well as for the development of logic, concentration, and other related skills.

ЗМІСТ

	С.
Перелік скорочень та умовних позначок	10
Вступ.....	11
1. Аналіз предметної області.....	12
1.1 Особливості розробки мобільних ігор на платформі Unity	12
1.2 Огляд моделей та алгоритмів гри в шахи	13
1.3 Огляд програмних аналогів	14
1.4 Постановка завдання.....	20
1.5 Висновки за розділом 1	21
2. Технічні та програмні засоби розробки	22
2.1 Вибір мови програмування	23
2.2 Вибір інтегрованого середовища розробки.....	25
2.3 Складові платформи Unity	26
2.3.1 Об'єкти в Unity.....	27
2.3.2 Панель Інспектора.....	27
2.3.3 Ієрархія в Unity	29
2.3.4 Асети в Unity	30
2.3.5 Спрайти в Unity	31
2.3.6 Кастомізація робочої області	32
2.4 Компоненти в Unity.....	32
2.5 Висновки за розділом 2	37
3. Реалізація проєкту	39
3.1 Ієрархія та об'єкти проєкту	39
3.2 Розроблені класи	42
3.2.1 Клас "Game"	42
3.2.2 Клас "ChessMan"	45
3.2.3 Клас "MovePlate"	47

3.2.4 Клас "Menu"	48
3.2.5 Клас "PossibleMoves"	50
3.3 Висновки до розділу 3	51
4. Експлуатація та тестування програми	52
4.1 Опис програми.....	52
4.2 Експлуатація програми.....	52
4.3 Тестування і опис принципів експлуатації програми	58
4.4 Висновки за розділом 4	58
Висновки	59
Перелік джерел посилання	60
Додаток А Текст програми.....	62
A.1 Клас Menu	63
A.2 Клас Game	66
A.3 Клас ChessMan.....	71
A.4 Клас PossibleMoves	91
A.5 Клас MovePlate	99
Додаток Б Слайди презентації	101

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

IDE – Integrated Development Environment;

iOS – iPhoneOS. Операційна система пристроїв від Apple;

VS– VisualStudio.

ВСТУП

Шахи - це старовинна настільна гра, яка виникла більше 1500 років тому. Ця гра вимагає від гравців високого рівня стратегічного мислення та тактичних здібностей. Шахи також відомі своєю здатністю розвивати мислення та пам'ять, тому їх часто використовують в якості тренувального інструменту для мозку. Незважаючи на те, що шахи виникли дуже давно, до сих пір вони залишаються популярною грою по всьому світу завдяки своїм цікавим правилам та нескінченній кількості можливих комбінацій[2].

Чому шахи до цього часу є популярними? По-перше, це гра, яка надає гравцям безліч можливостей для покращення своїх навичок та стратегій. По-друге, шахи вважаються універсальною грою, яку можуть грати люди будь-якого віку та культури. І, нарешті, шахи надають можливість для соціальної взаємодії та створення спільноти, де люди можуть ділитися своїми знаннями та досвідом[3].

Отже, актуальною задачею є створення зручного додатку, який можна буде використовувати на будь-якій платформі для гри в шахи.

Об'єкт дослідження – ігри, створені на платформі Unity.

Предмет дослідження – програмне забезпечення мобільного додатку «Шахи».

Мета роботи – Розроблення мобільного додатку для гри у шахи.

Для досягнення цієї мети необхідно виконати наступні завдання:

- розробити логіку гри в шахи, включаючи правила руху фігур, перевірку на шах та мат, обробку ходів гравців і перемогу;
- розробити зручний інтерфейс для взаємодії з додатком, включаючи візуалізацію дошки, фігур, кнопок та інших елементів управління;
- забезпечити можливість гри та вирішення задач на різних платформах, таких як комп'ютери, мобільні пристрої тощо;
- протестувати та оптимізувати додаток для забезпечення його ефективності та стабільності.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Особливості розробки мобільних ігор на платформі Unity

Unity є одним з найпопулярніших інструментів для розробки мобільних ігор, і це зумовлено кількома факторами. Ось деякі особливості розробки мобільних ігор на платформі Unity:

- кроссплатформовість. Unity дозволяє розробляти ігри, які можуть працювати на різних мобільних платформах, таких як iOS та Android. Це забезпечує широке охоплення аудиторії та збільшує потенційну кількість користувачів;

- простота використання. Unity має інтуїтивний інтерфейс та надає розробникам широкий набір інструментів для створення ігрових сценаріїв, управління графікою та фізикою, анімації персонажів, обробки вхідних даних та багато іншого. Це полегшує процес розробки та дозволяє швидко створювати прототипи та тестувати ідеї;

- гнучкість. Unity підтримує мову програмування C#, яка є потужною та гнучкою. Розробники можуть використовувати C# для створення складних логічних сценаріїв, управління геймплеєм та взаємодії з користувачем. Мова C# також пропонує широкий набір бібліотек та інструментів, які полегшують розробку ігор;

- підтримка візуальних ефектів. Unity має потужну графічну підсистему, що дозволяє створювати красиві та реалістичні візуальні ефекти. Розробники можуть створювати вражаючі графічні сцени, використовуючи шейдери, освітлення, частинки та інші технології, що допомагають покращити візуальний досвід користувача;

- співтовариство розробників. Unity має велику та активну спільноту розробників, яка надає підтримку, документацію та онлайн-курси. Розробники можуть швидко знайти відповіді на свої питання, поділитися своїм досвідом та взаємодіяти з іншими фахівцями;

– можливості розширення. Unity дозволяє розробникам використовувати сторонні розширення, плагіни та ресурси для розширення функціональності. Це дає розробникам доступ до різноманітних інструментів та ресурсів, що полегшують розробку та покращують якість готового продукту.

Розробка мобільних ігор на платформі Unity має багато переваг та може бути ефективним вибором для розробників, які бажають створити ігри для широкої аудиторії на різних мобільних платформах[6].

1.2 Огляд моделей та алгоритмів гри в шахи

Гра в шахи є складною задачею з точки зору комп'ютерного моделювання. Основні виклики полягають у визначенні оптимальних ходів, оцінці стану гри та розрахунку варіантів розвитку партії. Тут представлено кілька моделей та алгоритмів, які використовуються в грі в шахи:

– мінімакс. Мінімакс є одним з основних алгоритмів для прийняття рішень у грі в шахи. Він базується на принципі "мінімаксного критерію" та розглядає можливі ходи гравців до певної глибини. Після оцінки стану гри кожен хід оцінюється за його вигащність або програшність, враховуючи відповідність противникові. Цей алгоритм дозволяє знайти оптимальний хід, що максимізує вигащ або мінімізує програш;

– альфа-бета відсікання. Цей алгоритм є покращенням мінімакса та використовується для зменшення кількості гілок, які потрібно розглянути. Він використовує два параметри - альфа та бета - як межі для оцінки ходів. Альфа представляє максимальну оцінку, яку гравець, який розглядає максимізуючий хід, може гарантувати, тоді як бета представляє мінімальну оцінку, яку гравець, який розглядає мінімізуючий хід, може гарантувати. Це дозволяє виключити непотрібні гілки та прискорює процес пошуку оптимального рішення;

– штучна нейронна мережа. Штучні нейронні мережі використовуються для навчання гри в шахи на основі великого обсягу даних. Вони можуть вивчати різні патерни, тактики та стратегії шахової гри. Нейронні мережі можуть бути

навчені грати в шахи за допомогою алгоритму з підсиленням, де вони отримують позитивну або негативну винагороду за кожен хід та змінюють свої ваги, щоб покращити свою гру;

- моделі гри з базами даних. Існують бази даних зі шаховими партіями, які можуть бути використані для аналізу та навчання. Ці бази даних містять інформацію про ходи та стратегії відомих шахових гравців. Моделі можуть використовувати ці дані для визначення оптимальних ходів та розробки нових стратегій гри.

Ці моделі та алгоритми можуть бути поєднані та вдосконалені для створення потужних шахових програм та додатків. Важливо враховувати, що гра в шахи є складною задачею, і вирішення її повністю може бути непрактичним. Однак з використанням вищезгаданих моделей та алгоритмів можна досягти добрих результатів у грі в шахи на мобільних платформах [1].

1.3 Огляд програмних аналогів

Існує безліч додатків для iOS та Android, пов'язаних із шахами, але деякі з них є особливо популярними. Ось деякі з них:

- Chess.com. Це один з найпопулярніших додатків для гри в шахи на мобільних пристроях. Він має велике співтовариство користувачів та пропонує широкий вибір функцій, включаючи гру з комп'ютером та іншими гравцями, навчання шахам та багато іншого;

- Chess Tactics Pro (Puzzles). Це додаток для навчання шаховим тактикам. Він пропонує більше 1000 рівнів складності та допомагає покращити гравецькі навички;

- Chess Free. Це безкоштовний додаток для iOS та Android, який дозволяє грати в шахи з комп'ютером або іншими гравцями, а також налаштувати рівень складності;

- Lichess. Один з найпопулярніших додатків для гри в шахи на мобільних пристроях, доступний на iOS та Android. Цей додаток пропонує

безкоштовну гру з іншими користувачами в режимі онлайн, а також можливість розв'язувати шахові завдання та вивчати теорію гри.

Першим прикладом онлайн-додатку на тему Шахи буде Chess.com (рис. 1.1). Переваги та недоліки онлайн-додатку наведено у таблиці 1.1



Рисунок 1.1 – Chess.com[4]

Таблиця 1.1–Переваги та недоліки додатку Chess.com

Плюси	Мінуси
Велике співтовариство користувачів: Chess.com має широке співтовариство шахових гравців, що дозволяє знайти багато суперників та партнерів для гри.	Платні опції: деякі функції та можливості на Chess.com є доступними тільки за певну плату.
Різні режими гри: додаток пропонує різні режими гри, включаючи гру з комп'ютером, гру з іншими гравцями онлайн та навчання шахам.	Реклама: у безкоштовній версії Chess.com можуть з'являтися рекламні банери та відеоролики.
Навчальні матеріали: Chess.com має багато навчальних матеріалів, пазли та статей, які допомагають покращити навички гри.	Обмежені можливості безкоштовної версії: деякі функції та матеріали можуть бути доступні лише для платних користувачів.
Висока якість гри: додаток Chess.com пропонує високу якість гри зі стабільним з'єднанням та хорошою графікою.	Вимоги до Інтернет-з'єднання: для гри в режимі онлайн необхідне стабільне та швидке Інтернет-з'єднання.

Другим аналогом є мобільний додаток Chess Tactics Pro (рис. 1.2).

Переваги та недоліки онлайн-додатку Chess.com наведено у таблиці 1.2



Рисунок 1.2 – Chess Tactics Pro (Puzzles)

Таблиця 1.2 – Переваги та недоліки додатку ChessTacticsPro

Плюси	Мінуси
Навчання шаховим тактикам: додаток пропонує більше 1000 рівнів складності для вивчення шахових тактик та покращення гравецьких навичок.	Відсутність мультиплеєра: додаток не має можливості грати з іншими гравцями онлайн, а фокусується лише на навчанні.
Зручний інтерфейс: додаток має зрозумілий та зручний інтерфейс, що дозволяє легко розуміти та розв'язувати шахові головоломки.	Обмежені можливості: у порівнянні з іншими додатками, ChessTacticsPro може мати обмежені функціональні можливості.
Прогрес та статистика: додаток відстежує ваш прогрес у вирішенні головоломок і надає статистику, яка допомагає оцінити ваші успіхи та зростання.	Відсутність додаткових режимів гри: додаток зосереджений на головоломках і не має інших режимів гри або навчальних матеріалів.
Зручна навігація та рівні складності: додаток пропонує зручну навігацію між рівнями та варіантами завдань, а також можливість вибору рівня складності.	Відсутність багатомовності: додаток може бути обмежений певними мовами та не підтримувати багатомовність.

Інший аналог який можна розглянути це додаток Chess Free(рис. 1.3).

Переваги та недоліки онлайн-додатку наведено у таблиці 1.3.



Рисунок 1.3 – Chess Free

Таблиця 1.3 – Переваги та недоліки додатку ChessFree

Плюси	Мінуси
Безкоштовний: додаток ChessFree є безкоштовним для встановлення та використання, що дозволяє грати в шахи без додаткових витрат.	Обмежені функції: у порівнянні з платними аналогами, додаток може мати обмежені функціональні можливості та менший набір функцій.
Гра з комп'ютером та іншими гравцями: додаток дозволяє грати в шахи з комп'ютером на різних рівнях складності або з іншими гравцями, які використовують додаток.	Відсутність режиму онлайн: додаток не надає можливості грати в режимі онлайн з гравцями поза додатком або на віддалених пристроях.
Налаштування рівня складності: додаток дозволяє вибирати рівень складності гри з комп'ютером, що дозволяє підлаштовувати гру під власні навички та здатності.	Відсутність навчальних матеріалів: додаток може не мати вбудованих навчальних матеріалів або пояснень для початківців, які хочуть навчитися грати в шахи.
Зручний інтерфейс: додаток має зрозумілий та зручний інтерфейс, що дозволяє легко розуміти та грати в шахи навіть початківцям.	Наявність реклами: додаток може містити рекламу, яка може бути відображена під час гри та впливати на загальне враження користувача.

Інший аналог подібного який можна розглянути це онлайн-додаток Lichess.org (рис 1.4). Переваги та недоліки онлайн-додатку наведено у таблиці 1.4.

Таблиця 1.4 – Переваги та недоліки додатку Lichess.org

Плюси	Мінуси
Безкоштовний та відкритий додаток: Lichess.org є безкоштовним для встановлення та використання, а також відкритим для спільноти, що дозволяє залучати співрозробників та внести свій внесок.	Відсутність офіційного додатка: наразі відсутня офіційна мобільна програма для Lichess.org, але є мобільна версія веб-сайту.
Гра в режимі онлайн: Lichess.org надає можливість грати в режимі онлайн з іншими користувачами з усього світу, що дозволяє знайти суперників будь-якого рівня.	Обмежена офлайн-гра: відсутність повноцінної офлайн-гри, оскільки для гри потрібне підключення до Інтернету.
Велика спільнота користувачів: Lichess.org має широке співтовариство активних гравців, що дозволяє знаходити суперників швидко та взаємодіяти з іншими гравцями.	Відсутність деяких функцій: деякі функції, які присутні в інших додатках, можуть бути відсутніми у Lichess.org, наприклад, деякі навчальні матеріали чи функції аналізу.
Навчальні ресурси та інструменти: Lichess.org пропонує різноманітні навчальні ресурси, такі як шахові пазли, статті, відео та тренувальні режими, що допомагають покращити шахові навички.	Відсутність офіційного додатка: наразі відсутня офіційна мобільна програма для Lichess.org, але є мобільна версія веб-сайту.
Безрекламний досвід: Lichess.org не містить рекламу, що забезпечує безперешкодний геймплей та не порушує враження від гри.	



Рисунок 1.4 – Lichess.org[7]

Порівняльну таблицю всіх аналогів наведено у таблиці 1.5.

Таблиця 1.5– Порівняння аналогів

Аналоги Критерії порівняння	Chess.com	Chess Tactics Pro	Chess Free	Lichess.org
Користувачі	Має велике співтовариство, що дозволяє знайти багато суперників та партнерів для гри	Відсутність мультиплеєра: додаток не має можливості грати з іншими гравцями онлайн, а фокусується лише на навчанні	Додаток дозволяє грати в шахи з комп'ютером на різних рівнях складності або з іншими гравцями	Надає можливість грати в режимі онлайн з іншими користувачами з усього світу, що дозволяє знайти суперників будь-якого рівня
Режими	Гра з комп'ютером, гра з іншими гравцями онлайн, навчання шахам	Відсутні додаткові режими, лише гра	дозволяє вибирати рівень складності гри з комп'ютером	Різноманітнішахові пазли, ігри проти користувачів та комп'ютеру та інші тренувальні режими
Навчальні матеріали	Багато навчальних матеріалів, пазли та статей	Більше 1000 рівнів складності для вивчення шахових тактик та покращення навичок	не має вбудованих навчальних матеріалів для початківців	різноманітні навчальні ресурси: шахові пазли, статті та тренувальні режими
Безкоштовна версія / Платні опції	Обмежені можливості безкоштовної версії: деякі функції та матеріали можуть бути доступні лише для платних користувачів			Є безкоштовним для встановлення та використання, а також відкритим для спільноти
Реклама	У безкоштовній версії може містити			Не містить
Особливі технічні вимоги	в режимі онлайн необхідне стабільне та швидке Інтернет з'єднання	Через відсутність мультиплеєра інтернет не потрібен і добре працює навіть на старих версіях телефонів	Не має особливих технічних вимог	

1.4 Постановка завдання

Для повноцінної гри в шахи, враховуючи загальні правила гри, наступні функції та можливості, повинні бути реалізовані наступні функції: [5]

а) розміщення фігур:

- 1) початкове розміщення фігур на дошці;
- 2) розміщення фігур залежно від кольорних гравців (білий і чорний);
- 3) врахування початкових позицій кожної фігури (позиції стартових позицій пішок, розташування королів, тощо);

б) пересування фігур:

- 1) перевірка можливих ходів для кожної фігури;
- 2) обмеження руху фігур згідно з їхніми правилами;
- 3) перевірка чи хід валідний, зокрема перевірка на шах;

в) взяття фігур:

- 1) реалізація можливості взяття фігур під час переміщення;
- 2) перевірка на легальність взяття фігур згідно з правилами;

г) правила гри:

- 1) перевірка на шах та мат;
- 2) виведення повідомлень про шах та мат;
- 3) перевірка на пат (нічия) та виведення повідомлень про пат;

д) перетворення пішок:

- 1) можливість перетворення пішок на інші фігури під час досягнення кінця дошки;

е) відслідковування поточного гравця;

- 1) перемикання між гравцями після кожного ходу;
- 2) відображення поточного гравця;

ж) завершення гри:

- 1) визначення переможця;
- 2) виведення повідомлення про переможця;

3) зупинка гри після завершення.

1.5 Висновки за розділом 1

У розділі 1 було розглянуто платформу для створення додатків для гри в шахи, а також оглянуто деякі популярні додатки пов'язані з шахами, що дозволила виділити основні вимоги до проєкту.

Загалом, можна зробити наступні висновки:

- Unity є потужною та гнучкою платформою для розробки додатків, включаючи ігри в шахи. Вона надає розробникам доступ до різноманітних інструментів та можливостей, що дозволяють створювати кроссплатформові додатки для різних пристроїв;
- Unity підтримує створення 3D-графіки, анімацій, фізики та інших елементів, що можуть бути використані для створення шахівниці та фігур;
- Unity має інтуїтивно зрозумілий інтерфейс розробки та багато ресурсів, таких як документація та онлайн-курси, які допомагають розробникам швидше освоїти платформу;
- існують різні додатки пов'язані з шахами, такі як Chess.com, Chess Tactics Pro і Lichess.org. Кожен з них має свої переваги та недоліки, такі як наявність різних функцій, режимів гри та навчальних матеріалів.

Загалом, вибір платформи та додатка для створення гри в шахи залежить від конкретних потреб та вимог розробника. Unity може бути привабливим варіантом для розробки кроссплатформових додатків зі зручним інтерфейсом та багатими можливостями для налаштування графіки та ігрових елементів. Також варто розглянути інші технічні засоби, щоб знайти оптимальне рішення для конкретного проєкту.

2 ТЕХНІЧНІ ТА ПРОГРАМНІ ЗАСОБИ РОЗРОБКИ

Для розробки додатку для гри в шахи існує різноманітні технічні та програмні засоби, які можуть бути використані для створення функціонального та естетичного додатку. Ось декілька з них:

- вибір мови програмування: залежить від конкретних потреб розробки. Для створення мобільних додатків для iOS та Android можуть бути використані мови програмування, такі як Swift або Objective-C для iOS та Java або Kotlin для Android. Для розробки додатку для ПК можуть використовуватися мови, такі як C++ або C#;

- інтегровані середовища розробки (IDE): IDE надають зручне середовище для програмування та розробки додатків. Деякі популярні IDE для розробки мобільних додатків включають Xcode для iOS та Android Studio для Android. Для розробки додатків для ПК можна використовувати IDE, такі як VS або Unity;

- фреймворки та бібліотеки: Фреймворки та бібліотеки дозволяють розробникам ефективно використовувати готові компоненти та функціонал для створення додатків. Наприклад, для розробки мобільних додатків можна використовувати фреймворки, такі як React Native або Flutter, які дозволяють створювати кросплатформені додатки. У разі розробки додатку для ПК використовуються фреймворки та бібліотеки, які підтримують створення графічного інтерфейсу, наприклад, WinForms або WPF для платформи Windows;

- 3D-двигуни: Якщо планується реалістичне візуальне відображення шахової дошки та фігур, можна використовувати 3D-движки, такі як Unity або Unreal Engine. Вони надають розширені можливості для роботи з 3D-моделями, освітленням, анімацією та фізикою, що дозволяє створити реалістичну та іммерсивну гру;

- бази даних: Для зберігання інформації про гравців, їхні рейтинги, історію партій та іншу статистику можна використовувати бази даних. Популярні

бази даних, такі як MySQL або SQLite, можуть бути використані для створення ефективного та надійного збереження даних;

– мережеві технології: Якщо бажаєте надати можливість грати в шахи онлайн або взаємодіяти з іншими гравцями, потрібно використовувати мережеві технології, такі як TCP/IP або WebSocket, для забезпечення комунікації між клієнтами.

Ці технологічні та програмні засоби надають широкі можливості для розробки додатків для гри в шахи. Вибір конкретних засобів залежить від потреб проекту, компетентностей розробника та цільової платформи. Важливо ретельно обрати засоби розробки, що дозволить створити функціональний, естетичний та користувацьки дружлюбний додаток для шахів.

2.1 Вибір мови програмування

Порівняння мов програмування наведено у таблиці 2.1

Таблиця 2.1– Порівняння мов програмування

Мова Критерії порівняння	C#	C++	Java	Python	JavaScript
Об'єктно-Орієнтованість	Так	Так	Так	Так	Ні
Платформи та фрейм-ворки	.NET, Unity, Xamarin	Unreal Engine, Qt	Android, Spring	Pygame, Django	Node.js, React
Синтаксис	Подібний до C/C++/Java	Подібний до C	Подібний до C/C++	Простий, експресивний	Синтаксично різноманітний, динамічний
Спільнота розробників	Велика, активна	Велика, активна	Велика, активна	Велика, активна	Велика, активна

Продовження таблиці 2.1

Мова Критерії порівняння	C#	C++	Java	Python	JavaScript
Приклади використання	Розробка ігор, вебдодатків, десктопних додатків	Розробка ігор, системного програмування, вбудованих систем	Розробка ігор, вебдодатків, мобільних додатків	Розробка ігор, вебдодатків, наукові обчислення	Розробка вебдодатків, браузерних ігор

C# - це мова програмування, розроблена компанією Microsoft, яка широко використовується для розробки різноманітних додатків, включаючи ігри. C# має численні переваги для розробки додатків для гри в шахи:

- синтаксис. C# має синтаксис, подібний до інших мов програмування з роду C, що робить його легким у вивченні для розробників з досвідом у таких мовах, як C, C++ або Java;
- об'єктно-орієнтоване програмування. C# підтримує концепції об'єктно-орієнтованого програмування, що дозволяє організовувати код у логічні структури, такі як класи, об'єкти, успадкування, поліморфізм тощо. Це сприяє створенню структурованого та модульного коду;
- платформа .NET. C# базується на платформі .NET, що надає розробникам багатий набір функцій та бібліотек для роботи зі звуком, графікою, мережами, базами даних та багато іншого. Це полегшує розробку ігрових додатків, оскільки вже існує багато готових компонентів для реалізації різноманітних функцій;
- широке співтовариство. C# має велике та активне співтовариство розробників, що означає наявність багато документації, підручників, форумів та онлайн-ресурсів, де можна знайти допомогу та поради щодо розробки.

Всі ці переваги роблять C# привабливим вибором для розробки додатків для гри в шахи, забезпечуючи зручну та потужну мову програмування [8].

2.2 Вибір інтегрованого середовища розробки

Існує велика кількість інтегрованих середовищ розробки, таких як VisualStudio, IntelliJIDEA, Eclipse, PyCharm, Xcode та інші. Кожне з них пристосовано і підходить для різних операційних систем, мов програмування тощо. Порівняльна таблиця IDE наведена у таблиці 2.2 [8] [12].

Таблиця 2.2 – Порівняння IDE

Мова Критерії порівняння	Visual Studio	IntelliJ IDEA	Eclipse	PyCharm	Xcode
Підтримувані мови програмування	C#, C++, Java, Python та інші	Java, Kotlin, Scala та інші	Java, C++, PHP та інші	Python, JavaScript, HTML/CSS та інші	Swift, Objective-C, C++ та інші
Інтеграція з платформами	.NET, Unity, Xamarin	Android, Spring, JavaFX	Android, Java, PHP, JavaScript та інші	Django, Flask, Web та інші	iOS, macOS, watchOS, tvOS
Функціональність та продуктивність	Висока	Висока	Середня	Висока	Середня
Підтримка плагінів	Так	Так	Так	Так	Ні
Налагоджування та профілювання	Повна підтримка	Повна підтримка	Повна підтримка	Повна підтримка	Повна підтримка

Visual Studio - це одне з найпоширеніших інтегрованих середовищ розробки (IDE), розроблене компанією Microsoft, яке надає розробникам багатий набір інструментів для програмування різноманітних додатків. В поєднанні з платформою Unity, VS стає потужним інструментом для розробки додатків для гри в шахи. Ось кілька переваг такого поєднання:

- інтеграція з Unity. VS повністю інтегрується з Unity, що дозволяє зручно розробляти скрипти та ігрову логіку безпосередньо в середовищі Unity. Це

спрощує взаємодію з іншими елементами гри та дозволяє швидше вносити зміни та перевіряти результати;

- підтримка мови C#. VS є потужним інструментом для розробки на мові C#. Він надає автоматичне завершення коду, розширені можливості налагодження, відстежування помилок та інші корисні функції, що полегшують процес розробки;

- багатофункціональність. VS надає розробникам широкий спектр функцій, таких як керування версіями, вбудовані інструменти для роботи з базами даних, інтеграцію з системами контролю версій, засоби для автоматичного тестування та багато іншого. Всі ці функції сприяють покращенню продуктивності та якості розробки;

- наявність плагінів. VS підтримує розширення за допомогою плагінів, що дозволяє розширити його функціональність. Для Unity існують спеціальні плагіни, які полегшують роботу з Unity-проектами, забезпечують інтеграцію з іншими інструментами та надають додаткові можливості для розробки.

Використання Visual Studio у поєднанні з Unity дозволяє розробникам створювати потужні та привабливі додатки для гри в шахи з більшою ефективністю та зручністю. За допомогою цих інструментів розробники можуть швидко створювати, налагоджувати та вдосконалювати свої проекти шахів, забезпечуючи високу якість та задоволення користувачів.

2.3 Складові платформи Unity

Юніті (Unity) - це програмний продукт, який забезпечує все необхідне для створення ігор, доповненої реальності та рішень віртуальної реальності. У цьому розділі розглянуто об'єкти, ієрархії, ассети, спрайти та інші функціональні можливості в Unity.

2.3.1 Об'єкти в Unity

Основними елементами всіх ігрових проведення рушіїв є об'єкти або ігрові об'єкти. Об'єкти складаються з компонентів, що робить їх унікальними і дає можливість додавати різні функціональні можливості до ігрових об'єктів. У Unity ігрові об'єкти можуть бути створені для будь-якого візуального елементу, що може бути розміщений на сцені, такого як персонаж, перешкоди або об'єкти навколишнього середовища.

Об'єкти в Unity можуть бути різноманітні і можуть мати самостійне функціональне призначення або використовуватися як частина штучного інтелекту, анімації, системи вводу або скриптів. Об'єкти можуть бути екземплярами класів, наприклад, єдиного об'єкта, вікна, текстури чи полотна.

Незалежно від того, яке призначення має об'єкт в грі, його всі компоненти можна змінювати на панелі інспектора. У панелі інспектора ви можете налаштовувати компоненти, досліджувати властивості, додавати нові компоненти до об'єкта та змінювати всі інші параметри.

Важливою особливістю ігрових об'єктів в Unity є їх здатність взаємодіяти між собою та з окремими компонентами в рамках ігрового рушія. Це дозволяє розробникам створювати складні ефекти, інтерактивні ігри та багато іншого [11].

2.3.2 Панель Інспектора

Панель інспектора - це основне вікно, яке забезпечує безліч інструментів для розробки ігор в ігрових рушіях. Він відображає детальну інформацію про виділені об'єкти в редакторі та дозволяє змінювати їх властивості.

У панелі інспектора можна редагувати компоненти, додані до об'єктів, та змінювати їх параметри. Ви можете налаштовувати властивості об'єктів, змінювати спрайти, відео та текстові елементи, а також встановлювати скрипти на об'єкт, редагувати їх параметри та тимчасово відключати їх. Також з цієї панелі ви

можете налаштовувати властивості середовища, камери, ефектів, екрана, освітлення та іншого.

Вікно інспектора зазвичай складається з двох частин:

- поле вибору об'єкта у верхній частині панелі;
- розділ властивостей об'єкта у нижній частині панелі.

Ви можете показувати або приховувати різні розкриваються списки, щоб більш ефективно працювати з об'єктами. Також ви можете змінювати та встановлювати нові властивості, створювати та змінювати змінні, налаштовувати анімації та багато іншого.

Панель інспектора - це один із найбільш використовуваних та важливих інструментів для розробки ігор в ігрових рушіях, і він дозволяє розробникам повністю контролювати свої ігрові об'єкти та елементи, спрощуючи процес розробки ігор та підвищуючи їх якість. (рис 2.1)

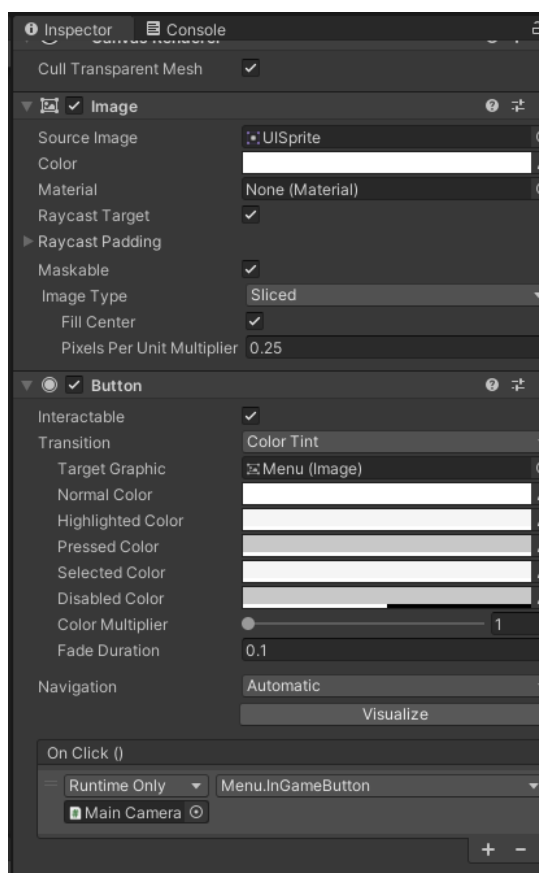


Рисунок 2.1 –Зовнішній вигляд панелі Інспектор

2.3.3 Ієрархія в Unity

Ієрархія в Unity - це структура, яка відображає всі об'єкти на сцені та їх взаємодію. Об'єкти розміщуються в ієрархії відповідно до їх взаємного розташування на сцені: батьківський об'єкт містить дочірні об'єкти, які можуть містити ще інші дочірні об'єкти.

Кожен об'єкт в ієрархії має свої атрибути, які відображають його параметри та властивості. За допомогою ієрархії можна легко змінювати, додавати та видаляти об'єкти на сцені, а також керувати їх властивостями та взаємодією між ними.

Крім того, ієрархія використовується для створення складних об'єктів та систем у грі: наприклад, для створення ієрархії зі складовими об'єктами, які моделюють рух персонажа.

Також ієрархія дозволяє контролювати взаємодію між об'єктами. Якщо дочірній об'єкт змінюється, то це може вплинути на всі батьківські об'єкти в ієрархії, і навпаки. Наприклад, якщо змінити розмір батьківського об'єкта, то всі дочірні об'єкти автоматично пристосуються до нового розміру[13].

Приклад ієрархії наведено на рис. 2.2.

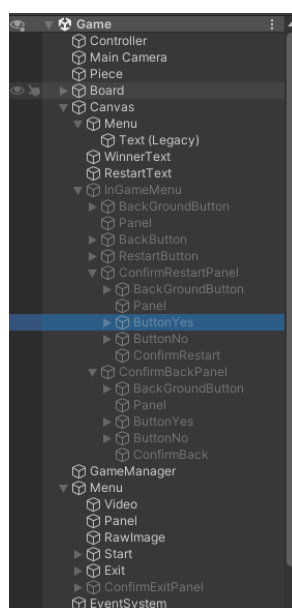


Рисунок 2.2 – Ієрархія в Unity

2.3.4 Асети в Unity

Асети - це графічні та звукові ресурси, такі як зображення, звукові ефекти, моделі персонажів тощо, які імпортуються в Unity для використання у створенні ігор. Ці асети можуть бути куплені в Asset Store, знайдені в Інтернеті або створені самостійно.

У Unity є дві основні категорії асетів: 3D-асети та 2D-асети. 3D-асети використовуються для створення тривимірних моделей та об'єктів, а 2D-асети для створення 2D-екранів, ігрових персонажів та інших графічних елементів.

Після того, як асет було імпортовано в Unity, його можна використовувати, створюючи об'єкти на сцені гри. Наприклад, створивши 3D-об'єкт, можна прикріпити до нього текстури, анімації, звукові ефекти тощо, щоб зробити його більш динамічним та реалістичним.

Щоб додати новий асет до вашого проєкту в Unity, потрібно вибрати опцію "Import new Asset" у розділі Assets в меню Project. Після того, як асет буде імпортований в проєкт, його можна перетягнути на об'єкт на сцені або в ієрархії.

Асети є дуже важливим елементом в створенні ігор та інтерактивних додатків в Unity. Вони дозволяють створювати складні моделі, ефекти та звукові обробки, що додають реалістичності та інтерактивності в гру. Приклад розділу Асетів наведено на рис. 2.3.1 [10].

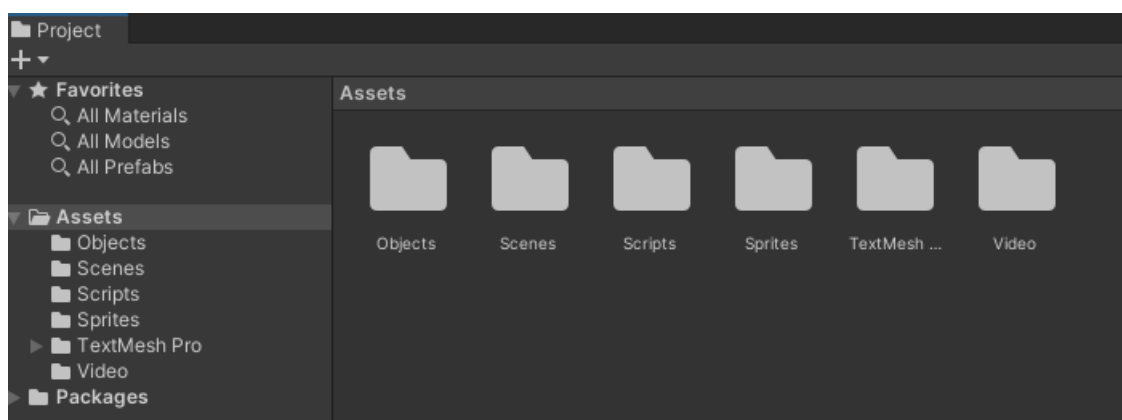


Рисунок 2.3.1 – Асети в Unity

2.3.5 Спрайти в Unity

Спрайти - це 2D-графічні зображення, які використовуються у створенні ігор та додатків. У Unity спрайти використовуються для створення ігрових персонажів, фонів та інших графічних елементів.

У Unity можна використовувати два типи спрайтів. Перший тип - звичайні спрайти, які складаються з одного зображення та відображаються на сцені як 2D-об'єкти. Другий тип - складений спрайт або Sprite Sheet, який містить кілька спрайтів в одному зображенні. У складеному спрайті кожен окремий спрайт можна використовувати як окремий елемент в грі.

Для створення нового спрайту в Unity потрібно вибрати опцію "Create new Sprite" в розділі Assets в меню Project або перетягнути зображення в ієрархію на сцені. Далі необхідно налаштувати параметри спрайту, такі як розмір, масштаб, відображення та інші властивості.

Одним зі способів використання спрайтів в Unity є їх розміщення на 2D-об'єктах на сцені. Наприклад, спрайт можна прикріпити до GameObject на сцені за допомогою компонента Sprite Renderer. Крім того, спрайти можна анімувати, змінюючи їх позицію, розмір, напрямок та інші параметри.

Спрайти є дуже важливим елементом в створенні 2D-ігор та графічних додатків в Unity. Потужність та гнучкість цього інструмента дозволяє створювати інтерактивні та цікаві графічні анімації та ефекти. Приклад меню спрайтів наведено на рис. 2.3.2 [14].



Рисунок 2.3.2 – Спрайти в Unity

2.3.6 Кастомізація робочої області

Unity має гнучку систему налаштування робочої області, яка дає змогу користувачам налаштовувати різні інструменти та редактори для створення ігор та додатків. Також можна змінювати розміщення вікна та вибирати найбільш підходящий стиль оформлення.

2.4 Компоненти в Unity

Є багато важливих компонентів у юніті, такі як transform, camera, sprite, video player, canvas тощо. Ось короткий опис таких компонентів:

- компонент transform. Відповідає за позиціонування, масштабування та обертання об'єкта в 3D-просторі (рис 2.4.1);
- компонент camera. Забезпечує відображення сцени на екрані. Він налаштовує параметри візуалізації, такі як перспектива, близькість та дальність (рис 2.4.2);
- компонент sprite. Відповідає за відображення двовимірних графічних об'єктів в сцені. Цей компонент також може використовуватися для створення анімації (рис 2.4.3);
- компонент Video Player. Обробляє відео в реальному часі в іграх. Він може відтворювати відео та додавати теги та субтитри (рис 2.4.4);
- компонент canvas. Відображає двовимірні графічні об'єкти (наприклад, текст, зображення). Він підходить для створення інтерфейсів користувацьких ігор (рис 2.4.5);
- компонент Text. Відображає текстові елементи на екрані. Він дозволяє налаштовувати параметри тексту, такі як шрифт, колір, розмір та вирівнювання (рис 2.4.6);

- компонент Animation. Використовується для створення та управління анімаціями. Він дозволяє визначати ключові кадри, переходи між ними та інші параметри анімації (рис 2.4.7) [15];
- компонент Rigidbody. Відповідає за фізичну поведінку об'єкта. Він надає можливість об'єкту реагувати на гравітацію, сили, столкнення та інші фізичні ефекти;
- компонент Collider. Використовується для визначення областей стовпчика для об'єкта. Він може бути використаний для виявлення столкнень та взаємодії з іншими об'єктами в сцені;
- компонент Audio. Використовується для відтворення звукових ефектів та музики в грі. Цей компонент дозволяє налаштовувати параметри звуку, такі як гучність, позиція та ефекти звуку (рис 2.4.8);
- Компонент Light. Використовується для створення та налаштування освітлення в сцені. Він може визначати тип світла, його інтенсивність, колір та інші параметри освітлення;
- Компонент Particle System. Використовується для створення та управління ефектами частинок, таких як дим, вогонь, вибухи та інші. Він дозволяє налаштовувати параметри частинок, такі як швидкість, розмір, колір та життєвий цикл.

Для додавання скриптів у вигляді компонентів потрібно.

Крок 1. Натиснути правою кнопкою миші на об'єкт в ієрархії та вибрати "Додати компонент";

Крок 2. Вибрати тип компонента "Скрипт";

Крок 3. Ввести назву скрипта та натиснути на кнопку "Створити та додати";

Крок 4. У відкритому редакторі скриптів можна написати свій власний код для компонента;

Крок 5. Якщо обрати створений компонент, то можна налаштувати змінні, доступні зі скрипта, в редакторі компонентів, який з'явиться справа на панелі інспектора [16].

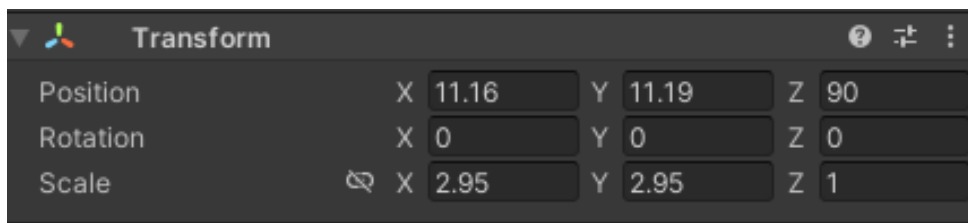


Рисунок 2.4.1 – Компонент transform

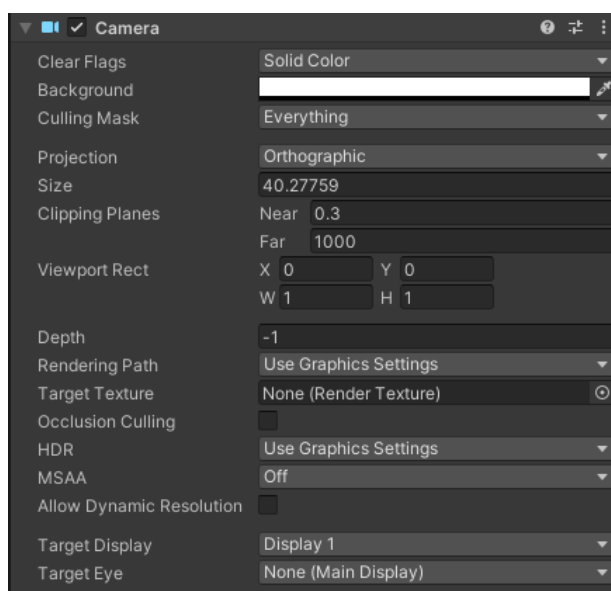


Рисунок 2.4.2 – Компонент camera

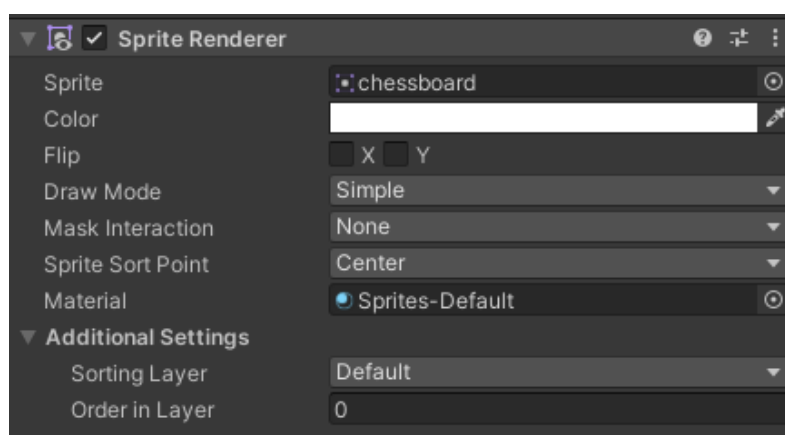


Рисунок 2.4.3 – Компонент Sprite

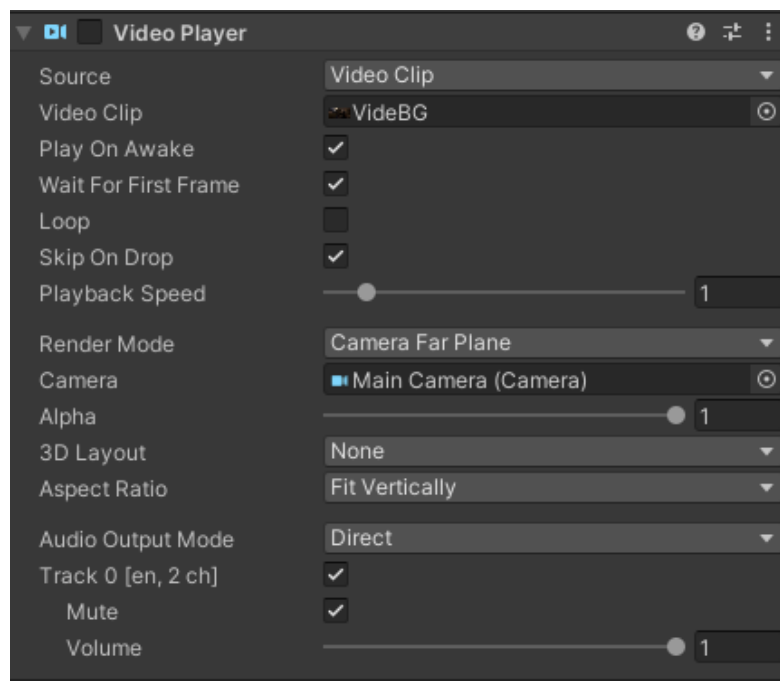


Рисунок 2.4.4 – Компонент Video Player

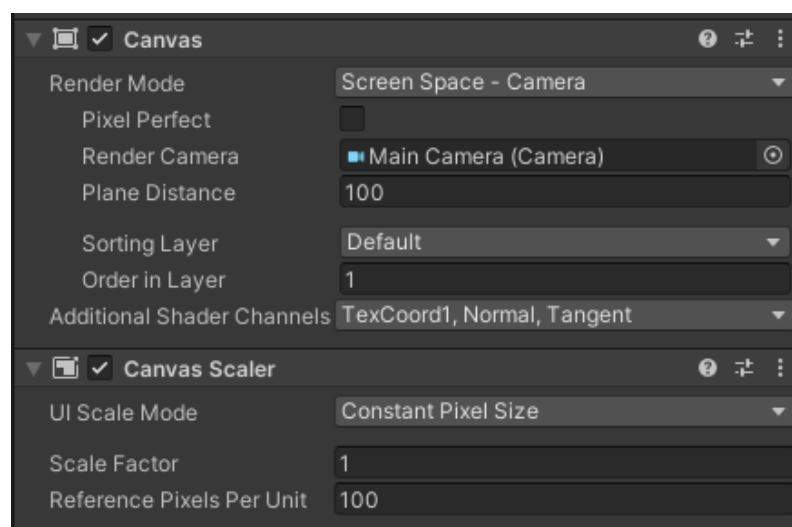


Рисунок 2.4.5 – Компонент Canvas

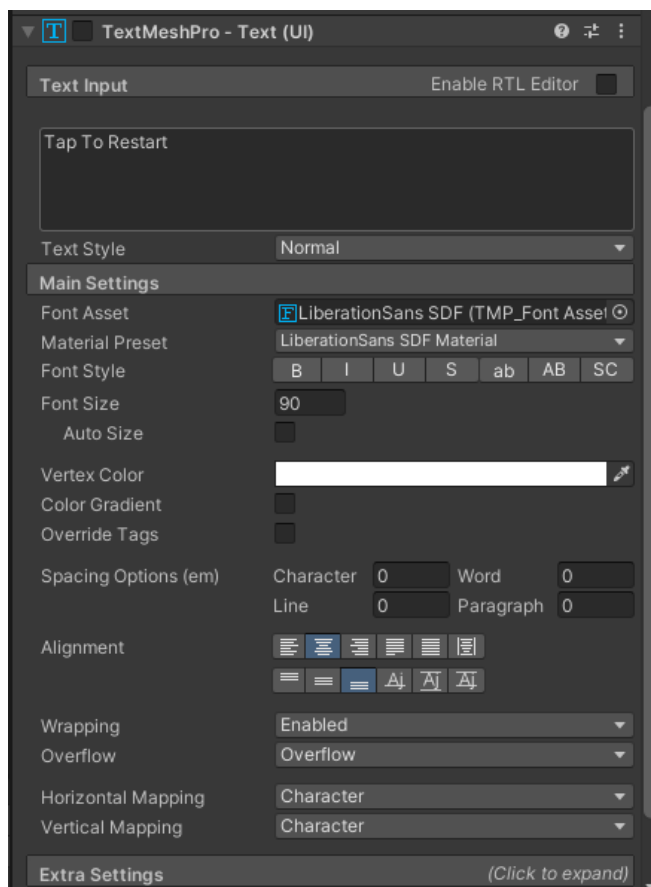


Рисунок 2.4.6 – Компонент Text

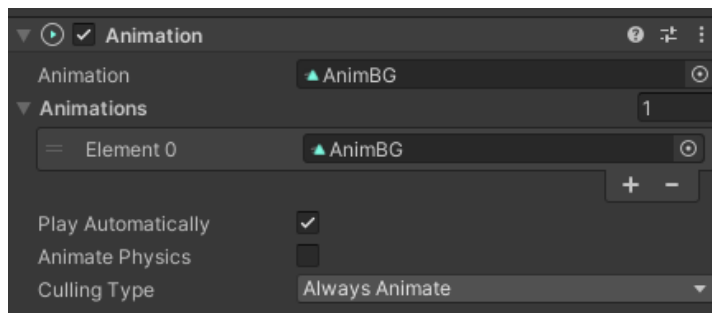


Рисунок 2.4.7 – Компонент Animation

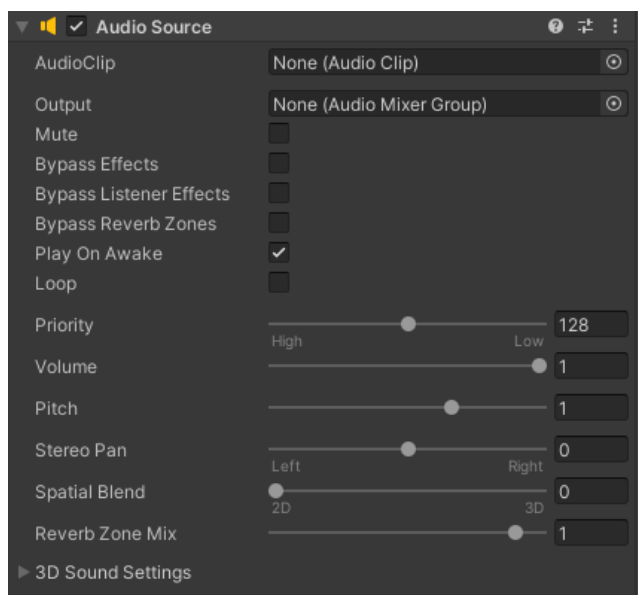


Рисунок 2.4.8 – Компонент Audio

2.5 Висновки за розділом 2

Unity та C# надають розробникам зручну та потужну платформу для створення ігор та інтерактивних додатків. Unity є одним з найпопулярніших гральних движків, який має широку екосистему та готові компоненти, що спрощують процес розробки. Завдяки крос-платформеності Unity розробники можуть створювати ігри та додатки для різних платформ, що дозволяє досягти більшої аудиторії користувачів.

Мова програмування C#, яка використовується в Unity, має багато переваг. C# є типізованою мовою з сильною перевіркою типів, що сприяє більш надійній розробці. Вона також має широкий набір бібліотек та інструментів, що робить її потужним інструментом для розробки ігор та додатків. Завдяки синтаксису C#, розробники можуть створювати лаконічний і зрозумілий код, що сприяє його легкості читання та підтримки проєктів.

Unity та C# також надають широкі можливості для створення візуальних ефектів, анімацій, фізики та штучного інтелекту, що дозволяє розробникам створювати цікаві та захоплюючі гральні світи. Крім того, Unity має розвинуту спільноту розробників, яка пропонує велику кількість ресурсів, уроків та документації для вивчення та підтримки.

В цілому, використання Unity та C# забезпечує розробникам інструменти, необхідні для створення високоякісних ігор та інтерактивних додатків. Це зручне поєднання, яке дозволяє скоротити час та зусилля, витрачені на розробку, та досягти бажаних результатів за допомогою потужних інструментів та широкої підтримки спільноти.

3 РЕАЛІЗАЦІЯ ПРОЄКТУ

3.1 Ієрархія та об'єкти проєкту

На рисунку 3.1 Наведено всю ієрархію розробленого додатку «Шахи».

Приклади компонентів, що використовуються у об'єктах проєкту, наведено на рисунках 3.2-3.5.

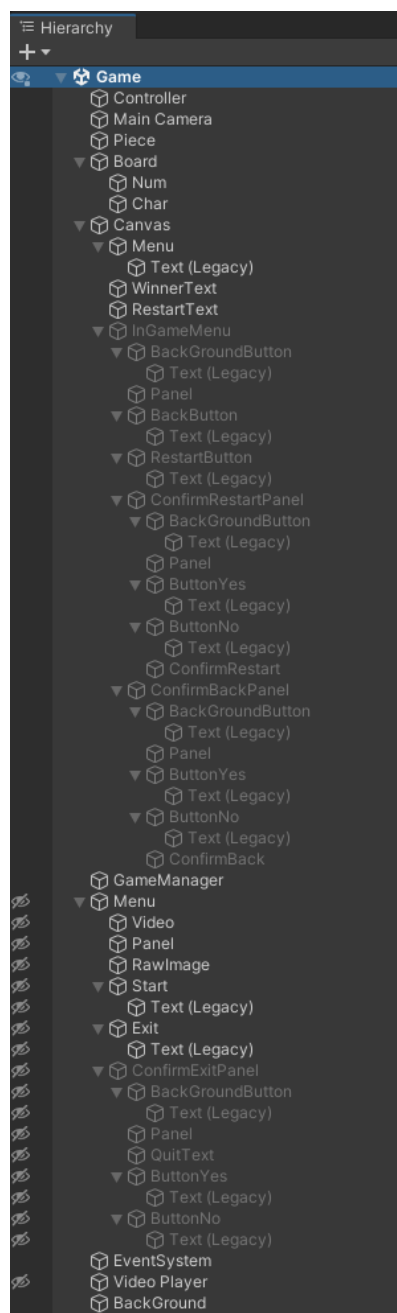


Рисунок 3.1 Ієрархія додатку «Шахи»

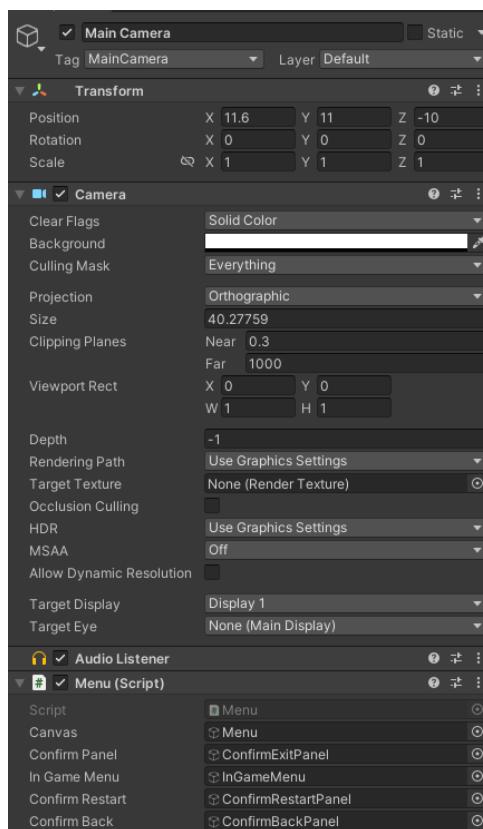


Рисунок 3.2 Компоненты об'єкту «MainCamera»

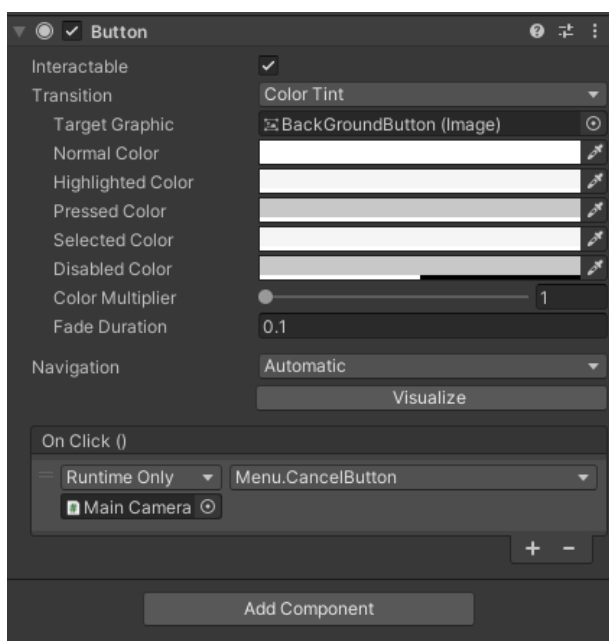


Рисунок 3.3 Компонент кнопки «InGameMenu»

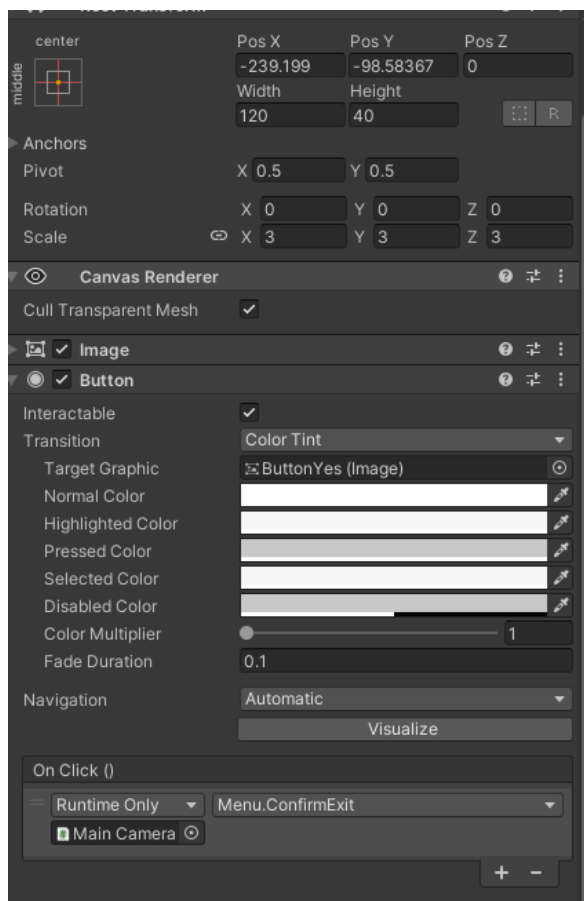


Рисунок 3.4 Компоненти кнопки виходу з додатка

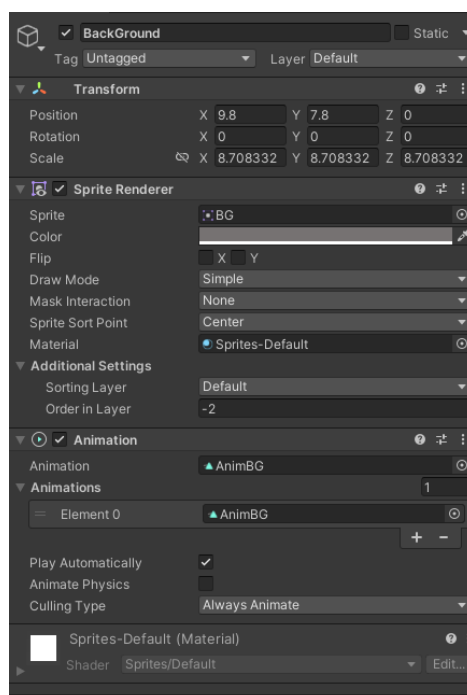


Рисунок 3.5 Компоненти об'єкту «Background»

3.2 Розроблені класи

В ході роботи були розроблені численні класи, схема яких представлена на рисунку 3.6.

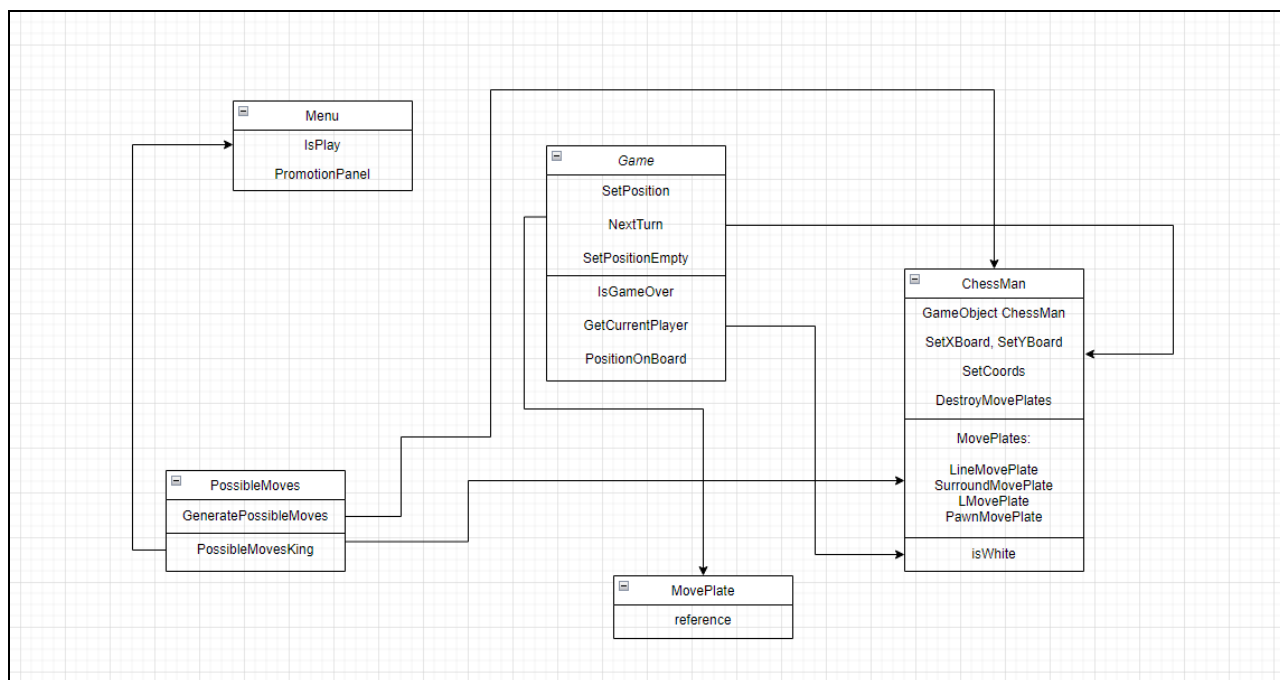


Рисунок 3.6 Схема залежностей класів

3.2.1 Клас "Game"

Клас "Game" відповідає за управління грою шахи. Він містить різні методи, що керують розташуванням шахових фігур, ходами гравців та завершенням гри. Розглянемо кожен метод окремо:

а) Start:

- 1) цей метод викликається при запуску гри і встановлює початкове розташування шахових фігур для кожного гравця;
- 2) створюються об'єкти для кожної фігури та встановлюються їх початкові позиції на шаховій дошці;

3) координати фігур зберігаються в двовимірному масиві "pos" та використовуються для визначення розташування фігур на дошці;

4) для кожної фігури викликається метод "SetPosition", який встановлює фігуру на відповідну позицію на дошці;

б) Create:

1) цей метод створює об'єкт шахової фігури та встановлює його на певну позицію на дошці;

2) використовуючи параметри "name", "x" і "y", створюється об'єкт фігури за допомогою інстанціювання префабу "ChessPiece";

3) за допомогою компонента "ChessMan" встановлюються властивості фігури, такі як назва, координати на дошці та стан активності;

4) об'єкт фігури додається до батьківського об'єкту "Piece" для керування розташуванням фігур на екрані;

5) повертається створений об'єкт фігури;

в) CreatePawn:

1) цей метод аналогічний методу "Create", але використовується для створення пішаків;

2) додатково передається параметр "isFirstMove", який визначає, чи це перший хід пішака;

3) встановлюються властивості пішака за допомогою компонента "ChessMan", включаючи стан першого ходу;

4) об'єкт пішака додається до батьківського об'єкту "Piece" для керування розташуванням фігур на екрані;

5) повертається створений об'єкт пішака;

г) SetPosition:

1) цей метод встановлює фігуру на певну позицію на дошці;

2) за допомогою компонента "ChessMan" отримуються координати фігури на дошці;

3) встановлюється зв'язок між фігурою та відповідною позицією у двовимірному масиві "pos";

- 4) також зберігається посилання на об'єкт "ChessMan" у двовимірному масиві "chessMans";
- д) GetBoard. Цей метод повертає двовимірний масив "chessMans", який містить посилання на всі шахові фігури на дошці;
- е) SetPositionEmpty. Цей метод встановлює відповідну позицію у двовимірному масиві "pos" як порожню;
- ж) GetPosition. Цей метод повертає об'єкт фігури на певній позиції на дошці;
- з) PositionOnBoard:
 - 1) цей метод перевіряє, чи знаходиться позиція на дошці;
 - 2) перевіряється, чи координати знаходяться в межах розмірів двовимірного масиву "pos";
 - 3) якщо координати відповідають межах дошки, повертається значення true, в іншому випадку – false;
- и) GetCurrentPlayer. Цей метод повертає поточного гравця, який має право на хід (чорні або білі);
- к) IsGameOver. Цей метод повертає значення, що вказує, чи завершена гра;
- л) NextTurn:
 - 1) цей метод перемикає хід наступного гравця;
 - 2) якщо поточний гравець - "white", то наступний стає "black" і навпаки;
- м) Update:
 - 1) цей метод викликається кожний кадр гри;
 - 2) якщо гра завершена і гравець натиснув ліву кнопку миші, гра перезавантажується;
- н) GetArray:
 - 1) цей метод повертає об'єкт "ChessMan" з двовимірного масиву "pos" за вказаними координатами;

2) якщо на вказаній позиції немає фігури, повертається значення null;

o) Winner:

1) цей метод викликається при завершенні гри та оголошенні переможця;

2) встановлюється значення gameOver на true;

3) відповідний текст переможця виводиться на екрані шляхом увімкнення компонента TextMeshPro;

4) показується повідомлення про переможця та текст для перезавантаження гри.

Код класу "Game" наведено у Додатку А.2

3.2.2 Клас "ChessMan"

В класі ChessMan оголошені різні поля та методи, які використовуються для керування фігурами на шаховій дошці. Основні методи цього класу наступні:

- Activate. Цей метод викликається для активації фігури. Він налаштовує початкову позицію фігури, прив'язує спрайт до фігури та встановлює значення гравця (чорний або білий);

- SetCoords. Цей метод встановлює позицію фігури на шаховій дошці на основі її координат xBoard та yBoard. Він використовує ці координати для обчислення позиції фігури в просторі;

- GetXBoard, GetYBoard. Ці методи повертають значення координат xBoard та yBoard відповідно;

- SetXBoard, SetYBoard. Ці методи встановлюють значення координат xBoard та yBoard відповідно;

- OnMouseUp. Цей метод викликається, коли користувач відпускає кнопку миші після натискання на фігуру. Він перевіряє стан гри і поточного гравця, та ініціює відображення доступних ходів для фігури;

- DestroyMovePlates. Цей метод знищує всі об'єкти "movePlate" на дошці;
- InitiateMovePlates. Цей метод ініціює відображення доступних ходів для фігури, в залежності від її типу. Він викликає різні підметоди в залежності від типу фігури;
 - LineMovePlate. Цей метод відображає доступні ходи для фігури, які можуть бути здійснені вздовж прямих ліній. Він перевіряє наявність вільних клітинок та можливість зробити ход;
 - DiagonalMovePlate. Цей метод відображає доступні ходи для фігури, які можуть бути здійснені по діагоналі. Він перевіряє наявність вільних клітинок та можливість переміщення по діагоналі;
 - KnightMovePlate. Цей метод відображає доступні ходи для фігури "коня". Він перевіряє наявність вільних клітинок та можливість переміщення фігури у вигляді "Г"-подібного ходу;
 - PawnMovePlate. Цей метод відображає доступні ходи для фігури "пішака". Він перевіряє наявність вільних клітинок та можливість переміщення фігури вперед, а також можливість атаки на ворожу фігуру по діагоналі;
 - PassantMovePlate. Цей метод відображає доступні ходи для фігури "пішака" у випадку можливості атаки "в проході". Він перевіряє наявність можливості виконати цей особливий хід;
 - CastlingMovePlate. Цей метод відображає доступні ходи для фігури "короля" у випадку можливості рокіровки. Він перевіряє наявність умов для виконання рокіровки та відображає ходи, які можуть бути здійснені в результаті рокіровки;
 - Promotion. Цей метод викликається, коли фігура "пішак" досягає кінця шахової дошки і має можливість зробити просування. Він відкриває вікно для вибору фігури, на яку пішак буде замінений;
 - Move і Capture. Ці методи використовуються для переміщення фігури на нову позицію або захоплення ворожої фігури. Вони також оновлюють

координати фігури на дошці та перевіряють, чи не настане шах або мат після цього ходу;

– `undoMove`. Цей метод скасовує останній хід, відновлюючи попередні координати фігури та стан дошки.

Кодкласу "ChessMan" наведено у Додатку А.3

3.2.3 Клас "MovePlate"

Клас `MovePlate` відповідає за логіку та обробку подій пов'язаних з можливими ходами шахових фігур. Основні методи цього класу наступні:

а) `reference`. Змінна `reference` зберігає посилання на об'єкт фігури, яка має здійснити хід на певну позицію;

б) `chessMan`. Змінна `chessMan` містить посилання на компонент `ChessMan`, який представляє шахову фігуру, пов'язану з даним об'єктом `MovePlate`;

в) `matrixX` і `matrixY`. Ці змінні визначають позицію на шаховій дошці, до якої можна здійснити хід;

г) `attack`. Логічна змінна `attack` вказує, чи є цей хід атакуючим ходом;

д) `start`. Цей метод викликається при запуску гри. Якщо хід є атакуючим, то змінюється колір спрайту на червоний;

е) `OnMouseUp`. Цей метод викликається при відпусканні кнопки миші. Виконується логіка здійснення ходу:

1) отримується посилання на контролер гри;

2) якщо шахова фігура, яка здійснює хід, є першим ходом, то змінюється відповідна властивість фігури;

3) якщо хід є атакуючим, перевіряється, чи атаковано короля. Якщо так, викликається метод `Winner` контролера гри для оголошення переможця;

4) знищується фігура, яка знаходиться на позиції, куди здійснюється хід;

- 5) встановлюється порожня позиція в масиві pos контролера гри;
- 6) встановлюються нові координати фігури, яка здійснює хід;
- 7) встановлюється посилання на фігуру у відповідну позицію масиву pos контролера гри;
- 8) змінюється хід наступного гравця;
- 9) викликається метод DestroyMovePlates фігури, яка здійснила хід, для знищення позначок можливих ходів;
- ж) SetCoords. Цей метод встановлює координати позиції, до якої можна здійснити хід;
- з) SetReference. Цей метод встановлює посилання на фігуру, яка здійснює хід;
- и) GetReference. Цей метод повертає посилання на фігуру, яка здійснює хід.

Код класу "MovePlate" наведено у Додатку А.5

3.2.4 Клас "Menu"

Клас "Menu" відповідає за керування головним меню гри. Він містить кілька методів, які відповідають за різні функції меню. Розглянемо кожен метод окремо:

- Start. Перевіряє, чи триває гра. Якщо так, то вимикає головне меню;
- StartGame. Вимикає головне меню. Використовується при натисканні кнопки "Start";
- ExitButton: Активує панель підтвердження виходу. Також вмикає текстовий елемент "QuitText". Використовується при натисканні кнопки "Exit";
- ConfirmExit: Викликає функцію Application.Quit для виходу з додатку. Використовується при підтвердженні виходу з панелі підтвердження;
- CancelExit: Скасовує панель підтвердження виходу. Використовується при скасуванні виходу;

- InGameButton: Активує панель меню в грі. Використовується при натисканні кнопки "InGame";
- CancelButton: Скасовує показ панелі меню в грі. Використовується при скасуванні меню в грі;
- BackButton: Активує панель підтвердження повернення до головного меню. Також вмикає текстовий елемент "ConfirmBack". Використовується при натисканні кнопки "Back";
- ConfirmBackToMenu: Вмикає головне меню і вимикає панель меню в грі та панель підтвердження повернення до головного меню. Використовується при підтвердженні повернення до головного меню;
- CancelBack: Скасовує панель підтвердження повернення до головного меню. Використовується при скасуванні повернення до головного меню;
- ConfirmRestartGame: Встановлює значення IsPlay на true, що дозволяє розпочати нову гру. Завантажує поточний рівень знову для перезапуску гри. Використовується при підтвердженні перезапуску гри;
- CancelRestart: Скасовує панель підтвердження перезапуску гри. Використовується при скасуванні перезапуску гри;
- RestartButton: Активує панель підтвердження перезапуску гри. Також вмикає текстовий елемент "ConfirmRestart". Використовується при натисканні кнопки "Restart".

Загалом, клас "Menu" забезпечує функціональність для керування різними діями головного меню, такими як початок гри, вихід з гри, повернення до меню та перезапуск гри.

Кодкласу "Menu" наведено у Додатку А.1

3.2.5 Клас "PossibleMoves"

Клас "PossibleMoves" відповідає за генерацію можливих ходів для фігур на шаховій дошці. Основні елементи цього класу включають:

- possibleMovesArray та KingpossibleMovesArray. Двовимірні масиви розміром 8x8, які зберігають інформацію про можливі ходи для фігур та короля відповідно;
- IsPlay. Булева змінна, яка вказує, чи триває гра;
- GenerateKingMoves та GenerateMoves1. UnityEvent, які використовуються для реєстрації та виклику подій генерації ходів для короля та інших фігур;
- controller. Посилання на об'єкт, який містить компонент Game, який відповідає за управління грою;
- OnGenerateKingMoves та OnGenerateMoves. Статичні методи, які викликають відповідні події генерації ходів короля та інших фігур;
- Update. Метод, який викликається кожен кадр і перевіряє натискання певних клавіш для виконання певних дій, таких як генерація ходів, перевірка на шах і т.д;
- Start. Метод, який викликається під час запуску програми і реєструє методи генерації ходів для короля та інших фігур відповідно до відповідних подій;
- GeneratePossibleMove. Метод, який генерує можливі ходи для фігур на шаховій дошці. Використовуючи двовимірний масив ChessMans, він перебирає всі фігури на дошці і викликає відповідні методи для генерації можливих ходів для кожної фігури;
- GeneratePossibleMovesKing. Метод, який генерує можливі ходи для короля. Використовуючи двовимірний масив ChessMans, він перебирає всі фігури на дошці і викликає відповідні методи для генерації можливих ходів для короля;

- `CheckForCheck`. Статичний метод, який перевіряє, чи перебуває король в шаху. Він отримує інформацію про положення короля з об'єкту `Game`, а потім перевіряє, чи може хтось з фігур противника атакувати короля.

Код класу "Game" наведено у Додатку А.4.

3.3 Висновки до розділу 3

У розділі, присвяченому опису всіх методів, що присутні у програмі, були розглянуті наступні аспекти:

- описані всі методи, які використовуються у програмі. Кожен метод має свою визначену функціональність та виконує певні дії;
- кожен метод був детально описаний, включаючи його назву, вхідні та вихідні параметри (якщо є), а також його ціль та функцію у програмі;
- наведені приклади використання методів у коді програми, щоб продемонструвати, як вони можуть бути застосовані на практиці.

У результаті даного розділу була надана повна інформація про всі методи, які використовуються у програмі, та їх роль у загальній функціональності програми. Це допоможе розробникам або користувачам програми краще розуміти, які дії виконуються кожним методом та як вони взаємодіють між собою.

4 ЕКСПЛУАТАЦІЯ ТА ТЕСТУВАННЯ ПРОГРАМИ

4.1 Опис програми

Програма "Шахи" - це додаток, який дозволяє користувачам насолоджуватися грою в шахи на мобільному пристрої. Вона створена для тих, хто цінує логічні головоломки і стратегічне мислення.

Основна мета програми - надати користувачам можливість грати в шахи проти інших користувачів.

Програма має простий і зрозумілий інтерфейс, що дозволяє легко керувати фігурами та здійснювати ходи. Крім того, вона надає додаткові функції, такі як відображення можливих ходів.

Для використання програми потрібно мати мобільний пристрій на базі iOS або Android.

Завдяки програмі "Шахи" ви зможете насолоджуватися захоплюючою грою в шахи, розвивати своє логічне мислення та стратегічні навички. Розпочніть свою шахову пригоду прямо зараз!

Для підтримки роботи програми необхідно, щоб апаратна частина включала процесор (2 ядра, не менше 2,1 ГГц), оперативну пам'ять (від 100мб), жорсткий диск з наявним вільним простором (від 200 Мб).

4.2 Експлуатація програми

На головному екрані додатку (рис. 4.1) ви знайдете елементи керування, такі як кнопки "Start" і "Exit". При натисканні кнопки "Start" відкриється ігрове поле з фігурами, і ви зможете розпочати гру (рис. 4.2). Для переміщення фігур достатньо натиснути на будь-яку фігуру та обрати один з доступних ходів (рис. 4.3).

Після завершення гри в шахи виводиться повідомлення про результат. На екрані з'являється повідомлення, що вказує на те, який гравець переміг або чи

завершилася гра внічию. Користувач може обрати опцію для початку нової гри або повернення до головного меню. (рис. 4.4).

На екрані з ігровим полем також доступна кнопка "Menu", при натисканні на яку відкриється панель вибору двох кнопок "Restart" та "Back to Menu".

При натисканні обох кнопок користувача попросять підтвердити свою дію. При натисканні кнопки "Restart" партія буде перезапущена(рис. 4.5), а при натисканні "Back to menu" ви повернетесь до головного меню (рис. 4.6). Щоб вийти з додатку, достатньо натиснути кнопку "Exit" і також підтвердити це (рис. 4.7).

Також у додатку реалізована функція перетворення пішки. Коли пішка досягає останньої клітинки шахової дошки, відбувається її перетворення в іншу фігуру. При перетворенні пішки з'являється меню вибору фігури, де гравець може обрати бажану фігуру для заміни. Це дає гравцю можливість стратегічно використовувати перетворену фігуру для досягнення своїх цілей в грі. Вибір фігури може вплинути на подальші ходи та розвиток партії, роблячи перетворення пішки важливим етапом у грі на шаховій дошці. (рис. 4.8).

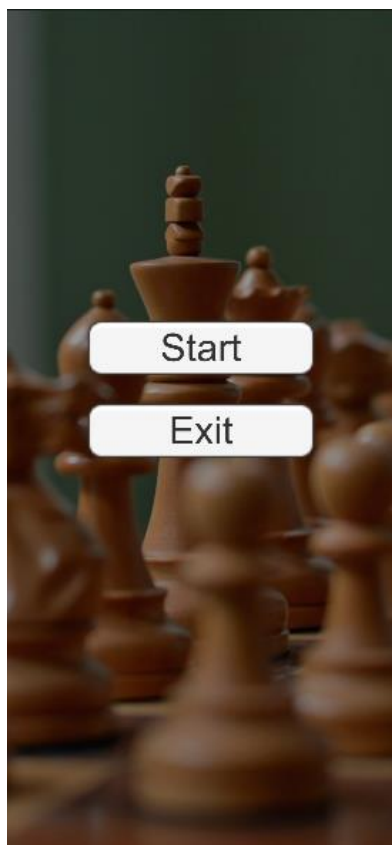


Рисунок 4.1 – Головне меню програми



Рисунок 4.2 – Ігрове поле



Рисунок 4.3 – Візуалізація доступних ходів фігури



Рисунок 4.4 – Повідомлення при завершенні партії



Рисунок 4.5 – Підтвердження дії "Restart"



Рисунок 4.6 – Підтвердження дії "Back To Menu"



Рисунок 4.7 – Підтвердження виходу з програми



Рисунок 4.8 – Меню вибору фігури для перетворення

4.3 Тестування і опис принципів експлуатації програми

Тестування програми виконувалось на основі перевірки реалізації функцій шляхом співставлення очікуваних результатів роботи і відображуваних результатів. Це дозволило встановити в результаті повну відповідність і вважати результати тестування успішними.

Отримані результати під час зокрема тестування інтерфейсу програми при виконанні цих функцій були використані для ілюстрації роботи з програмою в подальшому.

4.4 Висновки за розділом 4

В розділі описано процес роботи користувача з програмою. Наведені ілюстрації ігрового поля, варіанти повідомлень в процесі гри та інші робочі вікна програми.

Виконання тестування роботи додатку відбулось успішно.

ВИСНОВКИ

Виконаний аналіз предметної області шахових ігор дозволив виокремити основні поняття та методології, пов'язані з розробкою шахового додатку. Було проведено огляд існуючих програмних аналогів, включаючи веб-сайти та мобільні додатки. У результаті, з урахуванням їх обмежень, виявлено особливості розробленого шахового додатку.

Розробка шахового додатку була виконана з використанням фреймворку Unity та мови програмування C#. Була визначена структура даних, базові вимоги до програми, розроблена архітектура програми та алгоритм роботи на основі цих вимог. Були реалізовані відповідні компоненти шахової дошки, фігур, правил гри та логіки переміщення фігур.

Розроблений шаховий додаток покращує зручність гри в шахи, дозволяючи користувачам грати у будь-який зручний для них час та місце. Він також надає можливість вдосконалити навички гри в шахи, навчитися стратегії та тактики гри.

У підсумку, розроблений шаховий додаток дозволяє користувачам насолоджуватися грою в шахи, покращуючи їх досвід та взаємодію з цією захоплюючою грою.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Алгоритми шахових програм [Electronic resource]. – Access mode: <http://acm.mipt.ru/twiki/bin/view/Algorithms/ArlazarovChessAlgo>.
2. Новотельнов Н. А. Знайомтеся: шахи / Н.А. Новотельнов, О. Шека. – Леніздат, 1976. – С. 256.
3. Chaika, P., & Chaika, P. (2021). Шахи набирають популярність. У чому особливість гри в шахи. Науково-популярний Журнал Пізнавайка [Електронний ресурс]. – Режим доступу: <https://www.poznavayka.org/uk/nauka-i-svit/shahinabirayut-populyarnist-u-chomu-osoblivist-gri-v-shahi/>.
4. Chess com [Electronic resource]. – Access mode: <https://www.chess.com>
5. FIDE International Chess Federation [Electronic resource]. – Access mode: <https://handbook.fide.com/chapter/E012018>.
6. John Riccitiello sets out to identify the engine of growth for UnityTechnologies [Electronic resource]. – Access mode: <https://web.archive.org/web/20150117174953/http://venturebeat.com/2014/10/23/john-riccitiello-sets-out-to-identify-the-engine-of-growth-for-unity-technologies-interview/>.
7. Lichess org [Електронний ресурс] – Режим доступу: <https://lichess.org/>.
8. Microsoft [Electronic resource]. – Access mode: <https://visualstudio.microsoft.com/>.
9. StackOverflow [Electronic resource]. – Access mode: <https://stackoverflow.com/questions/tagged/unity-game-engine?tab=Newest>.
10. Unity Asset Store [Electronic resource]. – Access mode: <https://assetstore.unity.com/>
11. Unity Answers [Electronic resource]. – Access mode: <https://answers.unity.com/>.

12. Unity CEO on the prevalence of Unity [Electronic resource]. – Access mode: <https://techcrunch.com/2018/09/05/unity-ceo-says-half-of-all-games-are-built-on-unity/>.
13. Unity Learn [Electronic resource]. – Access mode: <https://learn.unity.com/>.
14. Unity Manual [Electronic resource]. – Access mode: <https://docs.unity3d.com/Manual/index.html>
15. Unity Manual Animation Section [Electronic resource]. – Access mode: <https://docs.unity3d.com/540/Documentation/Manual/AnimationSection.html>.
16. Unity Technologies [Electronic resource]. – Access mode: <https://unity.com/products/unity-platform>.

Додаток А
Текст програми

A.1 Клас Menu:

```

public class Menu : MonoBehaviour
{
    public GameObject canvas, ConfirmPanel, InGameMenu;
    public GameObject ConfirmRestart, ConfirmBack;
    public GameObject PromotionWhite, PromotionBlack;
    public GameObject controller;
    public void Start()
    {
        if (PossibleMoves.IsPlay)
        {
            canvas.SetActive(false);
        }
    }

    public void StartGame()
    {
        canvas.SetActive(false);
    }

    #region
    public void ExitButton()
    {
        ConfirmPanel.SetActive(true);

        GameObject.FindGameObjectWithTag("QuitText").GetComponent<TextMeshProUGUI
        I>().enabled = true;
    }

    public void ConfirmExit()
    {
        Application.Quit();
        Debug.Log("Exit");
    }

    public void CancelExit()
    {
        ConfirmPanel.SetActive(false);
    }
    #endregion
    //exit

```

```

#region
public void InGameButton()
{
    InGameMenu.SetActive(true);

}
public void CancelButton()
{
    InGameMenu.SetActive(false);
}
#endregion
// ingamemenu

```

```

#region

public void BackButton()
{
    ConfirmBack.SetActive(true);
}

```

```

GameObject.FindGameObjectWithTag("ConfirmBack").GetComponent<TextMeshPro
UGUI>().enabled = true;

```

```

}
public void ConfirmBackToMenu()
{
    canvas.SetActive(true);
    InGameMenu.SetActive(false);
    ConfirmBack.SetActive(false);
}

```

```

public void CancelBack()
{
    ConfirmBack.SetActive(false);
}

```

```

#endregion

```

```

//backbutton

```

```

#region

```

```

public void ConfirmRestartGame()
{
    PossibleMoves.IsPlay = true;
}

```

```

        SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex,
LoadSceneMode.Single);
    }
    public void CancelRestart()
    {
        ConfirmRestart.SetActive(false);
    }
    public void RestartButton()
    {
        ConfirmRestart.SetActive(true);

GameObject.FindGameObjectWithTag("ConfirmRestart").GetComponent<TextMeshPr
oUGUI>().enabled = true;
    }
    #endregion
    // restart

    #region
    public void OpenPromotionPanel(int x, int y)
    {
        ChessMan pawn = controller.GetComponent<Game>().GetArray(x, y);
        if (pawn != null)
        {
            bool isWhitePawn = pawn.isWhite();

            if (isWhitePawn)
            {
                OpenWhitePromotionPanel(x, y);
            }
            else
            {
                // OpenBlackPromotionPanel(x, y);
                Debug.Log("Promotion Black");
            }
        }
    }
    private void OpenWhitePromotionPanel(int x, int y)
    {
        PromotionWhite.SetActive(true);
    }
    /* private void OpenBlackPromotionPanel(int x, int y)
    {
        PromotionBlack.SetActive(true);
    } */
    #endregion

```

```

    // promotion
}

```

A.2 Клас Game:

```

public class Game : MonoBehaviour
{
    public GameObject ChessPiece;

    // Positions for each chesspiece
    private GameObject[,] pos = new GameObject[8, 8];
    private GameObject[] playerBlack = new GameObject[16];
    private GameObject[] playerWhite = new GameObject[16];

    private string currentPlayer = "black"; // ?
    private bool gameOver = false;

    public Transform Piece;
    private ChessMan[,] chessMans = new ChessMan[8,8];
    void Start()
    {
        playerWhite = new GameObject[]
        {
            Create("white_Rook", 0,0), Create("white_Knight", 1,0),
            Create("white_Bishop", 2,0),
            Create("white_Queen", 3,0), Create("white_King", 4,0),
            Create("white_Bishop", 5,0),
            Create("white_Knight", 6,0), Create("white_Rook", 7,0),

```

```

        Create("white_Pawn", 0,1), Create("white_Pawn",
1,1),Create("white_Pawn", 2,1),Create("white_Pawn", 3,1),
        Create("white_Pawn", 4,1),Create("white_Pawn",
5,1),Create("white_Pawn", 6,1),Create("white_Pawn", 7,1)
    };

    playerBlack = new GameObject[]
    {
        Create("black_Rook", 0,7), Create("black_Knight", 1,7),
Create("black_Bishop", 2,7),
        Create("black_Queen", 3,7), Create("black_King", 4,7),
Create("black_Bishop", 5,7),
        Create("black_Knight", 6,7), Create("black_Rook", 7,7),

        CreatePawn("black_Pawn", 0,6,true), CreatePawn("black_Pawn",
1,6,true),CreatePawn("black_Pawn", 2,6,true),CreatePawn("black_Pawn", 3,6,true),
        CreatePawn("black_Pawn", 4,6,true),CreatePawn("black_Pawn",
5,6,true),CreatePawn("black_Pawn", 6,6,true),CreatePawn("black_Pawn", 7,6,true)
    };

    for (int i = 0; i < playerBlack.Length; i++)
    {
        SetPosition(playerBlack[i]);
        SetPosition(playerWhite[i]);
    }

}

public GameObject Create(string name, int x, int y)
{

```

```

        GameObject obj = Instantiate(ChessPiece, new Vector3(0, 0, -1),
Quaternion.identity);
        ChessMan cm = obj.GetComponent<ChessMan>();
        cm.name = name;
        cm.SetXBoard(x);
        cm.SetYBoard(y);
        cm.Activate();
        obj.transform.SetParent(Piece);
        return obj;
    }

    public GameObject CreatePawn(string name, int x, int y, bool isFirstMove)
    {

        GameObject obj = Instantiate(ChessPiece, new Vector3(0, 0, -1),
Quaternion.identity);
        ChessMan cm = obj.GetComponent<ChessMan>();
        cm.name = name;
        cm.SetXBoard(x);
        cm.SetYBoard(y);
        cm.firstMove = isFirstMove;
        cm.Activate();
        obj.transform.SetParent(Piece);
        return obj;
    }

    public void SetPosition(GameObject obj)
    {
        ChessMan cm = obj.GetComponent<ChessMan>();
        pos[cm.GetXBoard(), cm.GetYBoard()] = obj;
    }

```

```

        chessMans[cm.GetXBoard(), cm.GetYBoard()] =
obj.GetComponent<ChessMan>();
    }

    public ChessMan[,] GetBoard()
    {
        return chessMans;
    }

    public void SetPositionEmpty(int x, int y)
    {
        pos[x, y] = null;
    }

    public GameObject GetPosition(int x, int y)
    {
        return pos[x, y];
    }

    public bool PositionOnBoard(int x, int y)
    {
        if (x < 0 || y < 0 || x >= pos.GetLength(0) || y >= pos.GetLength(1)) return
false;

        return true;
    }

    public string GetCurrentPlayer()
    {
        return currentPlayer;
    }

    public bool IsGameOver()
    {
        return gameOver;
    }

```

```
}
```

```
public void NextTurn()
{
    if (currentPlayer == "white")
    {
        currentPlayer = "black";
    }
    else
    {
        currentPlayer = "white";
    }
} // debug
```

```
public void Update()
{
    if (gameOver == true && Input.GetMouseButtonDown(0))
    {
        gameOver = false;

        SceneManager.LoadScene("Game");
    }
}

public ChessMan GetArray(int x, int y)
{
    if (pos[x, y] != null)
        return pos[x, y].GetComponent<ChessMan>();
    else return null;
}

public void Winner (string playerWinner)
```

```
{
    gameOver = true;
```

```
GameObject.FindGameObjectWithTag("WinnerText").GetComponent<TextMeshProUGUI>().enabled = true;
```

```
GameObject.FindGameObjectWithTag("WinnerText").GetComponent<TextMeshProUGUI>().text = playerWinner + " is the Winner";
```

```
GameObject.FindGameObjectWithTag("RestartText").GetComponent<TextMeshProUGUI>().enabled = true;
```

```
}
```

```
}
```

A.3 Клас ChessMan:

```
public class ChessMan : MonoBehaviour
```

```
{
```

```
    // References
```

```
    public GameObject controller;
```

```
    public GameObject movePlate;
```

```
    // Positions
```

```
    private int xBoard = -1;
```

```
    private int yBoard = -1;
```

```
    public int moveCount;
```

```
    public bool firstMove = true;
```

```

ChessMan[,] ChessMans = new ChessMan[8, 8];
// white player or black
public string player, fen;

    public Sprite black_Queen, black_Knight, black_Bishop, black_King,
black_Rook, black_Pawn,
        white_Queen, white_Knight, white_Bishop, white_King,
white_Rook, white_Pawn;
    private bool iswhite;

    bool isWhiteChess = false;

    public void Activate()
    {
        controller = GameObject.FindGameObjectWithTag("GameController");

        SetCoords();

        switch (this.name)
        {
            case "black_Queen": this.GetComponent<SpriteRenderer>().sprite =
black_Queen; player = "black"; break;
            case "black_Knight": this.GetComponent<SpriteRenderer>().sprite =
black_Knight; player = "black"; break;
            case "black_Bishop": this.GetComponent<SpriteRenderer>().sprite =
black_Bishop; player = "black"; break;
            case "black_King": this.GetComponent<SpriteRenderer>().sprite =
black_King; player = "black"; break;
            case "black_Rook": this.GetComponent<SpriteRenderer>().sprite =
black_Rook; player = "black"; break;

```

```

        case "black_Pawn": this.GetComponent<SpriteRenderer>().sprite =
black_Pawn; player = "black"; break;

        case "white_Queen": this.GetComponent<SpriteRenderer>().sprite =
white_Queen; player = "white"; break;
        case "white_Knight": this.GetComponent<SpriteRenderer>().sprite =
white_Knight; player = "white"; break;
        case "white_Bishop": this.GetComponent<SpriteRenderer>().sprite =
white_Bishop; player = "white"; break;
        case "white_King": this.GetComponent<SpriteRenderer>().sprite =
white_King; player = "white"; break;
        case "white_Rook": this.GetComponent<SpriteRenderer>().sprite =
white_Rook; player = "white"; break;
        case "white_Pawn": this.GetComponent<SpriteRenderer>().sprite =
white_Pawn; player = "white"; break;
    } // Привязка спрайтов к фигурам

}

#region
public void SetCoords()
{
    float x = xBoard;
    float y = yBoard;

    x *= 3.66f;
    y *= 3.66f;

    x += -1.66f;
    y += -1.66f;

```

```

        this.transform.position = new Vector3(x, y, -1.0f);
    }

    public int GetXBoard()
    {
        return xBoard;
    }

    public int GetYBoard()
    {
        return yBoard;
    }

    public void SetXBoard(int x)
    {
        xBoard = x;
    }

    public void SetYBoard(int y)
    {
        yBoard = y;
    }

    #endregion

    private void OnMouseUp()
    {
        GameObject controller =
GameObject.FindGameObjectWithTag("GameController");
        ChessMan chessman = GetComponent<ChessMan>();
        GenerateFen();
        if (!controller.GetComponent<Game>().IsGameOver() &&
controller.GetComponent<Game>().GetCurrentPlayer() != player)

```

```

    {
        DestroyMovePlates();
        InitiateMovePlates();
    }
}

```

```

public void DestroyMovePlates()
{
    GameObject[] movePlates =
GameObject.FindGameObjectsWithTag("MovePlate");
    for (int i = 0; i < movePlates.Length; i++)
    {
        Destroy(movePlates[i]);
    }
}

```

```

public void InitiateMovePlates()
{
    switch (this.name)
    {
        case "black_Queen":
        case "white_Queen":
            LineMovePlate(1, 0);
            LineMovePlate(0, 1);
            LineMovePlate(1, 1);
            LineMovePlate(-1, 0);
            LineMovePlate(0, -1);
            LineMovePlate(-1, -1);
            LineMovePlate(-1, 1);
            LineMovePlate(1, -1);

```

```
break;
```

```
case "black_Knight":
```

```
case "white_Knight":
```

```
    LMovePlate();
```

```
    break;
```

```
case "black_Bishop":
```

```
case "white_Bishop":
```

```
    LineMovePlate(1, 1);
```

```
    LineMovePlate(1, -1);
```

```
    LineMovePlate(-1, 1);
```

```
    LineMovePlate(-1, -1);
```

```
    break;
```

```
case "black_King":
```

```
case "white_King":
```

```
    SurroundMovePlate();
```

```
    break;
```

```
case "black_Rook":
```

```
case "white_Rook":
```

```
    LineMovePlate(1, 0);
```

```
    LineMovePlate(0, 1);
```

```
    LineMovePlate(-1, 0);
```

```
    LineMovePlate(0, -1);
```

```
    break;
```

```
case "black_Pawn":
```

```
    PawnMovePlate(xBoard, yBoard - 1);
```

```

        break;

        case "white_Pawn":
            PawnMovePlate(xBoard, yBoard + 1);
            break;
    }
}

public void LineMovePlate(int xIncrement, int yIncrement)
{
    Game sc = controller.GetComponent<Game>();
    int x = xBoard + xIncrement;
    int y = yBoard + yIncrement;
    while (sc.PositionOnBoard(x, y) && sc.GetPosition(x, y) == null)
    {
        MovePlateSpawn(x, y);
        x += xIncrement;
        y += yIncrement;
    }
    if (sc.PositionOnBoard(x, y) && sc.GetPosition(x,
y).GetComponent<ChessMan>().player != player)
    {
        MovePlateAttackSpawn(x, y);
    }
}

public void LMovePlate()
{
    PointMovePlate(xBoard + 1, yBoard + 2);
    PointMovePlate(xBoard - 1, yBoard + 2);
    PointMovePlate(xBoard + 2, yBoard + 1);
    PointMovePlate(xBoard + 2, yBoard - 1);

```

```

    PointMovePlate(xBoard + 1, yBoard - 2);
    PointMovePlate(xBoard - 1, yBoard - 2);
    PointMovePlate(xBoard - 2, yBoard + 1);
    PointMovePlate(xBoard - 2, yBoard - 1);
}
public void SurroundMovePlate()
{
    PossibleMoves.OnGenerateKingMoves();
    Game sc = controller.GetComponent<Game>();
    if (sc.PositionOnBoard(xBoard - 0, yBoard + 1))
        if (PossibleMoves.KingpossibleMovesArray[xBoard - 0, yBoard + 1] == 3)
            PointMovePlate(xBoard - 0, yBoard + 1);

    if (sc.PositionOnBoard(xBoard - 0, yBoard - 1))
        if (PossibleMoves.KingpossibleMovesArray[xBoard - 0, yBoard - 1] ==
3)
            PointMovePlate(xBoard - 0, yBoard - 1);

    if (sc.PositionOnBoard(xBoard - 1, yBoard - 1))
        if (PossibleMoves.KingpossibleMovesArray[xBoard - 1, yBoard - 1] ==
3)
            PointMovePlate(xBoard - 1, yBoard - 1);

    if (sc.PositionOnBoard(xBoard - 1, yBoard - 0))
        if (PossibleMoves.KingpossibleMovesArray[xBoard - 1, yBoard - 0] ==
3)
            PointMovePlate(xBoard - 1, yBoard - 0);

    if (sc.PositionOnBoard(xBoard - 1, yBoard + 1))

```

```

        if (PosibleMoves.KingpossibleMovesArray[xBoard - 1, yBoard + 1] ==
3)
            PointMovePlate(xBoard - 1, yBoard + 1);

        if (sc.PositionOnBoard(xBoard + 1, yBoard - 1))
            if (PosibleMoves.KingpossibleMovesArray[xBoard + 1, yBoard - 1] ==
3)
                PointMovePlate(xBoard + 1, yBoard - 1);

        if (sc.PositionOnBoard(xBoard + 1, yBoard - 0))
            if (PosibleMoves.KingpossibleMovesArray[xBoard + 1, yBoard - 0] ==
3)
                PointMovePlate(xBoard + 1, yBoard - 0);

        if (sc.PositionOnBoard(xBoard + 1, yBoard + 1))
            if (PosibleMoves.KingpossibleMovesArray[xBoard + 1, yBoard + 1] ==
3)
                PointMovePlate(xBoard + 1, yBoard + 1);

    }
    public void PointMovePlate(int x, int y)
    {
        Game sc = controller.GetComponent<Game>();
        if (sc.PositionOnBoard(x, y))
        {
            GameObject cp = sc.GetPosition(x, y);
            if (cp == null)
            {
                MovePlateSpawn(x, y);
            }
        }
    }

```

```

        else if (cp.GetComponent<ChessMan>().player != player)
        {
            MovePlateAttackSpawn(x, y);
        }
    }
}

```

```

public bool isWhite()

```

```

{
    Game game = controller.GetComponent<Game>();
    if (game.GetCurrentPlayer() == "white")
    {
        return true;
    }
    else
    {
        return false;
    }
}

```

```

private int Forward()

```

```

{
    if (GetComponent<SpriteRenderer>().sprite == white_Pawn)
    {
        return +1;
    }
    else if (GetComponent<SpriteRenderer>().sprite == black_Pawn)
    {
        return -1;
    }
}

```

```

        return +0;
    }
    public void PawnMovePlate(int x, int y)
    {
        Game sc = controller.GetComponent<Game>();
        if (sc.PositionOnBoard(x, y))
        {
            bool iswhite = isWhite();
            int forward = Forward();

            // Первый ход пешки (может двигаться на 2 клетки вперед)
            if (firstMove && sc.GetPosition(x, y) == null && sc.GetPosition(x, y +
forward) == null)
            {
                MovePlateSpawn(x, y);
                MovePlateSpawn(x, y + forward);
            }
            // Обычное движение на одну клетку вперед
            else if (sc.GetPosition(x, y) == null)
            {
                MovePlateSpawn(x, y);
            }

            if (sc.PositionOnBoard(x + 1, y) && sc.GetPosition(x + 1, y) != null &&
sc.GetPosition(x + 1, y).GetComponent<ChessMan>().player != player)
            {
                MovePlateAttackSpawn(x + 1, y);
            }

            if (sc.PositionOnBoard(x - 1, y) && sc.GetPosition(x - 1, y) != null &&
sc.GetPosition(x - 1, y).GetComponent<ChessMan>().player != player)

```

```

    {
        MovePlateAttackSpawn(x - 1, y);
    }

    // Превращение пешки
    if ((iswhite && y == 7) || (!iswhite && y == 0))
    {
        controller.GetComponent<Menu>().OpenPromotionPanel(x, y);

        // ?

        controller.GetComponent<Menu>().PromotionWhite.SetActive(false);
    }
}

public void MovePlateSpawn(int matrixX, int matrixY, bool isAttack = false)
{
    float x = matrixX;
    float y = matrixY;
    x *= 3.66f;
    y *= 3.66f;

    x += -1.66f;
    y += -1.66f;
    GameObject mp = Instantiate(movePlate, new Vector3(x, y, -3.0f),
Quaternion.identity);
    MovePlate mpScript = mp.GetComponent<MovePlate>();
    mpScript.SetReference(gameObject);
    mpScript.SetCoords(matrixX, matrixY);
}

```

```

        mp.GetComponent<MovePlate>().chessMan =
GetComponent<ChessMan>();
    }

    public void MovePlateAttackSpawn(int matrixX, int matrixY)
    {
        float x = matrixX;
        float y = matrixY;

        x *= 3.66f;
        y *= 3.66f;

        x += -1.66f;
        y += -1.66f;
        GameObject mp = Instantiate(movePlate, new Vector3(x, y, -3.0f),
Quaternion.identity);

        MovePlate mpScript = mp.GetComponent<MovePlate>();
        mpScript.attack = true;
        mpScript.SetReference(gameObject);
        mpScript.SetCoords(matrixX, matrixY);
        mp.GetComponent<MovePlate>().chessMan =
GetComponent<ChessMan>();
    }

    public string GenerateFen() // 2-мерный массив -> fen
    {
        fen = "";
        ChessMan[,] ChessMans = new ChessMan[8, 8];
        for (int i2 = 7; i2 >= 0; i2--)

```

```

{
    for (int i = 0; i < 8; i++)
    {

        ChessMans[i, i2] = controller.GetComponent<Game>().GetArray(i,
i2);

        if (ChessMans[i, i2] != null)
        {
            switch (ChessMans[i, i2].name)
            {
                case "white_King":
                    fen += "K";
                    break;
                case "black_King":
                    fen += "k";
                    break;

                case "white_Queen":
                    fen += "Q";
                    break;
                case "black_Queen":
                    fen += "q";
                    break;

                case "white_Rook":
                    fen += "R";
                    break;
                case "black_Rook":
                    fen += "r";
                    break;
            }
        }
    }
}

```

```
        case "white_Bishop":
            fen += "B";
            break;
        case "black_Bishop":
            fen += "b";
            break;

        case "white_Knight":
            fen += "N";
            break;
        case "black_Knight":
            fen += "n";
            break;

        case "white_Pawn":
            fen += "P";
            break;
        case "black_Pawn":
            fen += "p";
            break;
    }
}
else fen += "1";
}
fen += "/";
}
// Debug.Log(fen);

return fen;
```

```
} // ?
```

```
#region Generate Moves
```

```
public void PawnMovePlateGenerate(int localy)
```

```
{
```

```
    int x = xBoard;
```

```
    int y = yBoard + localy;
```

```
    Game sc = controller.GetComponent<Game>();
```

```
    if (sc.PositionOnBoard(x, y))
```

```
    {
```

```
        if (sc.PositionOnBoard(x + 1, y))
```

```
        {
```

```
            if (player == "white" && isWhite())
```

```
                PossibleMoves.possibleMovesArray[x + 1, y] = 2;
```

```
            else if (player == "black" && !isWhite())
```

```
                PossibleMoves.possibleMovesArray[x + 1, y] = 2;
```

```
        }
```

```
        if (sc.PositionOnBoard(x - 1, y))
```

```
        {
```

```
            if (player == "white" && isWhite())
```

```
                PossibleMoves.possibleMovesArray[x - 1, y] = 2;
```

```
            else if (player == "black" && !isWhite())
```

```
                PossibleMoves.possibleMovesArray[x - 1, y] = 2;
```

```
        }
```

```
    }
```

```
}
```

```
public void PointMovePlateGenerate(int localx, int localy)
{
    int x = xBoard + localx;
    int y = yBoard + localy;

    Game sc = controller.GetComponent<Game>();
    if (sc.PositionOnBoard(x, y))
    {
        GameObject cp = sc.GetPosition(x, y);
        if (cp == null)
        {
            if (player == "white" && isWhite())
                PossibleMoves.possibleMovesArray[x, y] = 3;
            else if (player == "black" && !isWhite())
                PossibleMoves.possibleMovesArray[x, y] = 3;
        }
        else if (cp.GetComponent<ChessMan>().player != player)
        {
            if (player == "white" && isWhite())
                PossibleMoves.possibleMovesArray[x, y] = 3;
            else if (player == "black" && !isWhite())
                PossibleMoves.possibleMovesArray[x, y] = 3;
        }
    }
}

public void LineMovePlateGenerate(int xIncrement, int yIncrement)
{
```

```

Game sc = controller.GetComponent<Game>();
int x = xBoard + xIncrement;
int y = yBoard + yIncrement;

while (sc.PositionOnBoard(x, y) && sc.GetPosition(x, y) == null)
{
    if (player == "white" && isWhite())
        PossibleMoves.possibleMovesArray[x, y] = 2;
    else if (player == "black" && !isWhite())
        PossibleMoves.possibleMovesArray[x, y] = 2;

    x += xIncrement;
    y += yIncrement;
}

if (sc.PositionOnBoard(x, y) && sc.GetPosition(x,
y).GetComponent<ChessMan>().player != player)
{
    if (player == "white" && isWhite())
        PossibleMoves.possibleMovesArray[x, y] = 2;
    else if (player == "black" && !isWhite())
        PossibleMoves.possibleMovesArray[x, y] = 2;
}
}

#endregion

public void KingMovePlateGenerate(int localx, int localy)
{
    int x = xBoard + localx;

```

```

int y = yBoard + localy;

Game sc = controller.GetComponent<Game>();
if (sc.PositionOnBoard(x, y))
{
    GameObject cp = sc.GetPosition(x, y);
    if (cp == null)
    {
        if (player == "black" && isWhite())
            PossibleMoves.KingpossibleMovesArray[x, y] = 3;
        else if (player == "white" && !isWhite())
            PossibleMoves.KingpossibleMovesArray[x, y] = 3;
    }
    else if (cp.GetComponent<ChessMan>().player != player)
    {
        if (player == "black" && isWhite())
            PossibleMoves.KingpossibleMovesArray[x, y] = 3;
        else if (player == "white" && !isWhite())
            PossibleMoves.KingpossibleMovesArray[x, y] = 3;
    }
}

}

#region

public void PromotionWhiteQueen(int x, int y)
{
    ChessMan pawn = controller.GetComponent<Game>().GetArray(x, y);
    Destroy(pawn.gameObject);
}

```

```

        controller.GetComponent<Game>().Create("white_Queen", x, y);

    }

    public void PromotionWhiteRook(int x, int y)
    {
        ChessMan pawn = controller.GetComponent<Game>().GetArray(x, y);
        Destroy(pawn.gameObject);
        controller.GetComponent<Game>().Create("white_Rook", x, y);
    }

    public void PromotionWhiteBishop(int x, int y)
    {
        ChessMan pawn = controller.GetComponent<Game>().GetArray(x, y);
        Destroy(pawn.gameObject);
        controller.GetComponent<Game>().Create("white_Bishop", x, y);
    }

    public void PromotionWhiteKnight(int x, int y)
    {
        ChessMan pawn = controller.GetComponent<Game>().GetArray(x, y);
        Destroy(pawn.gameObject);
        controller.GetComponent<Game>().Create("white_Knight", x, y);
    }

    #endregion
}

```

A.4 Клас PossibleMoves:

```

public class PosibleMoves : MonoBehaviour
{
    public static int[,] posibleMovesArray = new int[8, 8];
    public static int[,] KingpossibleMovesArray = new int[8, 8];
    public static bool IsPlay;
    public static UnityEvent GenerateKingMoves = new UnityEvent();
    public static UnityEvent GenerateMoves1 = new UnityEvent();

    public GameObject controller;

    public static void OnGenerateKingMoves()
    {
        GenerateKingMoves.Invoke();
    }
    public static void OnGenerateMoves()
    {
        {
            GenerateMoves1.Invoke();
        }
    }
    private void Update()
    {
        if (Input.GetKeyDown(KeyCode.H))
        {
            GeneratePosibleMove();
            // GeneratePossibleMovesKing();
        }

        if (Input.GetKeyDown(KeyCode.L))
        {

```

```

        CheckForCheck();
    }

    if (Input.GetKeyDown(KeyCode.F))
    {
        // GeneratePossibleMovesKing();
    }
}

public void Start()
{
    GenerateMoves1.AddListener(GeneratePosibleMove);
    GenerateKingMoves.AddListener(GeneratePossibleMovesKing);
}

void GeneratePosibleMove()
{
    ChessMan[,] ChessMans = new ChessMan[8, 8];

    for (int i = 0; i < 8; i++)
    {
        for (int b = 0; b < 8; b++)
        {
            posibleMovesArray[i, b] = 0;
        }
    }

    for (int i = 0; i < 8; i++)
    {
        for (int i2 = 0; i2 < 8; i2++)

```

```

{
    ChessMans[i, i2] = controller.GetComponent<Game>().GetArray(i,
i2);

    if (ChessMans[i, i2] != null)
    {
        switch (ChessMans[i, i2].name)
        {
            case "black_King":
            case "white_King":
                ChessMans[i, i2].PointMovePlateGenerate(-0, 1);
                ChessMans[i, i2].PointMovePlateGenerate(-0, -1);
                ChessMans[i, i2].PointMovePlateGenerate(-1, -1);
                ChessMans[i, i2].PointMovePlateGenerate(-1, -0);
                ChessMans[i, i2].PointMovePlateGenerate(-1, 1);
                ChessMans[i, i2].PointMovePlateGenerate(1, -1);
                ChessMans[i, i2].PointMovePlateGenerate(1, -0);
                ChessMans[i, i2].PointMovePlateGenerate(1, 1);
                break;

            case "black_Queen":
            case "white_Queen":
                ChessMans[i, i2].LineMovePlateGenerate(1, 0);
                ChessMans[i, i2].LineMovePlateGenerate(0, 1);
                ChessMans[i, i2].LineMovePlateGenerate(1, 1);
                ChessMans[i, i2].LineMovePlateGenerate(-1, 0);
                ChessMans[i, i2].LineMovePlateGenerate(0, -1);
                ChessMans[i, i2].LineMovePlateGenerate(-1, -1);
                ChessMans[i, i2].LineMovePlateGenerate(-1, 1);
                ChessMans[i, i2].LineMovePlateGenerate(1, -1);
                break;
        }
    }
}

```

```
case "black_Knight":
```

```
case "white_Knight":
```

```
    ChessMans[i, i2].PointMovePlateGenerate(1, 2);
    ChessMans[i, i2].PointMovePlateGenerate(-1, 2);
    ChessMans[i, i2].PointMovePlateGenerate(2, 1);
    ChessMans[i, i2].PointMovePlateGenerate(2, -1);
    ChessMans[i, i2].PointMovePlateGenerate(1, -2);
    ChessMans[i, i2].PointMovePlateGenerate(-1, -2);
    ChessMans[i, i2].PointMovePlateGenerate(-2, 1);
    ChessMans[i, i2].PointMovePlateGenerate(-2, -1);
    break;
```

```
case "black_Bishop":
```

```
case "white_Bishop":
```

```
    ChessMans[i, i2].LineMovePlateGenerate(1, 1);
    ChessMans[i, i2].LineMovePlateGenerate(1, -1);
    ChessMans[i, i2].LineMovePlateGenerate(-1, 1);
    ChessMans[i, i2].LineMovePlateGenerate(-1, -1);
    break;
```

```
case "black_Rook":
```

```
case "white_Rook":
```

```
    ChessMans[i, i2].LineMovePlateGenerate(1, 0);
    ChessMans[i, i2].LineMovePlateGenerate(0, 1);
    ChessMans[i, i2].LineMovePlateGenerate(-1, 0);
    ChessMans[i, i2].LineMovePlateGenerate(0, -1);
    break;
```

```
case "black_Pawn":
```

```

        ChessMans[i, i2].PawnMovePlateGenerate(-1);
        break;

    case "white_Pawn":
        ChessMans[i, i2].PawnMovePlateGenerate(1);
        break;
    }
}
}

GeneratePossibleMovesKing();

}

void GeneratePossibleMovesKing()
{
    ChessMan[,] ChessMans = new ChessMan[8, 8];
    for (int i = 0; i < 8; i++)
    {
        for (int b = 0; b < 8; b++)
        {
            KingpossibleMovesArray[i, b] = 0;
        }
    }

    for (int i = 0; i < 8; i++)
    {
        for (int i2 = 0; i2 < 8; i2++)
        {

```

```

ChessMans[i, i2] = controller.GetComponent<Game>().GetArray(i,
i2);

if (ChessMans[i, i2] != null)
{
    switch (ChessMans[i, i2].name)
    {
        case "black_King":
        case "white_King":
        {
            ChessMans[i, i2].KingMovePlateGenerate(0, 1);
            ChessMans[i, i2].KingMovePlateGenerate(0, -1);
            ChessMans[i, i2].KingMovePlateGenerate(-1, -1);
            ChessMans[i, i2].KingMovePlateGenerate(-1, 0);
            ChessMans[i, i2].KingMovePlateGenerate(-1, 1);
            ChessMans[i, i2].KingMovePlateGenerate(1, -1);
            ChessMans[i, i2].KingMovePlateGenerate(1, 0);
            ChessMans[i, i2].KingMovePlateGenerate(1, 1);
        }
        break;
    }
}

}

for (int x = 0; x < 8; x++)
{
    for (int y = 0; y < 8; y++)
    {
        if (possibleMovesArray[x, y] != 0 && possibleMovesArray[x, y] != 3)
        {

```



```

        kingY = y;
        break;
    }
}
else if (ChessMans[x, y].player == "black" && ChessMans[x,
y].isWhite())
{
    if (ChessMans[x, y].name == "black_King")
    {
        kingX = x;
        kingY = y;
        break;
    }
}
if (kingX != -1 && kingY != -1)
    break;
}
}
}
if (possibleMovesArray[kingX, kingY] == 2)

{
    Debug.Log("King is in check!");
    return true;
}
else
{
    Debug.Log("King is safe.");
    return false;
}

```

```

        return false;
    }
}

```

A.5 Клас MovePlate

```

public class MovePlate : MonoBehaviour
{
    public GameObject controller;
    GameObject reference = null;
    public ChessMan chessMan;
    int matrixX;
    int matrixY;
    public bool attack = false;

    public void Start()
    {
        if (attack)
        {
            gameObject.GetComponent().color = new Color(1.0f,
0.0f, 0.0f, 1.0f);
        }
    }
    public void OnMouseUp()
    {
        controller = GameObject.FindGameObjectWithTag("GameController");

        if (chessMan.firstMove) { chessMan.firstMove = false; }

        if (attack) // атака на короля, нужна проверка

```

```

    {
        GameObject cp =
controller.GetComponent<Game>().GetPosition(matrixX, matrixY);
        if (cp.name == "white_King")
controller.GetComponent<Game>().Winner("black");
        if (cp.name == "black_King")
controller.GetComponent<Game>().Winner("White");
        Destroy(cp);
    }

controller.GetComponent<Game>().SetPositionEmpty(reference.GetComponent<Chess
Man>().GetXBoard(), reference.GetComponent<ChessMan>().GetYBoard());
        reference.GetComponent<ChessMan>().SetXBoard(matrixX);
        reference.GetComponent<ChessMan>().SetYBoard(matrixY);
        reference.GetComponent<ChessMan>().SetCoords();
        controller.GetComponent<Game>().SetPosition(reference);
        controller.GetComponent<Game>().NextTurn();
        reference.GetComponent<ChessMan>().DestroyMovePlates();
    }
public void SetCoords(int x, int y)
{
    matrixX = x;
    matrixY = y;
}
public void SetReference(GameObject obj)
{
    reference = obj;
}
public GameObject GetReference()
{

```

```
        return reference;  
    }  
}
```

ДОДАТОК Б
Слайди Презентації



Рисунок Б.1– Слайд№1

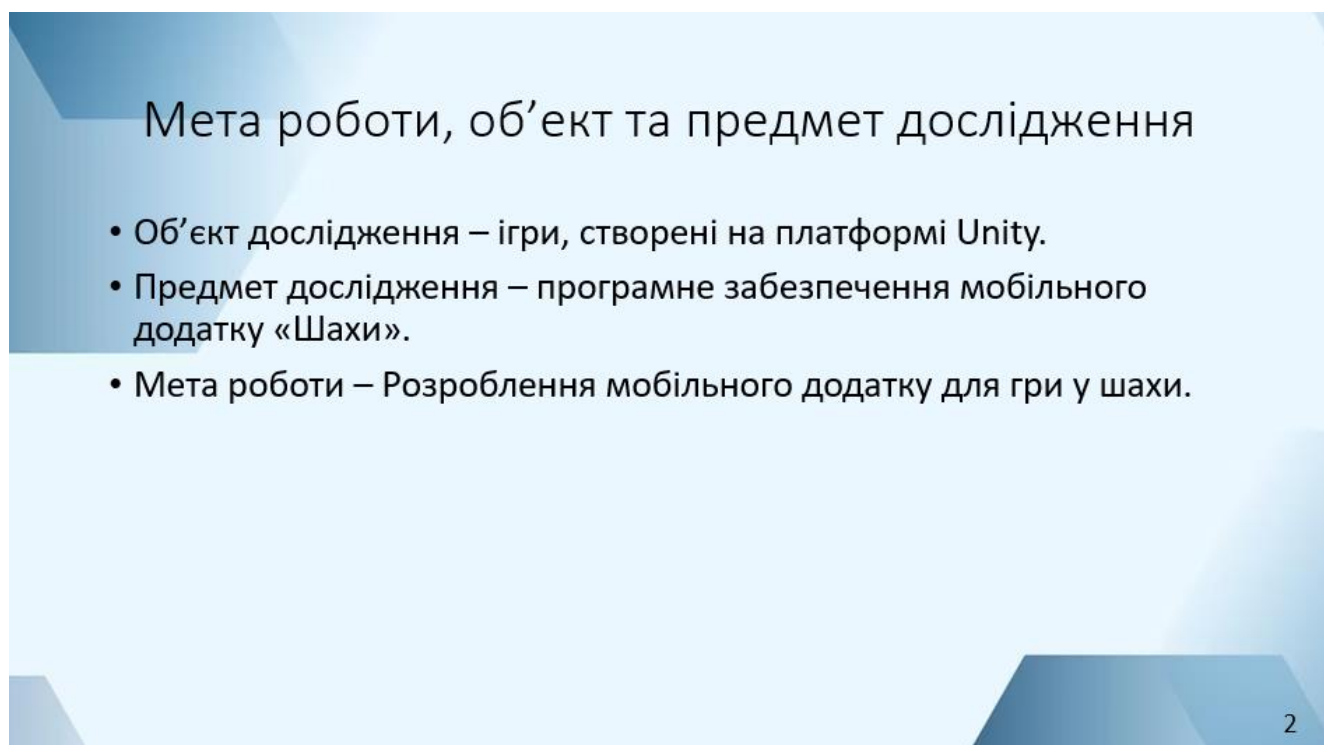


Рисунок Б.2– Слайд№2

Завдання до роботи

- Розробити логіку гри в шахи, включаючи правила руху фігур, перевірку на шах та мат, обробку ходів гравців і перемогу.
- Розробити зручний інтерфейс для взаємодії з додатком, включаючи візуалізацію дошки, фігур, кнопок та інших елементів управління.
- Забезпечити можливість гри та вирішення задач на різних платформах, таких як комп'ютери, мобільні пристрої тощо.
- Протестувати та оптимізувати додаток для забезпечення його ефективності та стабільності.

3

Рисунок Б.3– Слайд№3

Аналіз аналогів

Аналоги	Chess.com	Chess Tactics Pro	Chess Free	Lichess.org
Критерії порівняння				
Користувачі	Має велике співтовариство, що дозволяє знайти багато суперників та партнерів для гри	Відсутність мультиплеєра: додаток не має можливості грати з іншими гравцями онлайн, а фокусується лише на навчанні	Додаток дозволяє грати в шахи на різних рівнях складності або з іншими гравцями	Надає можливість грати в режимі онлайн з іншими користувачами з усього світу, що дозволяє знайти суперників будь-якого рівня
Режими	Гра з комп'ю-тером, гра з іншими гравцями онлайн, навчання шахам	Відсутні додаткові режими, лише гра	дозволяє вибирати рівень складності гри з комп'ютером	Різноманітні шахові пазли, ігри проти користувачів та комп'ютеру та інші тренувальні режими
Навчальні матеріали	Багато навчальних матеріалів, пазли та статей	Більше 1000 рівнів складності для вивчення шахових тактик та покращення навичок	не має вбудованих навчальних матеріалів для початківців	різноманітні навчальні ресурси: шахові пазли, статті та тренувальні режими
Безкоштовна версія / Платні опції	Обмежені можливості безкоштовної версії: деякі функції та матеріали можуть бути доступні лише для платних користувачів			є безкоштовним для встановлення та використання, а також відкритим для спільноти
Реклама	у безкоштовній версії може містити			Не містить
Особливі технічні вимоги	в режимі онлайн необхідне швидке інтернет з'єднання	стабільне та через відсутність мультиплеєра інтернет не потрібен і добре працює навіть на старих версіях телефонів	Не має особливих технічних вимог	

4

Рисунок Б.4– Слайд№4

Порівняння мов програмування

Мова	C#	C++	Java	Python	JavaScript
Критерії оцінювання					
Об'єктно-орієнтованість	Так	Так	Так	Так	Ні
Платформи та фреймворки	.NET, Unity, Xamarin	Unreal Engine, Qt	Android, Spring	Pygame, Django	Node.js, React
Синтаксис	Подібний до C/C++/Java	Подібний до C	Подібний до C/C++	Простий, експресивний	Синтаксично різноманітний, динамічний
Спільнота розробників	Велика, активна	Велика, активна	Велика, активна	Велика, активна	Велика, активна
Приклади використання	Розробка ігор, веб-додатків, десктопних додатків	Розробка ігор, системного програмування, вбудованих систем	Розробка ігор, веб-додатків, мобільних додатків	Розробка ігор, веб-додатків, наукові обчислення	Розробка веб-додатків, браузерних ігор

5

Рисунок Б.5— Слайд №5

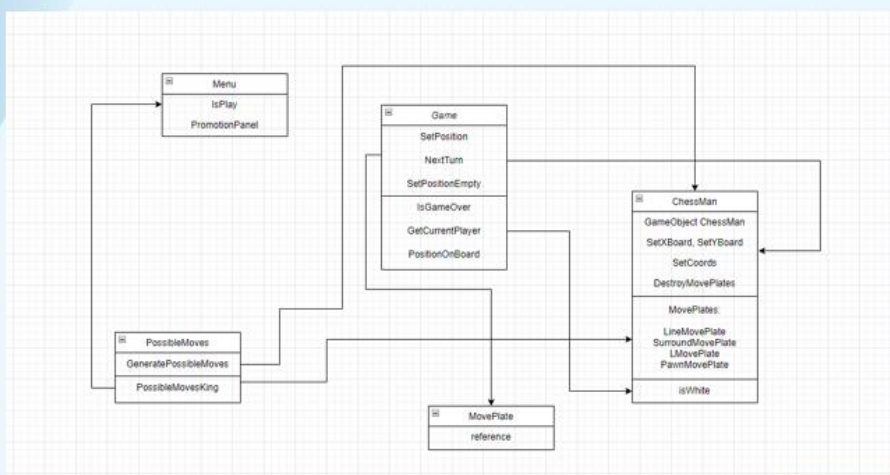
Порівняння інтегрованих середовищ розробки

Критерії	Visual Studio	IntelliJ IDEA	Eclipse	PyCharm	Xcode
Підтримувані мови програмування	C#, C++, Java, Python та інші	Java, Kotlin, Scala та інші	Java, C++, PHP та інші	Python, JavaScript, HTML/CSS та інші	Swift, Objective-C, C++ та інші
Інтеграція з платформами	.NET, Unity, Xamarin	Android, Spring, JavaFX	Android, Java, JavaScript та інші	PHP, Django, Flask, Web та інші	iOS, macOS, watchOS, tvOS
Функціональність та продуктивність	Висока	Висока	Середня	Висока	Середня
Підтримка плагінів	Так	Так	Так	Так	Ні
Налагоджування та профілювання	Повна підтримка	Повна підтримка	Повна підтримка	Повна підтримка	Повна підтримка
Спільнота розробників	Велика, активна	Велика, активна	Велика, активна	Велика, активна	Велика, активна
Підтримка систем контролю версій	Git, SVN, інші	Git, SVN, інші	Git, SVN, інші	Git, SVN, інші	Git, SVN, інші

6

Рисунок Б.6— Слайд №6

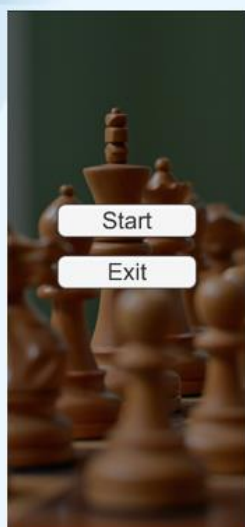
Схема залежностей класів



7

Рисунок Б.7– Слайд№7

Скріншоти основних сцен додатку



Головне меню програми



Ігрове поле

8

Рисунок Б.8– Слайд№8

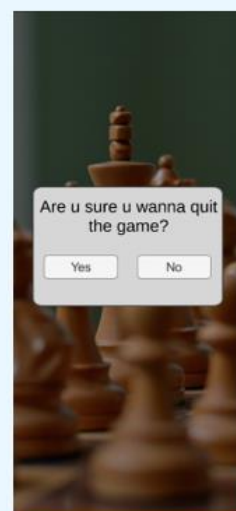
Скріншоти з гри



9

Рисунок Б.9– Слайд№9

Зовнішній вигляд меню та інших кнопок



10

Рисунок Б.10– Слайд№10

Висновки

- Було проведено огляд існуючих програмних аналогів, включаючи веб сайти та мобільні додатки. У результаті, з урахуванням їх обмежень, виявлено особливості розробленого шахового додатку.
- Була визначена структура даних, базові вимоги до програми, розроблена архітектура програми та алгоритм роботи на основі цих вимог. Були реалізовані відповідні компоненти шахової дошки, фігур, правил гри та логіки переміщення фігур.
- Розроблений шаховий додаток покращує зручність гри в шахи, дозволяючи користувачам грати у будь-який зручний для них час та місце. Він також надає можливість вдосконалити навички гри в шахи, навчитися стратегії та тактики гри.

11

Рисунок Б.11– Слайд№11

Дякую за увагу!

12

Рисунок Б.12– Слайд№12